Assignment Presentation Framework for CS1 Programming Problems

Rita Garcia
University of Adelaide
Adelaide, South Australia
rita.garcia@adelaide.edu.au

***Abstract*—A Computer Science programming assignment is a core assessment tool used by educators to measure students' understanding of programming concepts. A well-structured programming assignment can assist students in comprehending a problem and help them develop problem-solving skills. This research focuses on the presentation of well-structured programming assignments to help with students' understanding of the problem, supporting educators in the development of these assignments. To create well-structured programming assignments, we performed a literature review to identify design treatments that help students better understand introductory programming (CS1) assignments. An illustration study was then used to test the assignments from the identified design treatments. The illustration study collected students' perspectives on how the design treatments improved the understanding of the assignment design. From this work, we developed a framework from the identified design treatments. The results for the illustration study showed the design treatments helped students identify the problem's requirements and realise what is required to solve the problem. Though our work was examined within the CS1 context, our findings may be useful to similar assignment contexts, such as helping students identify requirements for solving problems.**

***Index Terms*—Assessment; Understanding; Framework**

I. INTRODUCTION

Decades of research in the field of education has shown that an assignment's presentation can influence students' learning and behaviours. Inappropriate forms of presentations can promote the adoption of surface-level learning approaches [22], which can influence students' approaches to learning [11]. Students might adopt concentrated bouts of study behaviour for summative assessments, such as examinations [19]. When students place more importance on grades than learning [8], they might adopt *cue-seeking strategies*, strategies used to discover clues or hints to determine what materials will be available on examinations [36]. Exam-focused study behaviours can negatively impact the learning approach [23].

Providing students with a range of activities, both formative and summative, can produce higher student scores [19]. In introductory programming (CS1) courses, programming assessments in conjunction with other assessment activities, such as quizzes, in-class work, and examinations can evaluate students' knowledge of learning materials [22]. Programming assignments can reinforce new concepts [17], which may help reduce students' exam-focus behaviours and study habits. Ensuring assignments are designed for better understanding can play a critical role in engaging students in purposeful learning, and support their completion of programming assignments [46].

Students learning could be impacted when presented with poorly constructed assignments, resulting in misconceptions that could 'impede students from acquiring new concepts in computer programming' [46]. The misconceptions can also result in lower marks that can negatively impact students' self-efficacy and attrition rates [38]. Well-constructed programming assignments can help discourage students from using poor learning behaviours, such as seeking cue-seeking strategies, and help them with complex concepts [17].

This research is focused on presentation aspects that support the development of well-constructed programming assignments, and is conducted in two parts. The first part involves a literature review that identifies peer-reviewed conference and journal papers that provide design treatments that can help students better understand CS1 programming assignments. The second part uses these design treatments to form a framework that further examined this framework by inviting students to participate in an illustration study. The illustration study was conducted at a large university in Australia, using a mixed-methods approach of a questionnaire and narrative interviews. The research questions guiding this study are:

- *RQ1: Are there existing assignment design treatments that can help students better understand programming problems?*
- *RQ2: What are students' perceptions of the assignment design based on the presented treatments?*

II. BACKGROUND

Programming assignments are assessment tools used to measure students' understanding of learning objectives [35], and are designed to exercise software development processes, such as identifying requirement specifications, and designing, planning and implementing solutions. Students believe CS assignments are difficult, contributing to their diminishing interest in CS over time [16]. Poorly constructed assignments can affect how students perceive their programming abilities, which can lead to low self-efficacy [26] and contribute to the attrition rates [7]. Lowered self-efficacy has been shown to negatively impact students' performance [31] and their engagement in the course [24]. To address low self-efficacy and attrition rates, practitioners can construct well-defined assignments [16].

Poorly constructed assignments can also contribute to students' adoption of Poor Learning Tendencies (PLTs) [5], habits used by students to compensate for their lack of understanding [5]. PLTs were identified and defined by the Project to

Enhance Effective Learning (PEEL) project, a project devoted to providing teaching techniques to practitioners that promote Good Learning Behaviours (GLBs). PLTs include impulsive attention, inadequate monitoring, and lack of reflective thinking when solving the problem. Impulsive attention, inadequate monitoring, and lack of reflective thinking are tendencies students might apply when processing the assignment description. Examples include focusing on the surface aspects [1] of the assignment description, which could lead to misunderstandings. PLTs might contribute to students' shallow understanding about concepts, which could later affect their ability to plan [20] and debug [34] programs. A well-defined assignment could help students engage in positive behaviours, such as Self-Regulated Learning [51], potentially increasing program comprehension [40]. Better understanding of the assignment can reduce software defects [2] that could lead to higher assignment completion rates [9] and lower attrition rates [38].

This research aims to help students avoid Poor Learning Tendencies through well-defined assignments. We use existing publications, peer-reviewed conference and journal papers, that identify design treatments to help form well-defined assignments. Prior assignment design research has focused on the inconsistencies in the presentation of assessment tasks [44] and for software engineering assignments [25], but this research focuses on the presentation of CS1 programming assignments to help students develop their understanding of the assignment.

