Analysis of Students' Peer Reviews to Crowdsourced Programming Assignments

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

Vilma Kangas
University of Helsinki
Helsinki, Finland
vilma.l.kangas@helsinki.fi

Henrik Nygren
University of Helsinki
Helsinki, Finland
henrik.nygren@helsinki.fi

Juho Leinonen
University of Helsinki
Helsinki, Finland
juho.leinonen@helsinki.fi

Arto Hellas
University of Helsinki
Helsinki, Finland
arto.hellas@cs.helsinki.fi

ABSTRACT

We have used a tool called CrowdSorcerer that allows students to create programming assignments. The students are given a topic by a teacher, after which the students design a programming assignment: the assignment description, the code template, a model solution and a set of input-output -tests. The created assignments are peer reviewed by other students on the course. We study students' peer reviews to these student-generated assignments, focusing on examining the differences between novice and experienced programmers. We then analyze whether the exercises created by experienced programmers are rated better quality-wise than those created by novices. Additionally, we investigate the differences between novices and experienced programmers as peer reviewers: can novices review assignments as well as experienced programmers?

CCS CONCEPTS

• **Information systems** → *Crowdsourcing*; • **Social and professional topics** → *Computing education*;

KEYWORDS

peer reviews, crowdsourcing, educational data mining

ACM Reference Format:
Nea Pirttinen, Vilma Kangas, Henrik Nygren, Juho Leinonen, and Arto Hellas. 2018. Analysis of Students' Peer Reviews to Crowdsourced Programming Assignments. In *18th Koli Calling International Conference on Computing Education Research (Koli Calling '18), November 22–25, 2018, Koli, Finland*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3279720.3279741

1 INTRODUCTION

Reviewing the work of others is a common practice in science. Articles are subjected to peer review when submitted for publication, medical experiments are scrutinized by authorities before studies can proceed, and funding bodies review project proposals. While the scholarly practice of reviewing has traditionally been conducted by experts, this is not the case when reviewing is used as a pedagogical practice.

When reviewing is conducted as a pedagogical means to an end, the reviewer may have little to no knowledge on the topic. The goal of the reviewer is not only to learn more about the topic, but also to learn to be a better reviewer. There exists a few examples of the use of reviews in computing education. For example, self-explanations can be used to prompt students to explain parts of code [21] and peer assessment can be used to provide more feedback and learning opportunities [13, 18–20].

In this work, we set out to study the differences between experts and novices as peer reviewers. Our study takes place in the context of an introductory programming course with students with varying programming backgrounds. As students have different prior exposure to programming, we believe that they may also act differently as reviewers. Those with prior experience may also review assignments better, but one could also argue that those with more knowledge on the topic may judge their peers too harshly. For the purposes of this study, we have used a crowdsourcing tool that allows the generation and review of programming assignments. The assignments are generated on a topic given by a teacher, for example "for-loops", and the reviews are conducted using a predefined Likert-like scale, where the student analyzes factors such as readability of the assignment code and the suitability of the programming assignment.

We analyze if the different factors are intertwined – for example, if the assignment is suitable for a particular student, is the code in the handout also clear. We also study the differences between novice and experienced programmers on the course, with regards to both programming assignments and peer reviews. Our research questions for this study are the following:

- RQ1. How do feedback question answers correlate with each other over the whole population?
- RQ2. How do the assignments created by students with programming experience differ from those created by novice programmers?
- RQ3. How do feedback question answer correlations differ between experienced and novice programmers?

Koli Calling '18, November 22–25, 2018, Koli, Finland

ACM ISBN 978-1-4503-6535-2/18/11…$15.00
https://doi.org/10.1145/3279720.3279741

This article is organized as follows. Next, we briefly explore the concept of crowdsourcing, focusing on the context of education, and discuss the quality of crowdsourced feedback. In Section 3, we describe the study context and data, which is followed by the methodology and results in Section 4. The results and limitations of our work are discussed in Section 5, and Section 6 concludes the work and outlines future streams of research.

2 BACKGROUND

The term crowdsourcing has been used to describe a wide range of collaborative activities where the purpose of a large user base is to provide services or ideas to a party or organization of some sort, either in real life or over the Internet [5]. Crowdsourcing has been used, for example, in social networking, evaluation tasks such as reviews and voting, artifact creation and idea sharing [4].

