Pirttinen, Nea; Denny, Paul; Hellas, Arto; Leinonen, Juho

Lessons Learned From Four Computing Education Crowdsourcing Systems

Published in:
IEEE Access

DOI:
10.1109/ACCESS.2023.3253642

Published: 01/01/2023

Document Version
Publisher's PDF, also known as Version of record

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 31 January 2023, accepted 26 February 2023, date of publication 6 March 2023, date of current version 10 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3253642

PERSPECTIVE

Lessons Learned From Four Computing Education Crowdsourcing Systems

NEA PIRTTINEN[1], PAUL DENNY[2], ARTO HELLAS[3], AND JUHO LEINONEN[2]
[1]Department of Computer Science, University of Helsinki, 00014 Helsinki, Finland
[2]School of Computer Science, The University of Auckland, Auckland 1142, New Zealand
[3]Department of Computer Science, Aalto University, 00076 Espoo, Finland

Corresponding author: Nea Pirttinen (nea.pirttinen@helsinki.fi)

The work of Nea Pirttinen was supported by the Jenny and Antti Wihuri Foundation. The work of Juho Leinonen was supported by the Ulla Tuominen Foundation.

ABSTRACT Crowdsourcing is a general term that describes the practice of many individuals working collectively to achieve a common goal or complete a task, often involving the generation of content. In an educational context, crowdsourcing of learning materials – where students create resources that can be used by other learners – offers several benefits. Students benefit from the act of producing resources as well as from using the resources. Despite benefits, instructors may be hesitant to adopt crowdsourcing for several reasons, such as concerns around the quality of content produced by students and the perceptions students may have of creating resources for their peers. While prior work has explored crowdsourcing concerns within the context of individual tools, lessons that are generalisable across multiple platforms and derived from practical use can provide considerably more robust insights. In this perspective article, we present four crowdsourcing tools that we have developed and used in computing classrooms. From our previous studies and experience, we derive lessons which shed new light on some of the concerns that are typical for instructors looking to adopt such tools. We find that across multiple contexts, students are capable of generating high quality learning content which provides good coverage of key concepts. Although students do appear hesitant to engage with new kinds of activities, various types of incentives have proven effective. Finally, although studies on learning effects have shown mixed results, no negative outcomes have been observed. In light of these lessons, we hope to see a greater uptake and use of crowdsourcing in computing education.

INDEX TERMS Contributing student pedagogy, crowdsourcing, crowdsourcing systems, learnersourcing.

I. INTRODUCTION

Crowdsourcing is the practice of having a large group of individuals contribute towards a common goal. The common goal may be grandiose such as building the largest encyclopedia in the world (i.e. Wikipedia) or something more modest, such as coming up with ideas for improving the local neighborhood. Often, reaching the common goal can be divided into smaller tasks, such as writing a single entry into Wikipedia, which then contributes towards reaching the common goal.

While there exist paid crowdsourcing services, such as Amazon MTurk, researchers and educators have had students participate in crowdsourcing efforts. One such effort, commonly used in education, is to have students create assignments into assignment pools [1], [2], which then can be shared to other students for learning and rehearsal. This act of using students as a crowd in a crowdsourcing activity is sometimes referred to as learnersourcing [3] – in this article, we include learnersourcing under the more commonly used banner of *crowdsourcing*. In computing education research, crowdsourcing has been used, for example, to create multiple choice questions [1], [4], introductory programming assignments [2], [5], and SQL exercises [6].

An instructor who considers adopting a crowdsourcing system for the first time may have several concerns. One concern is related to the quality of the crowdsourced content:

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

(1) can students really create artefacts such as assignments or course materials with high enough quality that they could serve as useful resources for other students? Instructors might also be hesitant to start using crowdsourcing systems due to (2) concerns around the time needed to set up and configure the tools. While this concern is relevant to the adoption of any new technology, students may require additional guidance to appreciate the benefits of crowdsourcing and support to use the tools effectively. Related to this are concerns around engagement – (3) if students do not apply appropriate effort to cooperate, the benefits of crowdsourcing are lost. Perhaps most importantly, instructors may question whether (4) the time students spend engaging with crowdsourcing tools is well spent, or whether different tasks would lead to superior learning outcomes. This paper strives to address these concerns using previous studies and the authors' perspectives from the use of educational tools that utilise crowdsourcing.

In this perspective article, we present four existing crowdsourcing tools we have developed and used in computing classrooms. We describe our motivations behind developing the systems and their purpose, what types of artefacts students create in the systems, and how these artefacts are evaluated. The primary contribution of this work is synthesizing lessons, supported by existing literature, that directly target some of the previously identified concerns.

This article is organised as follows. In Section II, we provide a background to research in crowdsourcing in education, as well as outline some known challenges. In Section III, we introduce four crowdsourcing systems used in the context of computing education. Section IV summarises various lessons learned, both from previous research on the crowdsourcing tools, as well as from the authors' own experiences. Section V concludes our perspective, addressing known challenges and potential solutions.

II. BACKGROUND

A. STUDENTS AS CONTRIBUTORS

While traditional models of teaching can be viewed as separating students and teachers into consumers and producers, the contributing student approach described by Collis and Moonen [7] involves students in preparing learning materials that can be shared with the class. A key part of the contributing student approach is the (technical) infrastructure, which supports the creation and dissemination of learning resources to others. Within computing education research, Hamer [8] described experiences from using the contributing student approach in a data structures course and in a formal methods course – many weaker students thrived using the approach.

This work, alongside that of an ITiCSE working group, led to the introduction of the term Contributing Student Pedagogy (CSP) to the computing education community. CSP is defined as *a pedagogy that encourages students to contribute to the learning of others and to value the contributions of others* [9]. The ITiCSE working group emphasized the dual view to student contributions; (1) students contribute to the learning of others through a variety of forms, and (2) the students value the contributions of others [9]. The working group further outlined common characteristics of CSP, including a focus on content instead of collaboration, assessing the quality of contributions, and facilitating the contributions through technology [9].

One aspect that is often given too little attention is students' contributions in peer assessment and peer-review activities, including the key step of evaluating artefacts. While peer assessment is often used to support the assessment process conducted by the teacher (see e.g. [10]), it involves students evaluating the contributions of other students [11] and, potentially, valuing these contributions where they provide actionable feedback that can improve the quality of a piece of work [12]. Such evaluation can serve multiple purposes, including providing insight on the created resources to the instructor, providing constructive feedback to the creators of the resources, and learning during the evaluation process [13].

B. CREATING, USING, LEARNING

When students engage in creating content that targets the concepts they are learning in a course, there are multiple learning effects at play. Before or during the process of creating an artefact, for example, a question or problem to be answered by their peers, students create self-explanations of the learned content for themselves [14]. This effect is particularly acute when students are expected to provide model solutions for their problems. There are differences in the quality and quantity of self-explanations between students; some are better at monitoring their comprehension failures and successes than others [14]. In general, however, the *self-explanation effect* suggests that students who explain examples to themselves learn better than those who do not [15].