III. LITERATURE REVIEW

A. Search Criteria

A mapping literature review [10] was used to identify design treatments for CS1 assignments. A meta-search engine was used to construct and send search queries to publication libraries, such as the ACM Digital Library, IEEE Xplore Library, and Springer. The search criteria focused on publications in peer-reviewed journals and conferences that were published after 2000, presenting design treatments for individual CS1 programming assignments. The inclusion criteria selected publications, peer-reviewed conference and journal papers, that discussed design treatments for CS1 assignments and might not be the focus of the study, but might be an observation made from performing the research. An exclusion criteria was developed to remove theoretical papers and any studies that did not focus on CS1 assignments.

The applied selection criteria used the following search string: (*Homework* OR *Exercises* OR *'Problem Description'* OR *Assignment* OR *Comprehension*) AND (*CS1* OR *'Computer Science'* OR *'Introductory Programming'*). The meta-search engine applied the search string to all search fields, such as *Title*, *Subject*, and *User* tags. Some keywords in the search string were selected from general terms in publications [26], [28] discovered early in this research. Additional keywords were added to the string to narrow the scope to CS1 courses.

B. Framework Design

After completing the literature review, we used a theoretical multi-level assignment educational design pattern to present the design treatments. The design pattern incorporates 'a variety of educational objectives into a single assignment by including the concepts on multiple knowledge and process levels' [28]. The multi-level refers to the pattern's layers of scaffolding, allowing students to independently learn as they progress in their studies. The design pattern has six sections that includes the assignment's problem, context, implementation, and solution. We used the implementation section to form the template presentation that includes the assignment's context, problem description, and hints that help students better understand the problem and concepts.

To collate the design treatments into the framework, we examined its purpose. Based on the purpose, we mapped its placement into *Context*, *Program Description*, or *Hints*. If the evaluation of the design treatment suggested the placement to more than one category, the design treatment's publication was further examined to decide the right category. The collation process produced a framework that retains the three *Implementation* sections for easier identification of the design treatments.

C. Literature Review Results

The initial selection criteria, described in Section III-A, generated 2169 publications, but the number of publications was reduced to 92 after applying the meta-search engine's *Peer-reviewed* filter to select peer-reviewed conference and journal papers. We validated the accuracy of the *Peer-reviewed* filter by applying it to another query that generated accurate results. An exclusion criteria was then applied to the 92 publications' abstracts and titles. The exclusion criteria reduced the results to 11 publications meeting the selection criteria because theoretical papers and any studies that did not focus on CS1 assignments were removed from the literature review.

A quality evaluation step [10] was performed to ensure the appropriate publications were selected, which included reviewing the abstracts to determine whether the design treatments contribute to the publication's results. The evaluation step produced 11 publications, which had their citations examined to determine if any were relevant for this review. These citations brought the final literature result to 14 publications, listed in Table 1. The table lists the 14 publications in alphabetical order by title and summarises the design treatments. In the following section, the design treatments are presented within the context of the assignment presentation framework.

D. Framework Results

The assignment presentation framework, shown in Figure 1, is divided into the three sections: *Context*, *Program Description*, and *Hints*. The design treatments are grouped together, such as "Core Objectives", listing their benefits and a brief description. These framework sections are in the order the design treatments would be presented in the assignment. For example, "Core Objectives" in the *Context* section is applied before the assignment's problem description.

The prevalence of design treatments within an assignment can define the scaffolding level that suits the students' abilities. A high number of design treatments would indicate a highly scaffolded assignment that is designed for students needing

Publications	Recommended Design Treatments from Publications
A code snippet library for CS1 [32]	Contains external code to help students learn more complex concepts, allowing practitioners to build interesting problems.
Characteristics of programming exercises that lead to poor learning tendencies: Part II [14]	Includes familiar tasks to promote learning reinforcement; Presents high-level justification to previously learned concepts.
Developing real-world programming assignments for CS1 [43]	Incorporates layers to challenge students and make the problem interesting and fun to complete.
Experiences in threading UML throughout a computer science program [39]	Includes UML diagrams within the assignment to better explain how control structures can be applied.
Exploring factors that include computer science introductory course students to persist in the major [6]	Provides interesting concepts with context; Provides instructional material related to students' background knowledge for better retention.
Extreme apprenticeship method in teaching programming for beginners [48]	Provides smaller goals and relevant examples with clean design treatments to help students find the starting point to solve the problem.
Interesting basic problems for CS1 [18]	Provides realistic problem to help increase motivation.
Note to self: Make assignments meaningful [30]	Provides socially-relevant material that students find interesting to keep them more engaged in solving the problem.
Novice programmers and the problem description effect [12]	Includes contextualisation that can help with motivation and engagement.
Principles for designing programming exercises to minimise poor learning behaviours in students [13]	Emphasises the key point of the problem by using bold text; Minimise overloading the student with unfamiliar concepts so they can remain focus.
Scaffolding for multiple assignment projects for CS1 & CS2 [29]	Provides design elements from lessons learned in the classroom.
Research directions for teaching programming online [41]	Presents published materials to construct acceptable practices to teach in online learning environments.
Visualizations in preparing for programming exercise sessions [3]	Includes visualisations into the problem that can help students with no programming experience better understand the problem.
Why the rhetoric of CS programming assignments matter [49]	Provides real-world context in the problem to keep students motivated.