In education, crowdsourcing has been used to create various types of content ranging from online books [16] and book supplements [7] to multiple-choice questions for rehearsal [2]. Besides generating educational material, crowdsourcing can also be used for collecting feedback or reviews [14]. Studies show that peer feedback can be accurate and constructive, even in large courses [8, 11].

In educational data mining and learning analytics, the vast majority of data-driven approaches rely on data that is generated by students. This might occur either actively or passively as they take part in a course. Such data has been used, for example, to study students' learning and behaviour [10], to identify at-risk students using predictive models [10], and to improve educational materials [12].

Hamer et al. [8] assessed peer feedback quality by comparing student-given feedback to feedback given by tutors. In a comparison of peer reviewers' total marks and tutors' total marks, the study found a notable correlation, and noted that students tend to undermark the scores of their peers. At the same time, higher performing students seem to give more appropriate scores than lower performing students.

The quality of questions collected with PeerWise [2] is good, as the majority of the collected questions and their explanations were correct and deemed relevant by instructors and peers [3]. Errors in questions or answers could be weeded out during peer review. The study is confident that students can distinguish between good and bad exercises. Besides erroneous questions, there were questions that were e.g. overly complicated, but could be improved by comments from the peer review.

The effects of anonymization on crowdsourced feedback are twofold: less dropping out of the review process, and increase in quality when looking at specific criticism, praise or suggestion as opposed to non-specific feedback concerning the target of review on a very general level [9]. This is compared to communal feedback without anonymization, where teams ask for feedback from all of the course participants, not just from their own social networks.

3 CONTEXT AND DATA

Our study was conducted on two overlapping introductory programming courses organized in Java, both held during spring 2018. One of the courses was organized for local students and had tutoring on campus, while the other was arranged as a MOOC that was free for anyone to attend. Both used the same material, which has 14 weeks of content, starting from basics of programming and ending with building larger programs with graphical user interfaces.

In addition to weekly programming assignments, the courses used a crowdsourcing tool called CrowdSourcerer [15] that allows the students to create programming assignments related to a certain topic of the course, e.g. loops. For each generated assignment, students provide (1) the assignment handout, (2) template code, (3) model solution, and (4) a set of input-output -tests.

The created assignments can then be reviewed by other students on the course. The tool was embedded into the material on eight of the fourteen weeks. The submission gathering and peer reviewing phases were organized in pairs so that every other week students created assignments and then those assignments were peer reviewed on subsequent weeks.

Completing the tasks where students were expected to create and review assignments was completely voluntary, and students were not awarded any course points for creating the assignments, nor for the peer reviews. The tool was not mentioned or advertised during the lectures or tutoring sessions, only embedded as a part of the course material.

The crowdsourcing system provides students eight feedback statements graded from one to five using a Likert-like scale. The feedback statements, translated from Finnish, are listed in Table 1. From this point onwards, we refer to the first feedback statement as FS1, second feedback statement as FS2, and so on.

#	Statement
1	Exercise is suitably difficult
2	Assignment handout is creative
3	Model solution is clearly separated from code template
4	Code is clean
5	Model solution corresponds to assignment
6	Assignment handout has been made according to instructions
7	Test inputs and outputs are reasonable
8	Assignment handout is clear

Table 1: Feedback statements used for peer reviews on the course. The statements are refererred to as FS1, FS2, etc. in the text.

For the analysis, we have combined the data from the two overlapping courses. In total, we have 1700 student-generated programming assignments and 4500 peer reviews from approximately 3000 students. Combining the data from the two courses was done to increase robustness – if we find similar characteristics from the overall population, we know that the results are not specific for only a certain group, such as online participants.

We focus on the peer review data, which consists of eight feedback statements graded from one to five using Likert-like scale, and the medians of these grades, as well as the date and time of the submission. The exercise side of the tool saves similar information from the exercise submission itself.

4 METHODOLOGY AND RESULTS

We decided to study the correlations between all of the statements, taking into account the programming background of each student.

Our hypothesis is that inspecting correlations can give us a clear view of how students see the review process, and whether novices and more experienced programmers understand review topics differently. If the scores from novices and experts correlate similarly, then it can be argued that they behave similarly as reviewers. As it would be more difficult to decipher the effects of previous programming experience as the course progresses and the novice programmers gain experience and confidence in their skills, we focus only on the first crowdsourcing assignment from the spring of 2018. Thus, we analyze student generated programming assignments that were completed during the second week of the course, and the reviews given during the third week.