Creating content, as opposed to simply reading content, also leads to improved recall [16]. This is referred to as the *generation effect*, which has been demonstrated in a wide variety of domains [17], [18], [19]. The effect was originally established within the context of memorisation tasks, where participants required to complete partial words in a list of word pairs tended to exhibit greater recall of the words compared to participants presented with an already completed list. Beyond simple memorisation of word pairs, the generation effect has been shown to generalise to more complex learning materials such as arithmetic problems [20], pictures of objects or scenes [18] and multiple-choice questions [21].

The generation effect is also related to the more broad *testing effect*, which states that being tested on previously studied material – taking tests – improves retention of the studied content [22], [23]. When compared to simply studying learning materials, taking tests during studying has been shown to improve recall especially in delayed tests, where a

final retention test is given at a later time [22]. In problem-solving, creating a solution is key; if a student simply tries to remember a solution presented to them, instead of actively solving the problem themselves, subsequent recall of the solution will be more difficult [24].

In general, having more learning content that provides opportunities for testing, be it from a crowdsourcing system or from the teacher, is beneficial for students. Providing feedback during testing can further enhance the testing effect [25], although there are some specific conditions where feedback may not be beneficial [26]. A large quantity of learning material can also support learning by facilitating *distributed practice*, i.e. distributing the learning over time and rehearsing the to-be-learned content repeatedly [27]. Distributed practice (and spaced repetition) has been shown to be more effective than massed practice, i.e. studying the source material for a longer duration in a single session [27], [28].

C. KNOWN CHALLENGES

There are a multitude of challenges related to CSP and crowdsourcing that have been voiced by researchers. In paid crowdsourcing, there have been concerns about the quality of data; researchers have posited that collected data can be inaccurate or even completely invalid [29], [30]. Although paid crowdsourcing has been used within computing education research (see e.g. [31], [32], [33], [34], [35], [36]), data quality has not received significant attention [33]. The concerns about data quality apply equally to learnersourcing contexts where students are not paid; this is a common instructor concern that we look to address in the lessons we synthesize in the current work.

Other concerns when using crowdsourcing in education include possible problems in fairness or uneven exposure to the content [8]; if, for example, content is distributed at random across all students, one could argue that many students may not gain access to the highest quality content available. Similarly, if content is created jointly, there is a need to distinguish between individual and group contributions [7], so as to minimise the risks associated with freeloading. One might also worry about the possible reduced role of the teacher, who no longer is in full control over the content that students see. Students' expectations towards their own roles in the course must also change – students could, for example, question why they should create content for their peers, as they may view this as the responsibility of the teacher [7].

Student engagement with crowdsourced activities also varies. For example, some students may contribute to creating materials, but they might not read what others have created [8]. Requiring all students to contribute content to a shared resource may lead to an increased incidence of plagiarism [7]. When publishing content that is visible to others, students who lack confidence may have concerns around being identified. Indeed, being identifiable influences how students behave and rate content produced by others, including introducing biases around gender and nationality [37].

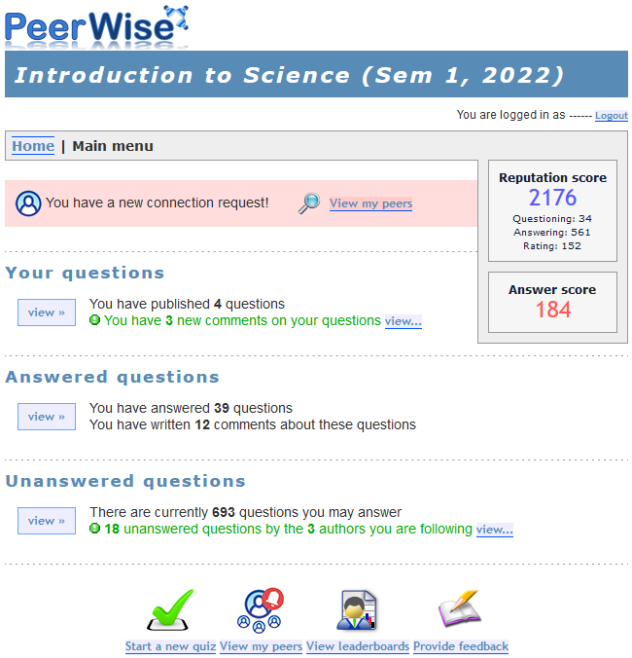

FIGURE 1. The main menu of the PeerWise system.

III. CROWDSOURCING SYSTEMS

A. PeerWise

PeerWise [1] is a web-based platform on which students create, publish and answer multiple-choice questions. Instructors and students use the same interface, although additional features are available for instructors to run reports and manage permissions.

The main artefacts produced in PeerWise are multiple-choice questions, consisting of three primary components: the question text (or stem), a set of answer options (including one correct answer), and an explanation. In addition to this, each published question has an associated comment thread to support student discussion, tags (or topics) which can be used for searching, and ratings for both quality and difficulty (which are also aggregated and used for filtering questions). As soon as a student publishes a question, it is visible for other students to answer. A student attempting a question is shown the question text and the list of answer options, and only after submitting an answer are they shown the correct option and the explanation as provided by the author. At this stage, they can also write comments and submit a rating for the question. Instructors can suggest a set of tags for students to choose from, and can assign "administrator comments" to questions which are highlighted separately from student written comments.

A common use of question repositories in PeerWise by students is for review and practice purposes leading up to summative tests and exams. Prior work has shown that answering activity in PeerWise typically increases rapidly before a test [38], and that answering questions is strongly predictive of subsequent test performance [39], [40].

TABLE 1. An overview of the differences in the creation phase of the crowdsourcing process between the four tools: PeerWise, CodeWrite, CrowdSorcerer, and SQL trainer.

	Artefact type	Instructor provides	Student creates	Artefact destination
PeerWise	MCQs	-	Question text, correct answer, explanation, tags	Other students
CodeWrite	Programming exercises	-	Question text, function header, model solution, test cases, tags	Other students
CrowdSorcerer	Programming exercises	Topic	Question text, template, model solution, test cases, tags	Instructor, other students (review only)
SQL Trainer	SQL exercises	Topic, databases	Question text, model solution	Other students

TABLE 2. An overview of the differences in the artefact review phase of the crowdsourcing process for the four tools.

	Artefact verification	Artefact review	Other students see
PeerWise	No verification	Other students after they complete assignment	Question text, MCQ options, correctness of answer and explanation
CodeWrite	Has to compile and pass created tests	Other students after they complete assignment	Question text, function header, failing tests
CrowdSorcerer	Has to compile and pass created tests	Other students as a separate task	Question text, model solution, test cases (when reviewing)
SQL Trainer	Has to compile	Other students after they complete assignment	Question text, database schema, expected output, correctness of answer

Instructors are also able to make use of the questions, for example by reviewing a question repository to identify topics that are challenging for students, or by selecting high-quality questions for use on summative tests and exams.