Table 1: List of Publications from the Literature Review with Recommended Design Treatments

additional guidance. Highly scaffolded assignments might be used early in the semester when CS1 students are starting to program. Fewer design treatments would indicate a low scaffolded assignment designed for students to solve the problem in a more independent learning environment, allowing them to take control of their learning and to solve learning objectives with less guidance [27]. Practitioners can select the design treatments for assignments, and decide on the scaffolding level. The remainder of this section presented the publications and their design treatments that form the framework.

1) Context: The *Context* provides the basis for the programming problem. The design treatments in this section supports the presentation of required concepts necessary for completing the assignment. The literature review produced three publications [13], [14], [29] supporting context. The design treatments suggest presenting in the assignment the learning objectives and required concepts so that students will reflect on the previously learned materials. The learning objectives and concepts are presented in bold to focus the students and discourage them from concentrating on superficial concepts [13].

Two publications [13], [14] focus on minimising Poor Learning Tendencies (PLTs) by gathering feedback from students and tutors on how to improve the learning experience. The first [13] identifies PLTs related to superficial attention, impulse attention, and staying stuck. The results suggest assignments emphasise key points of the problem by highlighting the tasks to help reduce students from making hasty problem-solving decisions, and limiting unfamiliar concepts in a single assignment. The tutors observed students focusing on one question when encountering a lot of new concepts. Providing students with an outline on how to approach solving the problem can help them stay on the right problem-solving path. The second publication [14] focuses on PLTs related to non-retrieval, and lack of internal and external reflective thinking. The results suggest assignments contain familiar tasks to reinforces learning, and a broad view of the assignment's purpose to help associate the concepts taught in the course to the problem.

The third publication [29] presents lessons learned from scaffolding CS1 and CS2 Java-based assignment problem description, providing detailed descriptions of the requirements in early assignments, and gradually reducing these requirements. Students and tutors described their experiences using these assignments. Their feedback produced best practices in assignment design that help students better understand the problems.

2) Program Description: Seven publications [12], [18], [30], [41], [43], [48], [49] focused on the problem description, where majority (71.4%) discuss contextualisation as an approach to help with students' understanding. Contextualisation can help students develop a more accurate mental representation of the solution [41], [48]. One publication [12] showed that

Context				
Element Practice	**Headings** • Helps to identify sections	**Core Objectives** • Minimises hasty decisions • Identifies tasks	**Context Information** • Promotes long-term retention • Deduces practicality from background	
Treatment	• Clean guidelines [48]	• Boldface text [13] • Detailed requirements [29]	• Purpose for assignment [14] • Interleave assignments [29]	
Program Description				
Element Practice	**Assignment Complexity** • Increases confidence • Reduces Poor Learning Tendencies	**Subject Matter** • Increases motivation • Improves self-efficacy		
Treatment	• Limit number of new concepts [12] • Incrementally adjust the content [41] • List of smaller goals [48] • Encapsulate complexity in external modules [30]	• Interesting [18], socially-relevant [30] and realistic [49] problems • Concepts with familiar [43] and motivating [12] contexts		
Hints				
Element Practice	**Diagrams** • Helps with design strategy • Explains software design	**Code Segments** • Encourages coding standards • Helps develop algorithms	**Examples** • Learns from a worked solution	**Visualization** • Provides further insight • Develops mental model
Treatment	• UML diagrams [39]	• Code segments as references [32]	• Relevant examples [48]	• Visualisations [3]

Figure 1: Assignment Presentation Framework

contextualised content did not have an impact on students' overall grades, but it did have an impact on their motivation and interest in the assignment. Another publication [6] studied eight factors that predicted students persistence in CS. One factor was related to contextualised content, supporting persistence and enabled students to retain more of the information.

Another publication [18] found assignments with real-world context can better demonstrate complex concepts by relating the CS1 concepts with familiar contexts. Another publication [30] studied student motivation. A survey was administered to 200 CS1 students, asking for testimonials on practical and socially-relevant assignments. The results showed students preferring practical and socially relevant assignments, increasing their interest in the assignment. Another publication [49] conducted a survey with 81 students evaluating different contextualised instructional materials. The survey results showed majority (54%) of the students prefer real-world contextualisation because they felt these assignments were easier to understand and motivated them to complete the assignment. Another publication [43] performed a test in a classroom environment that used two programming assignments with real-world contexts. The participants found these assignments fun and appreciated their programs being applied to real external websites.

Examining teaching approaches has identified design treatments that support students understanding through the assignment presentation. A study [48] explored Extreme Apprenticeship (XA) to teach Java. XA is a teaching approach based on principles that guide and provide feedback to students during the problem-solving process. Outcomes showed reduced dropout rate with assignments using design treatments that provided smaller goals to help students identify intermediate goals, specified a starting point to help students with the first step in the problem-solving process, and provided output exemplars to emphasise what the problem is trying to achieve.

Another publication [43] concluded layered assignments can provide struggling students with additional support while challenging high-achieving students. There is a basic solution within a layered assignment, giving struggling students a path to solve and an opportunity to complete other complex layers. Achieving some of the layers can increase the students' self-confidence and self-efficacy, possibly encouraging them to attempt more challenging layers in future assignments.