Students who had not provided information of their previous programming experience at the beginning of the course or who had not provided research consent were excluded from the data, leaving us with a set of 613 students. To analyze the impact of previous programming experience, we divide the students into two groups using median split on the number of hours previously programmed. Those who had reported twenty hours or less of programming experience were regarded as novices, while others were considered to have at least some knowledge on the topic. Out of the 613, 311 were regarded as novices, and 302 as having at least some programming background. From the 311 novices, 136 had no prior programming experience.

To answer RQ1, *How do feedback question answers correlate with each other over the whole population?*, we calculated the correlations between all the peer review question answers using the Spearman correlation method. We chose the Spearman correlation method because the Likert-like scale we used is rank-based meaning the distances between the different options could differ. The correlations shown in Table 2 show that all the answers for all the feedback statement answers correlate positively with each other.

For RQ2, *How do the assignments created by those with programming experience differ from the assignments created by novice programmers?*, we determined whether the different student groups created programming assignments of different quality. We evaluated the difference by applying the Kolmogorov–Smirnov test of statistical significance to the feedback statement answers given to the exercises created by the different groups. The results, shown in row p of Table 5, are not statistically significant, meaning that there is no evidence that the two groups create different quality exercises. Additionally, the medians are the same for all of the answers, regardless of the programming experience of the students.

Lastly, for RQ3, *How do feedback question answer correlations differ between experienced and novice programmers?*, we explored whether the relationships between the different feedback statements given between the subjects of the two groups differ. We calculated the same correlations for the novices and more experienced programmers separately (Tables 3 and 4). These correlations were all statistically significant.

We had to convert the correlation results presented in Tables 3 and 4 to a result of a statistical test so that we could determine whether there was really a difference between them. The Fisher r-to-z-transformation [6] was suitable for producing this transformation. After the transformation, we then converted the z-values the transformation yielded to p-values and applied the Šidák correction for multiple comparisons [17]. The correction was necessary since we are doing multiple comparisons, and thus seemingly significant results could occur due to random chance without the correction. We were able to use this correction because all the variables were positively correlated with each other. None of the differences of the correlations were statistically significant, i.e. there is no evidence that the correlations differ between these two groups: novices and more experienced programmers.

The feedback statements that have a moderate to high correlation (≥ 0.5) over the whole population are as follows (correlations reported in parenthesis).

- Exercise is suitably difficult and assignment handout is creative (0.6)
- Model solution corresponds to assignment and assignment handout has been made according to instructions (0.66)
- Model solution corresponds to assignment and test inputs and outputs are reasonable (0.58)
- Model solution corresponds to assignment and assignment handout is clear (0.54)
- Assignment handout has been made according to instructions and test inputs and outputs are reasonable (0.62)
- Assignment handout has been made according to instructions and assignment handout is clear (0.56)
- Test inputs and outputs are reasonable and assignment handout is clear (0.52)

We also calculated the p-values for each statement pair in each case. The p-values for the correlations in Table 2 were all under $3 * 10^{-10}$, meaning that all of them were statistically significant.

	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8
FS1	1.0	0.6	0.29	0.2	0.28	0.33	0.29	0.28
FS2		1.0	0.32	0.24	0.25	0.34	0.26	0.2
FS3			1.0	0.44	0.48	0.45	0.42	0.41
FS4				1.0	0.48	0.45	0.47	0.4
FS5					1.0	0.66	0.58	0.54
FS6						1.0	0.62	0.56
FS7							1.0	0.52
FS8								1.0

Table 2: Correlations for the peer review question answers. The p-values for all of the entries are less than 10^{-5} and are therefore left out of the table. Feedback statements FS1-FS8 are defined in Table 1.