Questions are published immediately, without any automatic verification. When students attempt to answer questions, their answers are assessed automatically by PeerWise, through comparison with the question author's suggested answer, and the answers submitted to the question by other students. Finally, students can rate the quality and difficulty of questions as they answer them.

PeerWise has gained widespread use, attracting contributions from 900,000 students across 3,000 institutions, including more than 80% of the Association of Commonwealth Universities (ACU) institutions across the UK and Canada [41]. The tool hosts more than 6,500,000 multiple choice questions, and 240,000,000 answers, ratings, and comments.

B. CodeWrite

CodeWrite [5] is a web-based platform on which students create, publish and answer function-based programming exercises. Given the need to safely execute student-submitted code for programming exercises, CodeWrite uses the JobeServer test server[1] which supports a wide variety of programming languages.

The primary artefacts produced in CodeWrite are function-based programming exercises. Each exercise consists of four main components: a text description of the problem to be solved, a function header (with function name, inputs and output type where necessary), a set of functional test cases described as input/output pairs, and a model solution.

When a student creates a question in CodeWrite, they must provide a model solution that successfully passes all of the test cases. Tests which fail are highlighted, and the author is able to correct these components of the question and resubmit. Questions are only published, and visible to other students in the course, once a working model solution is provided. Question authors can also tag their questions with relevant topics.

Students attempting questions in CodeWrite are shown the text description of the problem and the function header, and they edit their code within the browser. Upon submitting, any failing tests are highlighted to the student and they can edit their solution and resubmit. Once a successful solution has been submitted, the student can access a comment thread for the question, as well as submit a quality and difficulty rating. In addition, all previous successful submissions made by other students are revealed. This allows students to observe the variety of ways that the problem can be solved, and to compare their own code with potentially more efficient and elegant solutions.

All submitted code – either model solutions submitted by authors, or attempts submitted by other students – is executed in a test server. If the code does not compile, or otherwise contains syntax errors, the error messages generated by the corresponding compiler or interpreter are displayed to the student. If the code runs and passes all of the test cases it is considered successful, otherwise the failing test cases are highlighted to the student (passing tests are also shown). Students can rate the quality of the questions themselves, as well as evaluate the solutions submitted by other students using a simple up/down voting mechanism.

CodeWrite has been used in two institutions, the University of Auckland and Simon Fraser University on over 80 programming courses by 21,000 students.

C. CrowdSorcerer

CrowdSorcerer [2] is a computing education tool for learning programming. The artefacts produced with CrowdSorcerer are full, working programming assignments with handouts, model solutions, and test cases, all designed by the student using the tool. Teachers typically provide some guidance to this process, such as "Create an assignment that uses for-loops".

Test cases for checking attempted solutions are designed by the assignment author. Building of the test cases is

[1] https://github.com/trampgeek/jobe

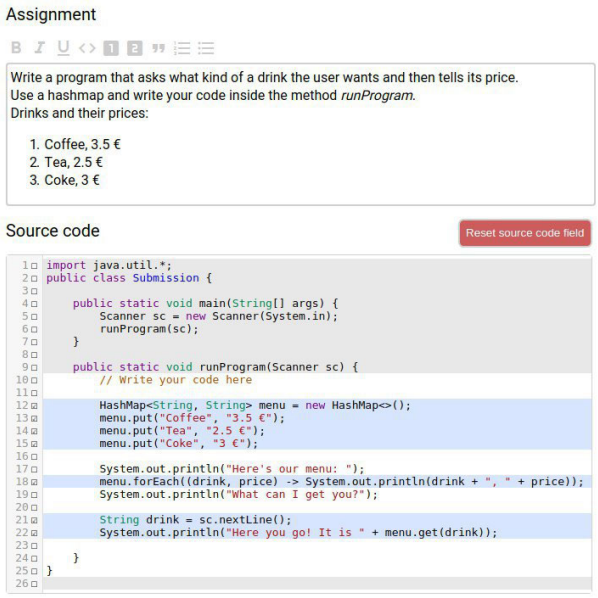

FIGURE 2. The basic assignment creation view of CrowdSorcerer.

scaffolded so that students are introduced to the process step-by-step: first, creating simple input-output pairs, and later, full test methods. Created assignments can also be tagged, using common tags suggested by the system (e.g. ''if-else'', ''for-loop'') or custom tags.

Students attempting an assignment in CrowdSorcerer only see the handout, and are provided with the model solution only after completing the assignment. The assignments are open-ended – even though the students provide a model solution to their programming assignment, it is possible that the correct end result can be achieved through different means. For teachers, CrowdSorcerer can be an easy way of adding different types of practice into the course material, and collecting a database of simple programming assignments that can be used in future course iterations.

CrowdSorcerer assesses the programming assignments automatically after they are submitted. The model solution is sent to an adapted Test My Code[2] [42] test server that acts as an automated assessment system. The test server checks for any compilation errors, and if there are none, runs the student-provided test cases for the model solution. An assignment is only marked 'finished', and published to other students, when all tests pass. All finished assignments are moved into a peer review pool and, during the peer review process, students answer an instructor-provided set of review statements on a Likert scale and give short written feedback. The reviewers also give the assignment tags they find suitable, and these tags can be cross-referenced with the assignment creator's tags when categorising the assignments.

While CrowdSorcerer is not in active use at the moment, during previous years, it has been used to create over 13,300 exercises and 17,000 peer reviews.

[2]https://github.com/testmycode

D. SQL TRAINER

SQL Trainer [6] is a semi-standalone system for practicing SQL queries. Students using the system work on SQL assignments from a range of topics (specified by the teacher); these topics can range from simple and complex SQL queries to queries that modify the database schema including creating, removing, and updating tables.

Two views are provided for artefact creation. Teachers (identified by a role received through the OAuth protocol) can define databases and topics, as well as create SQL assignments. Databases are defined by providing a publicly visible name for the database and a set of SQL statements used for creating the database. Topics, which are effectively assignment categories, are defined by providing a name, a description, and an ordering that is used when listing the topics. Both students and teachers can create assignments. When creating an assignment, a database and a topic are selected first, after which the author writes an assignment handout and a sample solution – i.e. the SQL statements needed to complete the assignment.

When students enter the system, they see a list of topics and their progress in those topics (often, for each topic, students are expected to complete a specific number of SQL assignments as well as create one or more new assignments for the specific topic). Students then choose a topic, after which they are given an assignment that they work on. The assignment is chosen randomly from the pool of assignments for that topic, although the system can also be used so that students are first given assignments created by the teacher, and only after completing these are they given assignments created by other students.

During the creation of an assignment, the entered SQL statements are evaluated against the chosen database to verify that they can be executed. Correctness of the assignment, e.g. whether the assignment handout and the provided SQL statements match, is not verified automatically. Created assignments are then added to the pool of assignments available for the particular topic.

When an assignment is successfully completed, the student can provide feedback on it by completing an instructor-created rubric. Feedback collected over time provides teachers information on the assignments, permitting data-driven insights into which assignments are likely good and which faulty. Faulty assignments can also be identified by the teacher as ones with many submissions but few completions, and such assignments can be removed.