3) Hints: The *Hints* section contains four publications [3], [32], [39], [48] that suggest design treatments that provide students with additional assistance to better understanding the problem. One publication [3] investigated the use of a visualisation tool for reviewing instructional materials. Students involved in this study were given a post-survey to provide their experiences with the visualisation tool. The study concluded visual and struggling students benefited from visualisations, resulting in better understanding of the problems and higher assignment grades. Another publication [39] conducted at the United States Military Academy at West Point evaluated the Unified Modelling Language (UML) in assignments. The study integrated UML use cases, class diagrams, and associations into assignments. The UML had replaced a previous ad-hoc visual

language used at the academy. Students included in this study appreciated UML because it allowed them to better describe their programs' intentions, such as the interactions between objects, identify workflow, and better understand state changes.

Another publication [32] suggested using an external codebase to encapsulate complex programming concepts that are beyond students' cognitive abilities. The codebases give students the opportunity to develop more complex and interesting assignments earlier in their learning. Three different codebases (manuals, memorisation, and libraries) were examined for C++ and Java. The manuals overwhelmed students, while memorisation was proven ineffective when they attempted to apply into their programming solutions. The code snippet library was the most successful in helping students develop algorithms, while reducing their frustration during development. The last publication [48] suggested using apprenticeship learning through examples that give students the opportunity to interact with interesting problems, which decreased dropout rates.

IV. Illustration Study

We adopt a mixed methods approach [15] to evaluate the assignment design framework. The mixed methods design uses quantitative data collection to better understand participants' prior programming experiences, while the qualitative phase performs more in-depth data collection.

A. Participants

We invited 95 students from a 12-week CS1 course (August 2018 Semester 1) to participate in the illustration study. Students were invited by an announcement posted in the Canvas Learning Management System. Four male students responded, and received a movie poster and a voucher for participating. The study was conducted in a lab environment with tutors and other students from the course developing and discussing assignments. The study was conducted on the day the assignment was due, giving them the opportunity to recall on how the assignment helped them during the development process.

B. Instructional Instrument Design

The instructional instrument is a programming assignment provided in the middle of the semester (week 6) for a CS1 course at a large university in Australia. The instructional instrument was the fifth programming assignment administered in the course. The prior four programming assignments were also developed with the framework and were highly scaffolded, providing students with core objectives, smaller goals, examples, hints, and visualisations. The fifth assignment gave us the opportunity to speak with students with prior exposure to assignments designed with the framework. As students progressed through the semester, and practised programming concepts, the design treatments in the assignments were reduced.

The instructional instrument was developed in the context of Soloway's *Rainfall Problem* [42] that averages rainfall over a period of days. The design treatments were selected based on students' exposure to programming concepts. The instructional instrument, shown in Figure 2, had students practising `functions` and `refactoring`. These programming concepts were introduced to students the same week as the assignment was administered, so these concepts were highly scaffolded. The `refactoring` design treatments provided the requirements and code to refactor [41]. These requirements were detailed [29] and presented in list format [48]. The boldface format [13] is used to help students identify the words **drawGradient** and **endValue** as the function and variable names. Students have previously practiced `loops` and `variables` in earlier assignments, so these concepts were presented with less scaffolding. The instructional instrument does not identify `loops` to construct the gradient scale. Instead, the looping process is described as *"The gradient starts with 0 (black) and lightens until it reaches **endValue**"*. Overall, the design treatments used in the instructional instrument were presenting subgoals in list format [47], using boldface text to identify important concepts [13], using visualisations [3], and presenting code fragments as reference [32].

Program Description
Below is code that draws a number of lines to form a grey-scale gradient. The gradient starts at 0 (black) and lightens until it reaches **endValue**. You should copy this code into the Processing IDE to see how it works. Try passing in different values to the **drawGradient** function to see how the output changes.

```
void setup(){
  size(400,400);
  background(255);
  drawGradient(255);
}

void drawGradient(int endValue){
  for(int i = 0; i <= endValue; i++){
    stroke(i);
    line(10,i,60,i);
  }
}
```

For your program, copy the above code and extend the **drawGradient** function to have the following parameters:

Parameter	Description	Possible Values
x	X-coordinate	
y	Y-coordinate	
vertical	Gradient orientation	True - vertical False - horizontal
endValue	Gradient end value	0 to 255
opacity	Gradient opacity	0 to 255
col	Gradient Colour	0 - greyscale 1 - red
widHei	Gradient width/height	

If any of the input parameters are invalid, your **drawGradient** function should display a warning, and not draw the gradient. See above for the valid values. Use a **for loop** to call **drawGradient** several times to design a pattern. The pattern should be produced by modifying the gradient attributes - i.e. end value, width/height, opacity, orientation (vertical/horizontal), or colour.

Figure 2: Instructional Instrument (Assignment 5)

C. Questionnaire

The questionnaire was designed to gather participants' prior programming experiences and their use of problem-solving strategies. The questionnaire was presented in paper format before the first interview and contained three questions: one Likert scale and two open-text questions. The Likert question asked participants to rank their prior programming experience. The two open-text questions asked for the number of years programming and list the previously used languages.

