# Exploring the Complexity of Crowdsourced Programming Assignments

**Nea Pirttinen**
University of Helsinki
Helsinki, Finland
nea.pirttinen@helsinki.fi

**Juho Leinonen**
Aalto University
Espoo, Finland
juho.2.leinonen@aalto.fi

## ABSTRACT

CrowdSorcerer is a tool in which students can create their own programming assignments according to teacher's instructions, and later review their peers' assignments. In this lightning paper, we take a brief look into the level of complexity of assignments novice programmers create with the tool.

### Author Keywords

crowdsourcing, learnersourcing, assignment creation

## INTRODUCTION

Crowdsourcing has been used in computer science education mostly for student-based assignment creation. For example, PeerWise [1] allows students to create multiple-choice questions and revise course topics using questions created by their peers. In a similar way, computer science education tools like CodeWrite [3] and CrowdSorcerer [7] assist students in creating their own programming assignments according to guidelines given by the instructor.

Education tools like these allow students to approach the course topics from a different perspective than usual. If the tool also incorporates peer reviewing, which most of the tools do [1, 7] in some way, the students can also revise the course concepts through the review process. At the same time, collecting small assignments, programming-related or not, can help teachers to build a database of suitable exercises for quizzes or drill practice.

Peer review is widely used and often accurate enough in terms of coverage or quality [2, 4, 8, 6], at least for cases where the reviews will not directly affect the students' grades. However, since students are most often novices in the course topics, it is important to ensure that the quality of the collected assignments is reasonable, even after peer review. In this paper, we take a brief look into the complexity of programming assignments created by students during the second week of an introductory programming course.

Seventh SPLICE Workshop at SIGCSE 2021 "CS Education Infrastructure for All III: From Ideas to Practice", SPLICE'21, March 15-16, 2021, Virtual Event

Our research question is *What types of programming assignments do novice students create?* To answer this, a set of 91 assignments was categorized according to their types and features. The categorization identified seven different categories based on the complexity of the assignments.

## CROWDSORCERER

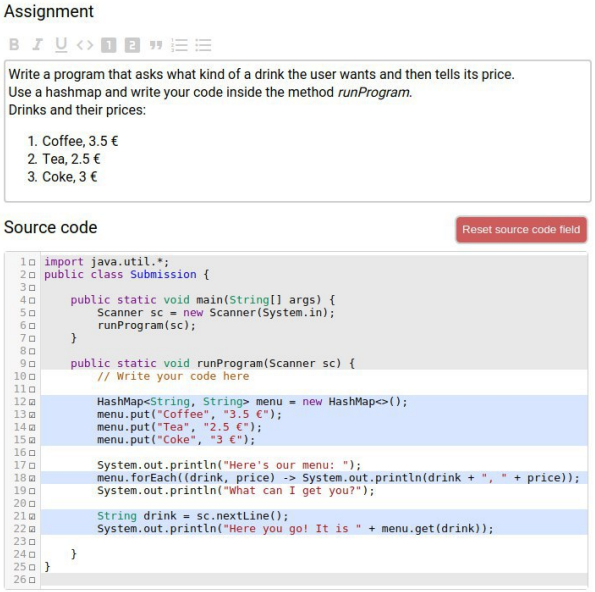


**Figure 1. The basic assignment creation view of CrowdSorcerer. The assignment field contains a student-written assignment handout. The source code field has a student-written code with the model solution marked with the checkboxes on the left (blue lines). The lines in gray form a template that cannot be edited by students.**

The data was collected using CrowdSorcerer [7], an embeddable tool used for programming assignment creation. The user interface of CrowdSorcerer is built with React[1]. The backend[2] is a Ruby on Rails application and holds most of the system's functionality and storage. The student-created assignments are sent to a Test My Code programming assignment evaluator server [9][3] to assess whether the assignments pass the student-created test cases.

The user interface for basic assignment creation can be seen in Figure 1. First, the user writes an assignment handout according to specifics given by the instructor. These instructions may

[1] https://github.com/rage/crowdsorcerer
[2] https://github.com/rage/crowdsorceress
[3] https://github.com/testmycode

require, for example, usage of conditional statements. Then, the user programs a model solution (blue lines in Figure 1) and code template for the assignment (everything else in the source code). The model solution is a full, working answer to the assignment, while the code template only contains the basic structure of the program without the crucial implementation lines. In order to help the student to recognize the relevant lines of code for the code template, and to keep the source code functional, parts of the code can be locked by the instructor so that the students are not able to edit these lines (gray lines in Figure 1). The user is also required to create test cases. The program code is automatically tested for compilation errors, and the user-given tests make sure that the program works as expected [5].

The data was collected from 91 students on an introductory Java programming course who gave their permission for using their data for scientific purposes, and completed an assignment using CrowdSorcerer. The instruction for the CrowdSorcerer assignment was: "Create an assignment that requires a student to create a program that reads an integer from the user, uses a conditional statement to inspect the integer and then prints a string. For tests, give an example input and the output the program will print with this input."

## RESULTS AND DISCUSSION

| Category | Students |
|---|---|
| Only printing | 2 |
| Single if | 2 |
| Multiple ifs | 2 |
| Simple if-else | 54 |
| Intermediate if-else | 16 |
| Advanced if-else | 8 |
| Loops (+ if-else) | 7 |

**Table 1. Categorized assignments, 91 in total.**

The student-created programming assignments were categorized according to their features. Since the instructions prompted for an assignment that asks the user for an integer input, uses a conditional statement and prints a string output, the simplest expected assignments should have at least one if-statement.