Although SQL Trainer is not currently in use, the tool has been used to collect over 11,300 SQL exercises and, in total, 185,000 submissions to attempt to complete these exercises. Around 1,500 students have used the system.

IV. LESSONS LEARNED

A. CONTENT QUALITY

The quality of the content created by students is of primary concern when that content is intended to be used to support

the learning of other students in a course. In their seminal paper advancing the use of contributing student pedagogies in computing education, Hamer et al. make this explicit by stating that ''if students' contributions are to play an important role in their peers' learning [...] then the quality of their contribution is important'' [9]. In fact, they argue that if the student-created content is to genuinely assist the learning of other students, then the quality of the content must be assessed. Abdi et al. similarly argue for the need for quality assessment of learnersourced content, and propose that learners themselves perform such evaluations [43]. Not only is this a scalable solution compared to having domain experts make quality assessments, but it serves to encourage students to think critically and make analytical judgements about the content they are interacting with.

1) ERRONEOUS CONTENT

There are several ways in which quality assessments can be made, both for individual items and more holistically across an entire repository. Perhaps of greatest concern in the context of a learnersourced activity is when content is incorrect and therefore potentially misleading. This is a particular concern for tools like PeerWise, where there are no tool-supported checks on the content itself. On the other hand, with tools like CodeWrite and CrowdSorcerer, there is at least a verification step prior to a question being published that confirms a model solution has been provided that compiles successfully and passes the supplied tests. Additionally, with CrowdSorcerer, the created assignments are not published automatically – thus, an added quality assurance step of expert review can be utilised. Similarly, SQL Trainer tests whether the SQL statements provided by the question author can be executed, but, there is no automated way to verify that the entered SQL statements are relevant to the accompanying question handout.

Although some errors are to be expected in any sufficiently large student-created resource, serious errors such as incorrect answers appear to be relatively infrequent and readily identified by students. Purchase et al. examined the quality of a repository of questions created using PeerWise by students in a Java-based CS1 course [44]. They found that 11% of the questions did not have the correct answer provided and selected by the author. Similar numbers were found in SQL Trainer, where 9.6% of the assignments created by students were not solved by any student [6]. A similar analysis to the one by Purchase et al. was conducted by Denny et al.; using a different repository, they reported almost identical findings, with 11.5% of questions published with an incorrect answer [45]. However, Denny et al. also found that students could effectively detect such errors. They analysed all of the questions in their sample classified as being incorrect, and found that in all cases the errors were discovered by students and discussed in associated comment threads. Moreover, where a correct option was available for these questions, it was the most popular option selected by students.

2) RATINGS

One of the strengths of learnersourcing is that large repositories of learning resources can be generated quickly. Often, such resources will grow to contain more content than any individual student can use, and thus there is a need to help students find content that is both relevant to them, and high quality. Using a rating system is a common way of collecting quality scores for learnersourced content, and there is evidence that it can work well. Abdi et al. explore several approaches for aggregating scores provided by students in a learnersourcing task, and find that in general student-assigned ratings correlate strongly with those provided by domain experts [43].

When examining the four tools, it can be observed that aggregating student-assigned ratings provides useful information. In PeerWise, students provide a holistic quality rating on a 6-point scale (from 0–5) for each question, as well as a difficulty rating on a 3-point scale (from 0–2). Purchase et al. found that the aggregated difficulty ratings correlated well with a more objective measure of difficulty – the proportion of answers to a question that were correct [44]. Similarly, Denny et al. found that not only did student-assigned quality ratings correlate with instructor-assigned ratings, but students used the ratings to identify useful content and avoid questions with low quality scores [45].

Aggregated quality ratings also provide some insight into the perceived quality of a repository as a whole. For example, Figure 3 shows the distribution of average quality ratings for approximately 2.25m questions in PeerWise (each of which have at least 10 quality ratings). Although the distribution is normal, it is not centered at the mid-point halfway between 2 (''fair'') and 3 (''good''). In fact, 78% of questions have an average quality rating above this mid-point, with 5.5 times more questions rated above 3 (''good'') than below 2 (''fair''), and 7.6 times more questions rated as above 4 (''very good'') than below 1 (''poor'').

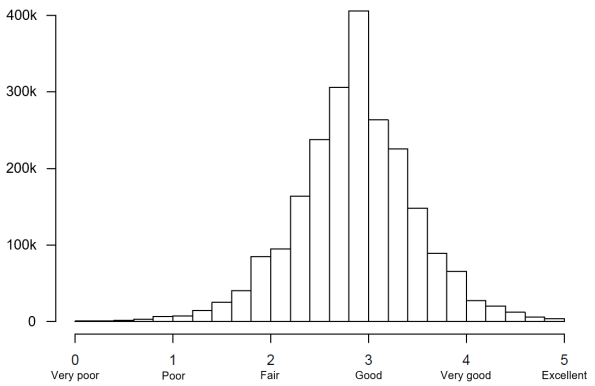

FIGURE 3. Average quality ratings of 2,253,546 questions in PeerWise with more than 10 ratings. Most questions have an average rating above the mid-point on the scale.

In CrowdSorcerer, students peer review the programming assignments using a five-point Likert scale (from a frowning face to a happy face). Based on a previous study, students

rate assignments created by experienced students and novice students similarly, and the peer review ratings do not differ significantly regardless of the previous programming experience of the reviewer [46].

For SQL Trainer, a previous study found that both assignments created by students and those created by the instructor were rated highly, although the instructor-created ones were on average rated slightly higher than those created by students [6]. The feedback questions with statistically significant differences – in favor of the teacher – were the clarity, suitability, and educational value of the questions, while no differences were observed in the easiness of the questions, or the confidence or frustration of the student towards their solution attempt.

3) CONCEPT COVERAGE

All four of the tools share the common goal of providing students with opportunities to apply the knowledge and skills they are learning. Therefore, an important criteria for assessing quality is the extent to which a repository provides these opportunities by covering all relevant concepts.

For tools like PeerWise and CodeWrite, students are typically given the freedom to select the topics and concepts that are targeted by the questions they author. Despite this, taken as a whole, student-generated repositories appear to provide at least some coverage of all topics relevant to a course. One study analysed 280 programming exercises published in CodeWrite by students in a Java-based CS1 course, classifying the questions with respect to the core language features that were used in student solutions [5]. In this case, not only were all topics targeted by the questions (with the least targeted language feature, arrays, being used in solutions to 17% of the questions) but nearly 90% of students who met the minimum participation requirements for the task (authoring one and answering 10 exercises) practised with 7 of the 8 core language features. Similar analyses involving student-authored MCQs with PeerWise have shown that the distribution of student-created questions with respect to topic correlates strongly with the topic distribution of instructor-authored questions that appear on summative assessments [47], and that good quality questions (with average student ratings above the overall repository mean) exist for all topics covered by chapters in the course textbook [44].