Two different analysis approaches were used on the responses: analysis of means, and thematic content analysis. Analysis of means was performed on the Likert scale question to identify the most frequent rating from the participants. Thematic content analysis [33] was performed on the two open-text questions, to identify and classify keywords that relate to the participants' prior programming experiences and languages. Thematic content analysis can be based on previously defined and emerging categories, where with emerging categories were added to the coding framework during the analysis process. Identifying participants' prior programming experience can help determine how much their backgrounds might influence their interpretation of programming assignments.

The spreadsheet containing the students' transcribed answers were imported into NVivo version 12 to identify coded themes. The analysis process started with two nodes to identify the two open-text questions. Another two nodes were created to identify no prior programming experience and programming languages. Any prior experience was defined as an emerging node with a label denoting the years of programming experience. Any programming languages were also identified as emerging nodes with the name of the programming language as the label. After the coding process, the themes were extracted from NVivo as a matrix to identify and present the coding frequencies.

D. Narrative Interviewing

This study adopts a narrative interview protocol [37] that collects students' perspectives on the assignment design. Narrative interviews enable students to share their experiences using the assignments while measuring their perceived understanding. In narrative interviewing, the interview begins with a broad question to encourage the student to share their experiences. For example, *"Can you tell me how you use the assignment description to better understand the problem?"* The interviewer then encourages additional narrative information with open-ended questions, such as *"Is there anything else you can say to ...?"* The narrative interview concludes with a question to recall points raised, such as *"What else can you say that helped you with the problem's context?"* These exemplar questions were used as guides for conducting the narrative interviews.

The students participated in two 30-minute interview sessions. The first session was conducted using the instructional instrument, shown in Figure 2, while the second was performed on a different assignment developed with the framework. During the interviews, audio-visual materials were collected for analysis. `SimpleScreenRecorder` [4] was the application used to collect the audio-visual materials and was available on the lab computers. Upon completion of the interviews, the recorded materials were professionally transcribed for further analysis and imported into NVivo for formal coding.

Directed content analysis [21] was used to code the interviews within pre-existing frameworks, while allowing for emerging categories to derive from the participants' responses. The analysis involved identifying interview segments representing themes on the design treatments influencing students' understanding of the assignment. The initial coding framework was based on the 14 design treatments within the assignment presentation framework. The initial coding also included three nodes that identified students' positive and negative responses to the design treatment, and identified general responses on the problem statement. In total, the initial coding framework contained 17 nodes: 14 design treatments, 2 positive/negative opinions, and 1 general statement. During the coding process, any emerging themes were assigned a new node. Upon coding completion, the coding frequencies created a matrix table exported from NVivo for discussion. Alternative forms reliability [15] compared the two interview sessions performed with the participants. The two sessions used different assignments, but measured the same variables in the coding framework that provided an alternative method of measuring internal consistency.

V. Illustration Study Results

A. Questionnaire

Table 2 shows the questionnaire results from the four participants. Two participants stated they had no prior experience, while a third, StudentC, stated he had Matlab. StudentD stated he learned Python, but did not elaborate on whether he learned Python through another CS course or self-taught. However, he did state he forgot all the programming concepts using Python. Table 2 also shows the participants' self-assessment, showing they ranked themselves as 'Inexperienced', the lowest level of programming experience in the questionnaire's Likert scale.

Student ID	Experience	Self-Assessment
StudentA	No prior experience	Inexperienced
StudentB	No prior experience	Inexperienced
StudentC	Matlab experience	Inexperienced
StudentD	Python, but forgot language constructs	Inexperienced

Table 2: Questionnaire Results from Participants

B. Interview Results

The overall results from the narrative interviews are presented in Table 3, but to demonstrate how the interviews were conducted and coded, we first present the sequential excerpts from StudentA's interview. The interviewer began by asking the student to draw on prior assignments to compare how the instructional instrument helped with their understanding. StudentA stated, *"I do definitely prefer this one because this one very clearly states what's needed. The last one (assignment) that I had a bit of difficulty with, it was just a paragraph of requirements. This one is very clearly stated what's needed."*

Theme	Result	Examples
Positive opinion	16 (84.21%)	*"This actually makes you sit down and methodically think about how you going to approach it."* StudentB (Positive opinion)
List format [48]	6 (31.58%)	*"I think having it set out as dot points really, clearly gives you all the points that you just need to address. It's just laying it out for you."* StudentB (Positive opinion)
Problem-solving skills	4 (21.05%)	*"I generally will reference back to it as I'm doing the assignment just to make sure that I'm ticking all the boxes that they want to see ticked."* StudentB (Positive opinion)
Boldface text format [13]	3 (15.79%)	*"I don't understand why that's (function name) is in bold. That is odd to me."* StudentA (Negative opinion) *"Maybe if its emboldened, people are paying attention to where the conditions are in a statement."* StudentB (Positive opinion)
Goal decomposition [48]	3 (15.79%)	*"was a bit more step by step, this one. So it had it really laid out for what you need to kind of add into the program."* StudentC (Positive opinion)
Negative opinion	3 (15.79%)	*"Actually had confusion with this problem that I got help with earlier, I started asking about it."* StudentC (Positive opinion)
Code fragments [32]	2 (10.53%)	*"structure is useful, so you sort of know how to start if off, because it's very intimidating to look at that and create it."* StudentD (Positive opinion)
General statements	2 (10.53%)	*"The last one (assignment) that I had a bit of difficult with, it was just a paragraph of requirements. This one is very clearly stated what's needed."* StudentA (Positive opinion)
Paragraph format	2 (10.53%)	*"The ones that are in paragraph form are harder, and every time I check back I have to read the entire paragraph again."* StudentA (Negative opinion)

Table 3: Narrative Interview Coded Framework

StudentA's statement shows positive feedback on the presentation (Coded as *General statements* and *Positive opinion*). He also raises difficulties interpreting the requirements when presented in paragraph format. This segment was coded as *Paragraph format* and *Negative opinion*.