5 DISCUSSION AND LIMITATIONS

The feedback statements that correlate with each other are the same for both novice and more experienced programmers, but the correlations between almost any two statements are higher for the more experienced than for the novices. The novices have higher correlations between statements "Test inputs and outputs are reasonable" – "Model solution is clearly separated from code template", "Test inputs and outputs are reasonable" – "Code is clean", and "Test inputs and outputs are reasonable" – "Model solution corresponds to assignment". As it can be clearly seen, all of these share statement FS7, "Test inputs and outputs are reasonable". It is possible that novice programmers rate the statement higher as

	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8
FS1	1.0, 0	0.63, $<10^{-5}$	0.35, $<10^{-5}$	0.2, 0.0003	0.26, $<10^{-5}$	0.36, $<10^{-5}$	0.29, $<10^{-5}$	0.33, $<10^{-5}$
FS2		1.0, 0	0.36, $<10^{-5}$	0.26, $<10^{-5}$	0.22, $6*10^{-5}$	0.37, $<10^{-5}$	0.24, $<10^{-5}$	0.19, 0.0004
FS3			1.0, 0	0.45, $<10^{-5}$	0.49, $<10^{-5}$	0.49, $<10^{-5}$	0.39, $<10^{-5}$	0.47, $<10^{-5}$
FS4				1.0, 0	0.45, $<10^{-5}$	0.44, $<10^{-5}$	0.41, $<10^{-5}$	0.42, $<10^{-5}$
FS5					1.0, 0	0.67, $<10^{-5}$	0.48, $<10^{-5}$	0.57, $<10^{-5}$
FS6						1.0, 0	0.61, $<10^{-5}$	0.61, $<10^{-5}$
FS7							1.0, 0	0.59, $<10^{-5}$
FS8								1.0, 0

Table 3: Correlations and p-values for the peer review question answers for those who had above median prior programming experience. Feedback statements FS1-FS8 are defined in Table 1. Each cell contains two rows, the correlation in the first row and the p-value in the second.

	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8
FS1	1.0, 0	0.58, $<10^{-5}$	0.21, 0.0001	0.16, 0.003	0.24, $<10^{-5}$	0.24, $<10^{-5}$	0.25, $<10^{-5}$	0.23, $2*10^{-5}$
FS2		1.0, 0	0.22, $3*10^{-5}$	0.19, 0.0003	0.17, 0.001	0.22, $2*10^{-5}$	0.21, $5*10^{-5}$	0.14, 0.01
FS3			1.0, 0	0.38, $<10^{-5}$	0.44, $<10^{-5}$	0.4, $<10^{-5}$	0.42, $<10^{-5}$	0.32, $<10^{-5}$
FS4				1.0, 0	0.43, $<10^{-5}$	0.36, $<10^{-5}$	0.46, $<10^{-5}$	0.35, $<10^{-5}$
FS5					1.0, 0	0.66, $<10^{-5}$	0.64, $<10^{-5}$	0.52, $<10^{-5}$
FS6						1.0, 0	0.59, $<10^{-5}$	0.54, $<10^{-5}$
FS7							1.0, 0	0.43, $<10^{-5}$
FS8								1.0, 0

Table 4: Correlations and p-values for the peer review question answers for those who had median or below median prior programming experience. Feedback statements FS1-FS8 are defined in Table 1. Each cell contains two rows, the correlation in the first row and the p-value in the second.

they are not yet introduced to testing, and therefore may not know what kind of test inputs and outputs are reasonable.

Feedback statements FS1-FS4 do not correlate much with other statements, whereas statements FS5-FS8 have high correlations with each other (see Table 1 for statements). Feedback statements FS5-FS8 concern how well the assignment corresponds to the given instructions. The instructions may have been, for example, "Write an assignment that asks the programmer to use a for-loop". Those assignments may then receive better feedback if the assignment clearly asks the programmer to use a for-loop, perhaps even using those exact words. However, a clear assignment made according to the instructions does not always mean the assignment is also

	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8
all median	4	4	5	5	5	5	5	5
all sd	0.97	1.03	1.13	0.65	0.81	0.75	0.77	0.79
high median	4	4	5	5	5	5	5	5
high sd	0.98	1.09	1.05	0.59	0.70	0.69	0.64	0.75
low median	4	4	5	5	5	5	5	5
low sd	0.95	0.99	1.04	0.63	0.82	0.75	0.82	0.83
p	1	0.88	1	1	1	1	0.88	1

Table 5: Statistics from peer review question answers, p-value Kolmogorov–Smirnov test with Holm–Šidák correction of multiple comparison. In the first row, medians and standard deviations are calculated for peer review question answers for exercises made by all students. In the second row, they are calculated for the more experienced students (high), and finally, in the third row, for the novices (low). The last row contains the p-values for the Kolmogorov–Smirnov test between the novice and more experienced programmers. Feedback statements FS1-FS8 are defined in Table 1.

creative, for example, which may be the reason why the correlations between statements FS2 and FS6, or FS2 and FS8 are not high.