Where CrowdSorcerer has been used previously, guidance has been provided by the course teacher, influencing concept coverage directly. With the instructions given, students tend to create programming assignments that are on the easier and simpler side [48], at least in the beginning of an introductory course. Students rarely create completely off-topic assignments, and are mainly able to follow the instructor-given instructions for the general topic of their assignment [49]. Similarly, for SQL Trainer, the instructor can choose the topics that are presented to students under which they create their assignments, and thus can somewhat influence topic coverage. The actual distribution of topics across assignments varies however since students can choose freely which of the presented topics they prefer to target.

B. ENGAGING STUDENTS

The way that learners perceive the value of an activity is an important element for success. In this context, Expectancy-Value Theory postulates that motivation to engage in a learning task is determined by a combination of two primary factors: the *expectation* that a learner has for succeeding, and the subjective *value* of engaging in the task [50]. For many students, creating learning content for use by their peers is an unfamiliar activity, and one where they may not immediately appreciate the value. Indeed, when adopting PeerWise for the first time in an object-oriented programming course at the University of Guyana, Singh notes that students reported a preference for answering questions, a more familiar learning task, compared to authoring questions [51]. Similarly, with CrowdSorcerer, students tend to slightly favour peer reviewing over creating assignments [2], which the authors hypothesise could be due to students perceiving peer reviewing as a less laborious task compared to assignment creation; however, another possibility is that peer reviewing is more familiar to students than assignment creation.

Marks are an effective motivator for students in many courses, partly as a signal to students for how they are expected to spend their time [52]. It has been observed that even a relatively small amount of credit, usually rewarded to students for creating or answering some minimum number of items, is sufficient for encouraging most students to participate. For example, the use of 1% credit for authoring one exercise in CodeWrite resulted in 75% class participation [53], whereas Devon et al. report close to 95% participation in PeerWise across a range of database, artificial intelligence and web development courses when offering 10% course credit [54]. Similarly, without incentive, only 40% of students used CrowdSorcerer [2]. In subsequent course iterations where a small mark incentive was added, a majority of students used the system.

Motivating students to participate, and in particular to create content, is critical in small courses where resources are being learnersourced by few students. If too little content is created, the resulting resource will not be useful as a practice repository. However, in very large courses where even a small proportion of active students can generate a repository of a useful size, there may be value in allowing students to choose whether to create or simply use the generated resources. In a very recent exploration of this idea in the context of a MOOC, Singh et al. find that when content creation is optional, learners perceive more value in creating questions and create higher quality questions than when content generation is required [55].

Although marks are clearly an effective motivator for most students, it is not always possible or appropriate to assign a large number of marks to learnersourced activities for which the quality of the artefacts are typically not graded by

experts. It is possible to impact student engagement, to some extent, through user interface design that is independent of course grading schemes. One approach which has shown some promise is the use of gamification, in which game-like elements are used as rewards to incentivise behaviours that are known to be useful [56], [57], [58]. In the context of education, Landers' theory of gamified learning posits that game attributes can affect behaviours or attitudes (such as cognitive effort or time-on-task) that are known to influence learning [59]. A recent study involving PeerWise found empirical support for this theory, establishing a causal link between gamification and learning outcomes [39]. In that work, the PeerWise badge system that rewards students for answering questions for practice elicited significantly higher levels of activity, and this translated to positive effects on subsequent exam scores.

C. STUDENT LEARNING

In designing a course and the associated learning activities, instructors make many decisions about how students are expected to spend their time. Instructors may be hesitant to adopt a learnersourced activity, and encourage their students to engage with it, if they are unsure of the learning benefits. In theory, students can benefit from both creating resources, and using the resources created by others (as discussed in Subsection II-B). There is a growing body of evidence in favour of both of these activities, although there is less focus in the literature on the benefits of generating resources [55].

In courses that utilise MCQs as part of the summative assessment, there appear to be clear benefits to students in generating their own MCQs. A randomised controlled experiment using PeerWise found that students who created practice questions performed significantly better on a subsequent exam compared to students who used the questions for practice but did not create their own [38]. In this work, students were randomly assigned to either an experimental group, who authored and answered questions, or to a control group who only answered the questions created by others but did not author any. The learning effects were most pronounced when students answered exam questions on topics that were targeted by the questions they created. A follow-up study, which randomly assigned students the topics that their questions should target to control for a topic-selection bias, found similar positive effects on exam scores for the generating students [60].

Automated programming assessment tools, which provide automatic feedback to students on programming tasks, are now ubiquitous in many introductory programming courses as the benefits of this type of practice are widely accepted by both students and instructors [61]. In fact, their use is now so ingrained that educators often observe students relying excessively on the feedback generated by the autograder rather than carefully planning out their solutions [62]. However, the exercises that students tackle in such tools are typically created by the instructor or domain expert. The generation of the programming tasks can be learnersourced, and there is evidence that students benefit from this. A randomised controlled experiment using CodeWrite had students in an experimental group create the programming tasks, which included designing the test cases and a model solution [63]. Students in both the experimental and control conditions were able to use these tasks for practice within CodeWrite. To control for time-on-task, students in the experimental group were required to solve fewer tasks overall. On a subsequent exam that included a programming component, students in the experimental group performed significantly better, achieving scores more than 10% higher than the control group.

Learning effects are not always so clear. Although CrowdSorcerer mainly focuses on programming assignment creation, the system does require students to create test cases for their assignment. Three versions of test case creation exist: 1) the student provides an input-output pair, 2) the student fills in parts of a test method, and 3) the student writes the entire test method. An investigation of which of these versions most helped students in testing-related exam questions yielded no significant differences [64]. A similar result has later been observed by Singh et al. [55] who found no significant differences between students who created MCQs and those who only answered them.

D. SYSTEM DESIGN

Our primary focus in this work was to present lessons that would be of interest to instructors considering adopting crowdsourcing tasks. We have also gathered several potentially useful lessons around tool design, namely around account management and student anonymity, which may be of interest to developers.

1) AUTHENTICATION AND COURSE ORGANISATION

One of the benefits of providing tools as a free service is that it is easy for instructors to adopt them without the need to install software or set up servers. For example, in PeerWise, a new instructor can request an account by submitting a form and providing only their name, institution and email address. This automatically sets up an institutional login page for them, through which they can log in and begin creating new courses and granting their students access. Although an LTI module is available for PeerWise which permits integration with learning management systems, it is still most common for local accounts to be created and stored within the PeerWise database. One of the advantages of this organisation is that multi-institutional activities can be set up easily. Denny et al. describe a collaborative activity involving students at the University of Auckland in New Zealand and students at Simon Fraser University in Canada using CodeWrite [65]. They found that students at both institutions reported learning from the activity and were positive about having resources that they had created be shared with students at other institutions.

While CrowdSorcerer can be modified to function with each institution's authentication and servers, this does require

significantly more time and effort from both the developers and the new institution. Following the integration work presented in [66], CrowdSorcerer has been successfully used in an introductory Python course at the University of Toronto.