The interviewer then draws out StudentA's opinion about the paragraph format by asking why he believes the instructional instrument was easier to understand. StudentA responded, *"This one was much nicer that I could basically check it off a list."* This statement shows the student using the list format to support self-evaluation, a Self-Regulated Learning (SRL) strategy that evaluates the quality and progress of work [50]. The student is using the SRL strategy to validate his work. This narrative segment is coded as *List Form*, *Problem-solving strategies*, and *Positive opinion* from the framework. The interview concluded with the interviewer asking StudentA if there was anything else he would like to say about the instructional instrument. StudentA stated, *"Nothing wrong with the assignment. It was really good for how I look at the problems."* StudentA's final statement reaffirms his positive experience with the presentation. This segment was coded as *General statements* and *Positive opinion*.

All interviews were coded using the same approach as the StudentA example segment. Table 3 presents the coding framework and example coded segments. Table 3 lists the frequencies of the segments coded to the themes. Not all the design treatment nodes were in the results because they were not used in the instructional instrument. The table shows those themes that appeared in the students' responses. The table also contains examples statements, which are coded as a positive or negative responses. The final coding framework contains 19 themes, the original 17 nodes (14 design treatment, 2 positive/negative opinion, and 1 general statement) and two emerging themes after coding the students' responses: the presentation of the information in paragraph format (*Paragraph format*), and using the assignment presentation to support problem-solving skills (*Problem-solving skills*). In the reliability analysis, Cronbach's alpha was found to be 0.73, an acceptable internal consistency.

The results show four design treatments themes from the framework: *Boldface text format* (15.79%) [13], *Code fragments* (10.53%) [32], *Goal decomposition* (15.79%) [47], and *List format* (31.58%) [47]. These results are not surprising since these design treatments were used within the instructional instrument. However, participants did not remark on one design treatment, visualising the **drawGradient** parameters in table format. Though the visualisation was not explicitly stated, participants might have referenced when identifying other treatments, such as *"structure is useful"* (StudentD) and *"laid out for what you need"* (StudentC).

The participants' preference for list format (*List format* (31.58%), All *Positive opinion*) over paragraphs (*Paragraph format* (31.58%), All *Negative opinion*) demonstrates their difficulty in identifying programming tasks with the natural language description. StudentA stated *"The ones that are in paragraph form are harder, and every time I check back I have to read the entire paragraph again"*, while StudentB stated *"dot points really, clearly gives you all the points that you just need to address"*. These statements describe the occurrence during the review process, where the list format seems better suited for finding the goals, which can help them remain focused. StudentA further elaborates that the paragraph format makes it difficult to find his immediate goals.

Most of the coded segments were positive (84.21%), such as *"very clearly stated what's needed."* This might be due to how the first question was posed, asking *"Can you tell me how you use the assignment description to better understand the problem?"* Negative responses (15.79%) were related to the *Paragraph format*. The results also showed participants using

the assignment presentation to solve the problem (*Problem-solving skills* (21.05%)). For example, StudentC stated, *"This was the one that kind of was a bit more step by step, this one. So it had it really laid out for what you need to kind of add into the program."* Participants used the assignment presentation to validate their work when solving the problem, using it as a requirements checklist. For example, StudentA stated, *"I just make sure that I've got all the requirements checked"*.

VI. DISCUSSION

The illustration study showed the assignment presentation framework is worth further exploring within CS and other disciplines. Design treatments applied in the instructional instrument help students better understand the assignment, such as the list format of subgoals helped students identify the problem's requirements and used the design treatment to validate the completion of their assignment. This presentation approach might also be useful in other subjects. Though the list presentation of subgoals was helpful to students, we would like to further explore the presentation of subgoals in a less scaffolded environment. Eventually, students will need to identify subgoals in paragraph format, but the narrative interviews showed students having difficulties with this format. As students gain more experience with programming assignments, we are interested in transitioning the presentation of subgoals to paragraph format. Further research can help define a transition scheme from highly supported list format to paragraph format.

The instructional instrument did not present all 14 design treatments, since the problem and the students' abilities did not require all the treatments. Future studies can further examine all the design treatments through multiple instructional instruments. Evaluating all the treatments might also provide guidance on its usage within a scaffolded environment.

The interview sessions were conducted on the day the assignments were due, giving students the opportunity to retrospectively reflect on how the assignment presentation helped them solve the problem. Their perspective might differ between recalling and seeing the assignment for the first time. We are interested in replicating the illustration study when students initially receive the assignment, which might generate different opinions on the design treatments.