As stated in Section 4, the p-values of the differences of the correlations presented in Tables 3 and 4 were not statistically significant. This means that according to our results, novices can peer review programming assignments as well as more experienced programmers, which is in line with previous studies [8].

Besides the similar correlations between novices and more experienced programmers, the medians are also alike, as can be seen in Table 5. This strengthens our belief that when peer reviewing, novice and more experienced programmers behave in the same way and give very similar reviews overall.

It has been suggested by Denny et al. that students may review too difficult exercises with good grades, as they may think that complex exercises provide a good practice for the exam, while tutors reviewing the same exercises assume that complexity will only confuse students and hinder learning [3]. Our results do not support these findings, as we did not find any statistically significant differences in the correlations of novices and experienced programmers. However, there is a difference in context, as the study by Denny et al. concentrates on programming assignment answers, whereas our study looks at whole crowdsourced assignments.

Generally, assignments receive very good peer reviews. Although it would be meaningful for peers to want to help and to give good grades, previous studies suggest that peers actually tend to undermark each others' exercises [8]. Since the tool in which students create the exercises automatically checks tests for compilation errors and faulty functions when the exercise is submitted, non-functional submissions are not accepted and thus never graded in the first place. Additionally, creating assignments was not compulsory on the courses, so those students who felt they were not competent enough for the task could just skip using the system completely.

Our study comes with a number of limitations, some of which are related to the system and some to our analysis. First, the students are not required to actually try to complete the assignment that they are reviewing. This can mean that the reviewing can be conducted in a perfunctory manner. While the assignment may seem clear enough when read through quickly, there may be inaccuracies or

missing information that only becomes apparent when trying to complete the assignment. Also, it is easier to notice if the model solution functions as the assignment suggests when completing the exercise and not only reading it through.

Second, we are analyzing a set of feedback questions that are rather well-defined and given to students in a concise manner. It is possible that simply the way the feedback is inputted, i.e. a matrix, creates a situation where certain items are ranked more closely to each others.

Third, we divided the students into two groups based on their previous experience. The division was conducted using median-split, where both groups ended up with a similar number of students. While such division was meaningful as our starting point, future studies may consider more extreme ends, e.g. comparing those with no programming experience at all with those with thousands of hours of programming experience.

Finally, we do not know how the students perceived their identity as the reader of the material while answering to the peer review statements. Thus, we do not know for example if more experienced reviewers answered the statement "Exercise was suitably difficult" based on the course context, or based on their own programming knowledge, which may vastly surpass the topics at hand.

6 CONCLUSIONS AND FUTURE WORK

In this work, we analyzed the differences between Likert-like peer feedback by novice programmers and more experienced programmers on student-generated programming assignments. The core result of our study is that novices can be as good at reviewing programming assignments as more experienced participants of introductory programming courses, which supports the previous findings by Hamer et al. [8] in the context of reviewing students' answers. Moreover, when analyzing the feedback on the quality of the student-generated assignments, we observed that the assignments generated by more experienced students do not differ quality-wise from the assignments generated by the students with little to no previous experience.

As novices can create assignments that are as good as those created by more experienced programmers, it is sensible to crowdsource programming assignments from introductory programming course students. This frees up the time of the instructors on the course as they do not have to design all the assignments by themselves. In addition, a large assignment pool could be used e.g. in student modeling [1] to provide more assignments adaptively to students who need more practice. As novices are also able to review assignments as well as their more experienced peers, the peer review process can be argued to be efficient at separating good and poor assignments. Altogether, the greatest implication of this study is that using a tool such as CrowdSorcerer is sensible at least from the point of view of creating assignments. As we can trust reviews from both experts and novices, we can include the best crowdsourced assignments as future course assignments. Future work should study whether using CrowdSorcerer also has educational value to students.

In our ongoing research, we are extending the peer reviews with expert reviews and interviews to further study how and why students provide particular types of feedback. For example, do students provide good feedback if the assignment looks similar to assignments that the student has seen on the course material, or is the feedback driven by some other factors such as amusement. We are also looking into the quality of the collected assignments, as well as their use in programming courses.