2) STUDENT ANONYMITY

All four of the tools we explored presented an anonymous interface to students. That is, when students are using the tools they cannot identify the authors of any of the crowdsourced learning content. This design decision was deliberate – primarily intended to promote engagement for students who may lack confidence. There is also some evidence that allowing anonymity in crowdsourced environments may help lead to more useful peer feedback and avoid biases during review phases. For example, Hui et al. [67] report that anonymous peer reviews provide more specific criticism and praise than reviews from identifiable reviewers, and students find the specificity of the reviews more helpful. Similar findings are reported by Lu and Bol [68] and Howard et al. [69] who note that anonymous reviews are more critical than non-anonymous ones. One multi-institutional study conducted using PeerWise found that when students were able to identify each other, racial and gender biases affected the way that they selected and reviewed each other's content [37].

E. POSITIONALITY

As the authors of this paper are also the developers of the tools inspected, we acknowledge the possibility of bias in our reporting. Additionally, as all of the authors are professionals in the field of computer science, this article focuses on lessons from computer science courses. Although one of the mentioned crowdsourcing tools (PeerWise [1]) has been used in other educational contexts, the impact on other fields is outside the scope of this paper.

V. CONCLUSION

Enrolments in computing courses often grow to many hundreds (or even thousands) of students. At such scale, crowdsourcing offers potentially large benefits to computing students and instructors. However, instructors may have valid concerns about incorporating student-generated learning content into their course. We conclude this paper by summarising several key lessons from those presented in Section IV, relating to common instructor concerns listed in Section I. We also mention some open challenges that are yet to be solved.

First of all, learnersourcing can work well, at least for the specific tasks presented in this paper – that is, creating multiple-choice questions, both small-scale and large-scale programming assignments, and assignments targeting SQL. One of the most pressing concerns related to crowdsourced material for educational use is its quality. We find ample evidence that the quality of learnersourced assignments – at least as supplementary materials alongside instructor-created resources – is sufficiently high to support learning in meaningful ways. Appropriate tool support can help, by providing mechanisms such as aggregated rating scores, to allow students to locate high quality, relevant and effective content. Future studies regarding peer review as an evaluation mechanism are required, though some preliminary results exist [70].

Secondly, we presented some lessons that might be of interest to developers of crowdsourcing tools, as well as computing education tool developers in general. We noted that having centralised account management is helpful for enabling cross-institutional studies whereas embedded systems can require significant integration work for this purpose. There may also be specific benefits to providing anonymity for students within crowdsourcing tools. There is emerging evidence that anonymity can result in more useful peer reviews of learnersourced content [67] and expose fewer biases related to gender and race [37].

Thirdly, we find that incentives may be necessary to engage students with crowdsourcing systems, especially since these systems are often supplementary in nature. Successful methods for engaging students include providing marks for participating in crowdsourcing activities and utilising gamification in the systems to increase engagement. However, since gamification is not a universal solution for all students [71], future research is needed to examine different ways of engaging students with crowdsourcing systems. There are suggestions that the intervals at which the materials are studied could be adjusted to optimize learning [28]; one possibility to encourage specific intervals for students would be gamification of the crowdsourcing systems that provide access to the content.

Finally, there are promising, but mixed, results related to the learning benefits of crowdsourcing activities. Students who participate in generating practice questions for exams have achieved higher marks when subsequently tested [38], [60]. However, similar results were not seen when students created test cases for crowdsourced programming assignments and were then later examined on their knowledge of testing [64]. Future work should examine more thoroughly the relationship between participating in crowdsourcing activities and learning outcomes.

Although caution regarding new approaches to teaching are understandable, given the documented benefits associated with learnersourcing, we would encourage computing instructors to explore what tool support is available and consider incorporating such tasks into their teaching.

ACKNOWLEDGMENT

The authors are grateful for the doctoral research grants awarded by Jenny and Antti Wihuri Foundation to the first author. Additionally, they are grateful for the postdoctoral research grant awarded by the Ulla Tuominen Foundation for the last author.

REFERENCES

[1] P. Denny, A. Luxton-Reilly, and J. Hamer, "The PeerWise system of student contributed assessment questions," in *Proc. 10th Conf. Australas. Comput. Educ.*, 2008, pp. 69–74.

[2] N. Pirttinen, V. Kangas, I. Nikkarinen, H. Nygren, J. Leinonen, and A. Hellas, "Crowdsourcing programming assignments with CrowdSorcerer," in *Proc. 23rd Annu. ACM Conf. Innov. Technol. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Jul. 2018, pp. 326–331.

[3] J. Kim, "Learnersourcing: Improving learning with collective learner activity," Ph.D. thesis, Massachusetts Inst. Technol., Cambridge, MA, USA, 2015.

[4] H. Khosravi, K. Kitto, and J. J. Williams, "RiPPLE: A crowdsourced adaptive platform for recommendation of learning activities," *J. Learn. Anal.*, vol. 6, no. 3, pp. 91–105, Dec. 2019.

[5] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, "CodeWrite: Supporting student-driven practice of Java," in *Proc. 42nd ACM Tech. Symp. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Mar. 2011, pp. 471–476.

[6] J. Leinonen, N. Pirttinen, and A. Hellas, "Crowdsourcing content creation for SQL practice," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, Jun. 2020, pp. 349–355.

[7] B. Collis and J. Moonen, "The contributing student: Learners as co-developers of learning resources for reuse in web environments," in *Engaged Learning With Emerging Technologies*. Berlin, Germany: Springer, 2006, pp. 49–67.

[8] J. Hamer, "Some experiences with the contributing student approach," in *Proc. 11th Annu. SIGCSE Conf. Innov. Technol. Comput. Sci. Educ.*, 2006, pp. 68–72.

[9] J. Hamer, Q. Cutts, J. Jackova, A. Luxton-Reilly, R. McCartney, H. Purchase, C. Riedesel, M. Saeli, K. Sanders, and J. Sheard, "Contributing student pedagogy," *ACM SIGCSE Bull.*, vol. 40, no. 4, pp. 194–212, Nov. 2008.

[10] F. Fagerholm and A. Vihavainen, "Peer assessment in experiential learning assessing tacit and explicit skills in agile software engineering capstone projects," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2013, pp. 1723–1729.

[11] A. Luxton-Reilly and P. Denny, "Constructive evaluation: A pedagogy of student-contributed assessment," *Comput. Sci. Educ.*, vol. 20, no. 2, pp. 145–167, Jun. 2010.

[12] T. D. Indriasari, A. Luxton-Reilly, and P. Denny, "Investigating accuracy and perceived value of feedback in peer code review using gamification," in *Proc. 26th ACM Conf. Innov. Technol. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 199–205.

[13] T. D. Indriasari, A. Luxton-Reilly, and P. Denny, "A review of peer code review in higher education," *ACM Trans. Comput. Educ.*, vol. 20, no. 3, pp. 1–25, Sep. 2020.

[14] M. T. H. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser, "Self-explanations: How students study and use examples in learning to solve problems," *Cognit. Sci.*, vol. 13, no. 2, pp. 145–182, Apr. 1989.

[15] K. VanLehn, R. M. Jones, and M. T. H. Chi, "A model of the self-explanation effect," *J. Learn. Sci.*, vol. 2, no. 1, pp. 1–59, Jan. 1992.