VII. CONCLUSION

This research was motivated to help students better understand programming problems through the presentation of the problem description. A literature review was performed to identify design treatments from peer-reviewed publications that were shown to influence students' understanding. These design treatments focused on structural formatting, visualisation, contextualisation, and elaboration on the problem description. The design treatments were collated that formed a framework that scaffolded CS1 programming assignments. To evaluate the framework, we performed an illustration study that examined the application of the framework to a CS1 programming assignment. Students participating in the interviews showed list formatting was helpful to them in solving the problem, while paragraph formatting was difficult to parse. The results also showed the assignment presentation supported students' use of Self-Regulated Learning strategies, such as self-evaluation.

There are limitations to this study. For the literature review, we strive to use keywords appropriate for producing the highest results from the search criteria, but the publications related to assignment design uses a broad list of terms to describe assignments and understanding. Another limitation is the small sample size (n=4) for the narrative interviews with male participants. The small scale was also noted in a study [45] bringing in the student voice into curriculum design with four participants. However, these studies are 'crucial that these kind of experiences are studied and reported, if we want to pursue for better collaboration between students and staff' [45, p. 9]. We wanted a balanced gender representation, and future studies can employ different recruiting strategies to diversify the study group with a balanced gender representation. The interviews only evaluates students' perception on their understanding of the problem. More work is required to map students' perceptions to their success in completing the assignment.

This study contributes to the field of Computer Science Education (CSE) by integrating the student voice into the assignment design process. The illustration study showed narrative interviews can collect students' feedback on assignment design. Our study method shows students' input to the assignment design process can highlight presentation approaches perceived as difficult to comprehend. We encourage other educators to use the illustration study to examine students' thinking about the problem beyond their performance. We intend to further examine the use of narrative interviewing on future assignments to ensure our presentation approach is appropriate for the students' abilities, and expand the framework when the CSE community identifies more design treatments.

REFERENCES

[1] B. Adelson. When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 10(3):483–495, 1984.

[2] M. Ahmadzadeh, D. Elliman, and C. Higgins. An analysis of patterns of debugging among novice computer science students. *Proceedings of the ITiCSE*, pages 84–88, 2005.

[3] T. Ahoniemi and E. Lahtinen. Visualizations in preparing for programming exercise sessions. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 178:137–144, July 2007.

[4] M. Baert. Simplescreenrecorder. https://www.maartenbaert.be/simplescreenrecorder, 2019. [Online; accessed 11-Feb-2019].

[5] J. R. Baird and J. R. Northfield. *Learning from the PEEL experience*. Peel Publications, 2nd edition, 1995.

[6] L. J. Barker, C. McDowell, and K. Kalahar. Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin*, 41(2):282–286, 2009.

[7] T. Beaubouef and J. Mason. Why the high attrition rate for computer science students: Some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, June 2005.

[8] H. Becker, B. Geer, and E. Hughes. *Making the grade: The academic side of college life*. Transaction, New Jersey, 1968.

[9] S. Bergin, R. Reilly, and D. Traynor. Examining the role of self-regulated learning on introductory programming. *Proceedings of the first international workshop on Computing education research*, pages 81–86, Oct. 2005.

[10] A. Booth, D. Papaioannou, and A. Sutton. *Systematic approaches to a successful literature review*. SAGE Publications, Jan. 2012.

[11] D. Boud and N. Falchikov. Aligning assessment with long term learning. *Assessment and Evaluation in Higher Education*, 31:399–413, 2006.

[12] D. Bouvier et al. Novice programmers and the problem description effect. *ITiCSE '16 Working Group Reports*, pages 103–118, 2016.

[13] A. Carbone, J. Hurst, I. Mitchell, and D. Gunstone. Principles for designing programming exercises to minimise poor learning behaviours in students. *ACSE '00 Proceedings of the 2016 ITiCSE*, pages 26–33, 2000.

[14] A. Carbone, J. Hurst, I. Mitchell, and D. Gunstone. Characteristics of programming exercises that lead to poor learning tendencies: Part ii. *ITiCSE '01 Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 93–96, 2001.

[15] J. Creswell. *Educational research: Planning, conducting, and evaluating quantitative and qualitative research*. Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research. Pearson, 2012.

[16] L. Denton and D. McKinney. Affective factors and student achievement: A quantitative and qualitative study. *34th Annual Frontiers in Education*, pages T1G–6, 2004.

[17] T. Feldman and J. Zelenski. The quest for excellence in designing CS1/CS2 assignments. *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education (SIGCSE '96)*, pages 319–323, 1996.

[18] J. Gal-Ezer, D. Lanzberg, and D. Shahak. Interesting basic problems for CS1. *ITiCSE '04 Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 275–275, June 2004.

[19] G. Gibbs and L. Lucas. Coursework assessment, class size and student performance. *Journal of Further and Higher Education*, 21(2):183–192, 1987.

[20] D. Ginat. Efficiency of algorithms for programming beginners. *27th SIGCSE Technical Symposium on Computer Science Education*, 28(1):256–260, 1996.

[21] O. Hazzan, Y. Dubinsky, L. Eidelman, and V. Sakhnini. Qualitative research in computer science education. *ACM SIGCSE Bulletin*, 38:408–412, March 2006.