REFERENCES

[1] Konstantina Chrysafiadi and Maria Virvou. 2013. Student modeling approaches: A literature review for the last decade. *Expert Systems with Applications* 40, 11 (2013), 4715–4729.

[2] Paul Denny, Andrew Luxton-Reilly, and John Hamer. 2008. Student Use of the PeerWise System. *SIGCSE Bull.* 40, 3 (June 2008), 73–77.

[3] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2009. Quality of Student Contributed Questions Using PeerWise. In *Proc. of the 11th Australasian Conference on Computing Education - Volume 95 (ACE '09)*. Australian Computer Society, Inc., Darlinghurst, Australia, 55–63.

[4] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. 2011. Crowdsourcing Systems on the World-Wide Web. *Commun. ACM* 54, 4 (April 2011), 86–96.

[5] Enrique Estellés-Arolas and Fernando González-Ladrón-De-Guevara. 2012. Towards an Integrated Crowdsourcing Definition. *J. Inf. Sci.* 38, 2 (April 2012), 189–200.

[6] Ronald Aylmer Fisher. 2006. *Statistical methods for research workers.* Genesis Publishing Pvt Ltd.

[7] Edward F Gehringer, Karishma Navalakha, and Reejesh Kadanjoth. 2011. A Student-Written Wiki Textbook Supplement for a Parallel-Architecture Course.

[8] John Hamer, Helen C. Purchase, Paul Denny, and Andrew Luxton-Reilly. 2009. Quality of Peer Assessment in CS1. In *Proc. of the 5th International Workshop on Computing Education Research Workshop (ICER '09)*. ACM, New York, NY, USA, 27–36.

[9] Julie Hui, Amos Glenn, Rachel Jue, Elizabeth Gerber, and Steven Dow. 2015. Using Anonymity and Communal Efforts to Improve Quality of Crowdsourced Feedback. In *HCOMP*.

[10] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proc. of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR '15)*. ACM, New York, NY, USA, 41–63.

[11] Chinmay Kulkarni, Koh Pang Wei, Huy Le, Daniel Chia, Kathryn Papadopoulos, Justin Cheng, Daphne Koller, and Scott R. Klemmer. 2013. Peer and Self Assessment in Massive Online Classes. *ACM Trans. Comput.-Hum. Interact.* 20, 6, Article 33 (Dec. 2013), 31 pages.

[12] Leo Leppänen, Juho Leinonen, Petri Ihantola, and Arto Hellas. 2017. Using and collecting fine-grained usage data to improve online learning materials. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. IEEE Press, 4–12.

[13] Lan Li, Xiongyi Liu, and Allen L. Steckelberg. 2009. Assessor or assessee: How student learning improves by giving and receiving peer feedback. 41 (06 2009), 525 – 536.

[14] Andrew Luxton-Reilly. 2009. A systematic review of tools that support peer assessment. *Computer Science Education* 19, 4 (2009), 209–232.

[15] Nea Pirttinen, Vilma Kangas, Irene Nikkarinen, Henrik Nygren, Juho Leinonen, and Arto Hellas. 2018. Crowdsourcing Programming Assignments with CrowdSorcerer. In *Proc. of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. ACM, New York, NY, USA, 326–331.

[16] Clifford A Shaffer, Ville Karavirta, Ari Korhonen, and Thomas L Naps. 2011. OpenDSA: beginning a community active-ebook project. In *Proc. of the 11th Koli Calling Int. Conference on Computing Education Research*. ACM, 112–117.

[17] Zbyněk Šidák. 1967. Rectangular confidence regions for the means of multivariate normal distributions. *J. Amer. Statist. Assoc.* 62, 318 (1967), 626–633.

[18] J. Sitthiworachart and M. Joy. 2003. Web-based peer assessment in learning computer programming. In *Proc. 3rd IEEE International Conference on Advanced Technologies*. 180–184.

[19] Jirarat Sitthiworachart and Mike Joy. 2004. Effective Peer Assessment for Learning Computer Programming. *SIGCSE Bull.* 36, 3 (June 2004), 122–126.

[20] Anne Venables and Raymond Summit. 2003. Enhancing scientific essay writing using peer assessment. *Innovations in Education and Teaching International* 40, 3 (2003), 281–290.

[21] Arto Vihavainen, Craig S Miller, and Amber Settle. 2015. Benefits of Self-explanation in Introductory Programming. In *Proc. of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 284–289.