The categorized assignments are found in Table 1. The categories can be described as follows:

- **Only printing:** The assignment does not include any conditionals or other relevant structures – the program only prints hard-coded statements. The Scanner, declared and locked in the source code as a hint for the students, is not used.
- **Single if:** The assignment has a singular if-statement. The simplest acceptable assignment for the instructions given.
- **Multiple ifs:** The assignment uses multiple if-statements one after another, but not nested or if-elses.
- **Simple if-else:** If-else with one relational expression.
- **Intermediate if-else:** Slightly more complicated assignment, such as using more complicated combinations of relational expressions, or multiple if-elses. See example assignment at the end of the section.
- **Advanced if-else:** The assignment uses complicated combinations of relational expressions, and nested conditionals.
- **Loops (+ if-else):** The assignment uses some kind of loop in addition to conditional expressions.

The assignments in the first category are automatically faulty from the review point of view, as they do not follow the given instructions. All of the assignments that included loops were either intermediate- or advanced-level regarding the use of conditionals and relational expressions. In total, 60 of the 91 assignments are categorized as *simple if-else* or simpler. As the majority of the students on the course are novices, this corresponds to the level of programs they are expected to be able to write at this point. Also, there are only so many ways one can implement the assignment according to the instructions with the tools the students have learned during the first two weeks of the course. Since completing CrowdSorcerer assignments did not award any points during this iteration of the course, it is also likely that students would rather use their effort on exercises that affect their grade. Thus, students who might have been able to create more complicated programming assignments may have chosen to implement simpler programs.

The topics of the assignments range from copies of previous course assignments to novel ideas that show students' interests outside of computer science, relating to, for example, music. There were no sanctions for re-implementing previously used code, as it was deemed better that the students at least try the tool and practice input-output testing in the process.

The following assignment is an example of a typical, relatively simple program, categorized as *intermediate if-else*. The assignment has been translated from Finnish to English by the authors. Java package imports and class declarations have been omitted for the sake of brevity.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("How many cats do you have? ");

    int cats = Integer.valueOf(sc.nextLine());

    if (cats == 0) {
        System.out.println("What a pity!");
    } else if (cats >= 1 && cats <= 4) {
        System.out.println("Exemplary work!");
    } else if (cats > 4) {
        System.out.println("Wow, what a clowder!");
    }
}
```

Altogether, these preliminary results indicate that students mostly follow instructions given when creating crowdsourced programming assignments. The quality of these assignments could be evaluated, for example, through peer review, and after enough assignments have been collected, the best ones can be used to build a database of small programming assignments. These can be reused in future courses, not only in our context, but shared with the computer science education community. In our future work, we are interested in studying assignment quality in more detail, and exploring whether students from different demographics create different kinds of assignments.

## REFERENCES


[1] Paul Denny, Andrew Luxton-Reilly, and John Hamer. 2008. The PeerWise System of Student Contributed Assessment Questions. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78 (ACE '08)*. Australian Computer Society, Inc., AUS, 69–74. DOI: http://dx.doi.org/10.5555/1379249.1379255

[2] Paul Denny, Andrew Luxton-Reilly, John Hamer, and Helen Purchase. 2009. Coverage of Course Topics in a Student Generated MCQ Repository. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*. ACM, New York, NY, USA, 11–15. DOI:http://dx.doi.org/10.1145/1562877.1562888

[3] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite: Supporting Student-driven Practice of Java. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 471–476. DOI: http://dx.doi.org/10.1145/1953163.1953299

[4] John Hamer, Helen C. Purchase, Paul Denny, and Andrew Luxton-Reilly. 2009. Quality of Peer Assessment in CS1 *(ICER '09)*. Association for Computing Machinery, New York, NY, USA, 27–36. DOI:http://dx.doi.org/10.1145/1584322.1584327

[5] Vilma Kangas, Nea Pirttinen, Henrik Nygren, Juho Leinonen, and Arto Hellas. 2019. Does Creating Programming Assignments with Tests Lead to Improved Performance in Writing Unit Tests?. In *Proceedings of the ACM Conference on Global Computing Education (CompEd '19)*. ACM, New York, NY, USA, 106–112. DOI:http://dx.doi.org/10.1145/3300115.3309516

[6] Juho Leinonen, Nea Pirttinen, and Arto Hellas. 2020. Crowdsourcing Content Creation for SQL Practice *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 349–355. DOI: http://dx.doi.org/10.1145/3341525.3387385

[7] Nea Pirttinen, Vilma Kangas, Irene Nikkarinen, Henrik Nygren, Juho Leinonen, and Arto Hellas. 2018a. Crowdsourcing Programming Assignments with CrowdSorcerer. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. ACM, New York, NY, USA, 326–331. DOI: http://dx.doi.org/10.1145/3197091.3197117

[8] Nea Pirttinen, Vilma Kangas, Henrik Nygren, Juho Leinonen, and Arto Hellas. 2018b. Analysis of Students' Peer Reviews to Crowdsourced Programming Assignments. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research (Koli Calling '18)*. Association for Computing Machinery, New York, NY, USA, Article 21, 5 pages. DOI:http://dx.doi.org/10.1145/3279720.3279741

[9] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding Students' Learning Using Test My Code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 117–122. DOI:http://dx.doi.org/10.1145/2462476.2462501