[22] G. Joughin. Assessment, learning and judgement in higher education: A critical review. In *Assessment, learning and judgement in higher education*, pages 13–27. Springer, Dordrecht, 2009.

[23] G. Joughin. The hidden curriculum: a critical review of research into the influence of summative assessment on learning. *Assessment & Evaluation in Higher Education*, 35(3):335–345, 2010.

[24] G. Kanaparan, R. Cullen, and D. Mason. Self-efficacy and engagement as predictors of student programming performance. *PACIS 2013 Proceedings*, 2013.

[25] N. Khairuddin and K. Hashim. Application of Bloom's taxonomy in software engineering assessments. *Proceedings of the 8th conference on Applied computer science*, pages 66–69, 2008.

[26] P. Kinnunen and B. Simon. My program is ok – Am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education*, pages 1–28, 2011.

[27] M. Knobelsdorf, C. Kreitz, and S. Böhne. Teaching theoretical computer science using a cognitive apprenticeship approach. *SIGCSE '14 Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 67–72, 2014.

[28] C. Köppe and L. Pruijt. Improving students' learning in software engineering education through multi-level assignments. *CSERC '14 Proceedings of the Computer Science Education Research Conference*, pages 57–62, Nov. 2014.

[29] C. L. Kussmaul. Scaffolding for multiple assignment projects in CS1 CS2. *OOPSLA*, pages 873–876, Oct. 2008.

[30] L. Layman, L. Williams, and K. Slaten. Note to self: Make assignments meaningful. *SIGCSE '07*, pages 459–463, 2007.

[31] E. Linnenbrink and P. Pintrich. The role of self-efficacy beliefs in student engagement and learning in the classroom. *Reading and Writing Quarterly*, 19:119–137, 2003.

[32] T. Lorenzen, L. Mondshein, A. Sattar, and S. Jung. A code snippet library for CS1. *ACM Inroads*, 3(1):41–45, Mar. 2012.

[33] C. Marshall and G. B. Rossman. *Designing Qualitative Research*. Sage Publications, London, 3rd edition, 1999.

[34] R. McCauley, S. Fitzgerald, G. Lewandowski, L. Murphy, B. Simon, L. Thomas, and C. Zander. Debugging: a review of the literature from an educational perspective. *Computer Science Education*, 18(2):67–92, 2008.

[35] M. McCracken et al. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 125–180, 2001.

[36] C. Miller and M. Parlett. Up to the mark: A study of the examination game. In *Guildford: Society for Research into Higher Education*. 1974.

[37] M. Powell, R. Fisher, and R. Wright. Investigative interviewing. In *Psychology and law: An empirical perspective*, pages 11–42. The Guilford Press, New York, NY, US, 2005.

[38] V. Ramalingam, D. LaBelle, and S. Wiedenbeck. Self-efficacy and mental models in learning to program. *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 171–175, 2004.

[39] A. S. Ruocco. Experiences in threading UML throughout a computer science program. *IEEE Transactions on Education*, 46(2):226–228, 2003.

[40] C. Schulte, T. Busjahn, T. Clear, J. Paterson, and A. Taherkhani. An introduction to program comprehension for computer science educators. *Proceedings of the 2010 ITiCSE Working Group*, pages 65–86, 2010.

[41] A. Settle, A. Vihavainen, and C. S. Miller. Research directions for teaching programming online. *Proceedings of the 10th International Conference on Frontiers in Education: Computer Science and Computer Engineering (CSCE)*, 2014.

[42] E. Soloway, J. Bonar, and K. Ehrlich. Cognitive strategies and looping constructs: An empirical study. *Communications of the ACM*, 26(11):853–860, Nov. 1983.

[43] D. E. Stevenson and P. J. Wagner. Developing real-world programming assignments for CS1. *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 158–162, 2006.

[44] E. Thompson, A. Luxton-Reilly, J. Whalley, M. Hu, and P. Robbins. Bloom's taxonomy for CS assessment. *Australasian Computing Education Conference*, pages 155–161, 2008.

[45] A. Tuhkala, A. Ekonoja, and R. Hämäläinen. Innovations in education and teaching international. In *Tensions of student voice in higher education: Involving students in degree programme curricula design*, pages 1–11, 2020.

[46] A. Veerasamy, D. D'Souza, and M.-J. Laakso. Identifying novice student programming misconceptions and errors from summative assessments. *Journal of Education Technology Systems*, 45(1):50–73, 2016.

[47] A. Venables, G. Tan, and R. Lister. A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the fifth international workshop on Computing education research workshop*, pages 117–128, 2009.

[48] A. Vihavainen, M. Paksula, and M. Luukkainen. Extreme apprenticeship method in teaching programming for beginners. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 93–98, 2011.

[49] J. Wolfe. Why the rhetoric of CS programming assignments matter. *Computer Science Education*, 14(2):147–163, 2004.

[50] B. Zimmerman. A social cognitive view of self-regulated academic learning. *Journal of Educational Psychology*, 81:329–339, 1989.

[51] B. Zimmerman and M. Pons. Development of a structured interview for assessing student use of self-regulated learning strategies. *American Educational Research Journal*, 23(4):614–628, 1986.