[16] R. J. Crutcher and A. F. Healy, "Cognitive operations and the generation effect.," *J. Experim. Psychol., Learn., Memory, Cognition*, vol. 15, no. 4, pp. 669–675, Jul. 1989.

[17] P. A. DeWinstanley and E. L. Bjork, "Processing strategies and the generation effect: Implications for making a better reader," *Memory Cognition*, vol. 32, no. 6, pp. 945–955, Sep. 2004.

[18] H. Kinjo and J. G. Snodgrass, "Does the generation effect occur for pictures?" *Amer. J. Psychol.*, vol. 113, no. 1, p. 95, 2000.

[19] D. L. Scapin, "Generation effect, structuring and computer commands," *Behaviour Inf. Technol.*, vol. 1, no. 4, pp. 401–410, Oct. 1982.

[20] B. Rittle-Johnson and A. O. Kmicikewycz, "When generating answers benefits arithmetic skill: The importance of prior knowledge," *J. Experim. Child Psychol.*, vol. 101, no. 1, pp. 75–81, Sep. 2008.

[21] M. R. Kelley, E. K. Chapman-Orr, S. Calkins, and R. J. Lemke, "Generation and retrieval practice effects in the classroom using PeerWise," *Teaching Psychol.*, vol. 46, no. 2, pp. 121–126, Apr. 2019.

[22] H. L. Roediger and J. D. Karpicke, "Test-enhanced learning: Taking memory tests improves long-term retention," *Psychol. Sci.*, vol. 17, no. 3, pp. 249–255, Mar. 2006.

[23] M. Carrier and H. Pashler, "The influence of retrieval on retention," *Memory Cognition*, vol. 20, no. 6, pp. 633–642, Nov. 1992.

[24] L. L. Jacoby, "On interpreting the effects of repetition: Solving a problem versus remembering a solution," *J. Verbal Learn. Verbal Behav.*, vol. 17, no. 6, pp. 649–667, Dec. 1978.

[25] R. E. Eisenkraemer, A. Jaeger, and L. M. Stein, "A systematic review of the testing effect in learning," *Paidéia (Ribeirão Preto)*, vol. 23, no. 56, pp. 397–406, Sep. 2013.

[26] B. Pastötter and K.-H.-T. Bäuml, "Reversing the testing effect by feedback: Behavioral and electrophysiological evidence," *Cognit., Affect., Behav. Neurosci.*, vol. 16, no. 3, pp. 473–488, Jun. 2016.

[27] H. Ebbinghaus, "Memory: A contribution to experimental psychology," *Ann. Neurosci.*, vol. 20, no. 4, p. 155, 2013.

[28] P. Smolen, Y. Zhang, and J. H. Byrne, "The right time to learn: Mechanisms and optimization of spaced learning," *Nature Rev. Neurosci.*, vol. 17, no. 2, pp. 77–88, Feb. 2016.

[29] M. Chmielewski and S. C. Kucker, "An MTurk crisis? Shifts in data quality and the impact on study results," *Social Psychol. Personality Sci.*, vol. 11, no. 4, pp. 464–473, May 2020.

[30] R. Kennedy, S. Clifford, T. Burleigh, R. Jewell, and P. Waggoner, "The shape of and solutions to the MTurk quality crisis," *SSRN Electron. J.*, vol. 8, no. 4, pp. 614–629, 2018.

[31] M. Mulvey, "Effects of visualization on algorithm comprehension," M.S. thesis, Dept. Comput. Sci., Univ. Wisconsin-Milwaukee, Milwaukee, WI, USA, 2015.

[32] M. J. Lee and A. J. Ko, "Personifying programming tool feedback improves novice programmers learning," in *Proc. 7th Int. Workshop Comput. Educ. Res.*, Aug. 2011, pp. 109–116.

[33] A. Hellas, A. Zavgorodniaia, and J. Sorva, "Crowdsourcing in computing education research: Case Amazon MTurk," in *Proc. 20th Koli Calling Int. Conf. Comput. Educ. Res.*, Nov. 2020, pp. 1–5.

[34] M. J. Lee and A. J. Ko, "Investigating the role of purposeful goals on novices engagement in a programming game," in *Proc. IEEE Symp. Vis. Lang. Human-Centric Comput. (VL/HCC)*, Sep. 2012, pp. 163–166.

[35] P. T. Wilson, J. Pombrio, and S. Krishnamurthi, "Can we crowdsource language design?" in *Proc. ACM SIGPLAN Int. Symp. New Ideas, New Paradigms, Reflections Program. Softw.*, Oct. 2017, pp. 1–17.

[36] M. J. Lee and A. J. Ko, "Comparing the effectiveness of online learning approaches on CS1 learning outcomes," in *Proc. 11th Annu. Int. Conf. Int. Comput. Educ. Res.*, Aug. 2015, pp. 237–246.

[37] G. Morales-Martinez, P. Latreille, and P. Denny, "Nationality and gender biases in multicultural online learning environments: The effects of anonymity," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, Apr. 2020, pp. 1–14.

[38] P. Denny, "Generating practice questions as a preparation strategy for introductory programming exams," in *Proc. 46th ACM Tech. Symp. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Feb. 2015, pp. 278–283.

[39] P. Denny, F. McDonald, R. Empson, P. Kelly, and A. Petersen, "Empirical support for a causal relationship between gamification and learning outcomes," in *Proc. CHI Conf. Hum. Factors Comput. Syst.* New York, NY, USA: Association for Computing Machinery, Apr. 2018, pp. 1–13.

[40] S. Snow, A. Wilde, P. Denny, and M. C. Schraefel, "A discursive question: Supporting student-authored multiple choice questions through peer-learning software in non-STEMM disciplines," *Brit. J. Educ. Technol.*, vol. 50, no. 4, pp. 1815–1830, Jul. 2019.

[41] P. Denny, "Four million questions and a few answers: Lessons from research on student-generated resources," in *Proc. ACM Conf. Global Comput. Educ.* New York, NY, USA: Association for Computing Machinery, May 2019, p. 1.

[42] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel, "Scaffolding students learning using test my code," in *Proc. 18th ACM Conf. Innov. Technol. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Jul. 2013, pp. 117–122.

[43] S. Abdi, H. Khosravi, S. Sadiq, and G. Demartini, "Evaluating the quality of learning resources: A learnersourcing approach," *IEEE Trans. Learn. Technol.*, vol. 14, no. 1, pp. 81–92, Feb. 2021.

[44] H. Purchase, J. Hamer, P. Denny, and A. Luxton-Reilly, "The quality of a PeerWise MCQ repository," in *Proc. 12th Australas. Conf. Comput. Educ.*, vol. 103, 2010, pp. 137–146.

[45] P. Denny, A. Luxton-Reilly, and B. Simon, "Quality of student contributed questions using PeerWise," in *Proc. 11th Australas. Conf. Comput. Educ.*, vol. 95, 2009, pp. 55–63.

[46] N. Pirttinen, V. Kangas, H. Nygren, J. Leinonen, and A. Hellas, "Analysis of students peer reviews to crowdsourced programming assignments," in *Proc. 18th Koli Calling Int. Conf. Comput. Educ. Res.*, Nov. 2018, pp. 1–5.

[47] P. Denny, A. Luxton-Reilly, J. Hamer, and H. Purchase, "Coverage of course topics in a student generated MCQ repository," *ACM SIGCSE Bull.*, vol. 41, no. 3, pp. 11–15, Jul. 2009.

[48] N. Pirttinen and J. Leinonen, "Exploring the complexity of crowdsourced programming assignments," in *Proc. 7th SPLICE Workshop (SIGCSE)*, 2021, pp. 1–4.
[49] N. Pirttinen, "On the quality of crowdsourced programming assignments," M.S. thesis, Dept. Comput. Sci., Univ. Helsinki, Helsinki, Finland, 2020.
[50] A. Wigfield, "Expectancy-value theory of achievement motivation: A developmental perspective," *Educ. Psychol. Rev.*, vol. 6, no. 1, pp. 49–78, Mar. 1994.
[51] L. Singh, "Technology enhanced peer learning with PeerWise: Experiences and perceptions from a developing country," *Caribbean Teach. Scholar*, vol. 4, no. 1, pp. 5–22, 2014.
[52] J. Biggs and C. Tang, *Teaching for Quality Learning at University*. Berkshire, U.K.: Open Univ. Press, 2011.
[53] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, "Understanding the syntax barrier for novices," in *Proc. 16th Annu. Joint Conf. Innov. Technol. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Jun. 2011, pp. 208–212.
[54] J. Devon, J. H. Paterson, D. C. Moffat, and J. McCrae, "Evaluation of student engagement with peer feedback based on student-generated MCQs," *Innov. Teaching Learn. Inf. Comput. Sci.*, vol. 11, no. 1, pp. 27–37, Jun. 2012.
[55] A. Singh, C. Brooks, Y. Lin, and W. Li, "What's in it for the learners? Evidence from a randomized field experiment on learnersourcing questions in a MOOC," in *Proc. 8th ACM Conf. Learn. Scale*. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 221–233.
[56] S. D. S. Borges, V. H. S. Durelli, H. M. Reis, and S. Isotani, "A systematic mapping on gamification applied to education," in *Proc. 29th Annu. ACM Symp. Appl. Comput.* New York, NY, USA: Association for Computing Machinery, Mar. 2014, pp. 216–222.
[57] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work? A literature review of empirical studies on gamification," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, Washington, DC, USA, Jan. 2014, pp. 3025–3034.
[58] T. D. Indriasari, A. Luxton-Reilly, and P. Denny, "Gamification of student peer review in education: A systematic literature review," *Educ. Inf. Technol.*, vol. 25, no. 6, pp. 5205–5234, Nov. 2020.
[59] R. N. Landers, "Developing a theory of gamified learning: Linking serious games and gamification of learning," *Simul. Gaming*, vol. 45, no. 6, pp. 752–768, Dec. 2014.
[60] P. Denny, E. Tempero, D. Garbett, and A. Petersen, "Examining a student-generated question activity using random topic assignment," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 146–151.
[61] H.-M. Chen, B.-A. Nguyen, Y.-X. Yan, and C.-R. Dow, "Analysis of learning behavior in an automated programming assessment environment: A code quality perspective," *IEEE Access*, vol. 8, pp. 167341–167354, 2020.
[62] E. Baniassad, L. Zamprogno, B. Hall, and R. Holmes, "STOP THE (AUTOGRADER) INSANITY: Regression penalties to deter autograder overreliance," in *Proc. 52nd ACM Tech. Symp. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Mar. 2021, pp. 1062–1068.
[63] P. Denny, D. Cukierman, and J. Bhaskar, "Measuring the effect of inventing practice exercises on learning in an introductory programming course," in *Proc. 15th Koli Calling Conf. Comput. Educ. Res.* New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 13–22.
[64] V. Kangas, N. Pirttinen, H. Nygren, J. Leinonen, and A. Hellas, "Does creating programming assignments with tests lead to improved performance in writing unit tests?" in *Proc. ACM Conf. Global Comput. Educ.* New York, NY, USA: Association for Computing Machinery, May 2019, pp. 106–112.
[65] P. Denny, D. Cukierman, A. Luxton-Reilly, and E. Tempero, "A case study of multi-institutional contributing-student pedagogy," *Comput. Sci. Educ.*, vol. 22, no. 4, pp. 389–411, Dec. 2012.
[66] N. Pirttinen and J. Leinonen, "Integrating CrowdSorcerer: Lessons learned," in *Proc. Workshop Comput. Sci. Educ. Infrastruct.*, 2019, pp. 24–25.
[67] J. Hui, A. Glenn, R. Jue, E. Gerber, and S. Dow, "Using anonymity and communal efforts to improve quality of crowdsourced feedback," in *Proc. AAAI Conf. Human Comput. Crowdsourcing*, vol. 3, Sep. 2015, pp. 72–82.
[68] R. Lu and L. Bol, "A comparison of anonymous versus identifiable e-peer review on college student writing performance and the extent of critical feedback," *J. Interact. Online Learn.*, vol. 6, pp. 100–115, Jun. 2007.
[69] C. D. Howard, A. F. Barrett, and T. W. Frick, "Anonymity to promote peer feedback: Pre-service teachers comments in asynchronous computer-mediated communication," *J. Educ. Comput. Res.*, vol. 43, no. 1, pp. 89–112, Jul. 2010.
[70] N. Pirttinen and J. Leinonen, "Can students review their peers? Comparison of peer and instructor reviews," in *Proc. 27th ACM Conf. Innov. Technol. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Jul. 2022, pp. 12–18.
[71] L. Hakulinen, T. Auvinen, and A. Korhonen, "Empirical study on the effect of achievement badges in TRAKLA2 online learning environment," in *Proc. Learn. Teaching Comput. Eng.*, Mar. 2013, pp. 47–54.

NEA PIRTTINEN is currently a Doctoral Researcher with the University of Helsinki, Finland. Her research interests include the use of crowdsourcing in computer science education and analysis of the quality of student-created assignments.

PAUL DENNY is currently an Associate Professor of computer science with The University of Auckland, New Zealand. His research interests include developing and evaluating tools for supporting collaborative learning, particularly involving student-generated resources and exploring the ways that students engage with these environments.

ARTO HELLAS is currently a Senior University Lecturer with Aalto University, Finland. His current research interest includes understanding and improving learning in digital learning environments.

JUHO LEINONEN is currently a Postdoctoral Researcher with The University of Auckland. His research focuses on how to best support and engage diverse learner populations with educational technology and artificial intelligence. In particular, by developing deeper insights into students' learning through fine-grained learning analytics, by utilizing educational technology, artificial intelligence, and large language models for supporting students and teachers alike, and by leveraging learner sourcing to create ample learning opportunities tailored to distinct student needs.