E-assessment in programming courses: Towards a digital ecosystem supporting diverse needs?

Aparna Chirumamilla[1][0000-0001-6985-8476] and Guttorm Sindre[2][0000-0001-5739-8265]

Department of Computer Science, Norwegian University of Science and Technology, Norway, {aparnav, guttors}@ntnu.no

Abstract. While a number of advantages have been discussed on e-learning/e-assessment tools, little research has been reported on programming courses. Today, the different types of questions have been used in exams based on course type, e.g., Text-based questions, mathematical questions, and programming questions. All these question types require supporting plug-ins for e-assessments. In this study, we provide our practical experience on programming exams in Inspera Assessment and Blackboard Learn, especially focusing on Parsons problems (drag-and-drop questions) and code writing questions. Our findings indicate that currently, tools have basic support for programming exams, and also there is a low-level integration between the tools. However, the adaptability of any exam system could depend on the interoperability between the platforms and external plugins. Hence, more improvements can be made with the implementation of e-assessments in digital ecosystems while it requires a lot of changes internally and outside institutions. In the paper, we will explain how a digital ecosystem within e-assessment could improve assessments and how it supports diverse needs of programming exams.

Keywords: Digital ecosystem, e-Assessments. Programming exams, Parson problems, Code writing,

1 Introduction

Many universities are transitioning from pen and paper exams to e-exams [1]. At the same time, formative e-assessment is receiving increased attention [2]. With automated self-tests where students can get immediate feedback, it is possible to have rapid feedback cycles scale to large and distributed classes without overloading the teaching staff. However, e-assessment systems need to be well adapted to user needs, supporting appropriate assessment tasks for the intended learning outcomes. The development of good test items is often time-consuming, so universities could save effort and increase quality if tests could be shared across countries and learning institutions [3]. Also, it would be interesting to share data and metadata, e.g., about the performance of various student groups, for benchmarking and adaptive testing.

A digital ecosystem is a business ecosystem based on an organizational network in the context of digital technology [4-6]. Digital ecosystems are formed based on digital objects (digital content, products, ideas, software, hardware, infrastructure) that are interchanged and shared between independent actors [7]. The potential advantages of digital ecosystems in e-learning were outlined more than a decade ago [8, 9]. An e-learning ecosystem is the learning community, together with the enterprise, united by a learning management system (LMS) and it is formed by three categories of components: content providers, consultants, and infrastructure [8]. For the e-assessment aspects of such an ecosystem, sharing of content (e.g., tests and test items) and metadata (e.g., anonymized student scores on test items, to assess difficulty) would be a key ingredient. In addition, easy development and good availability of plug-ins to support various needs in e-assessment would be essential. Traditional monolithic systems might have the ambition that customers find all the features they require within the system. However, user needs will be quite diverse, related to different disciplines and learning outcomes, pedagogical approaches, assessment types, different devices to be used, students with special needs, languages and cultures, and different national rules and regulations of assessments, grading and collection of personal information. In addition, the system should be able to evolve quickly to cater for new needs [10], e.g., new learning methods, test types, technology.

Although monolithic systems may include many features, these will tend to be features that a sufficient number of mainstream customers require, while more specialized needs will not be supported. Moreover, they tend to become heavy and slow to respond to changes. If an e-learning system has an open, well-documented API, this could allow for plug-ins from other vendors, or from universities themselves, with niche expertise to quickly develop functionality supporting specific needs. Our research questions for this paper are: ***RQ1:*** *To what extent does e-learning/e-assessment tools support e-assessment tasks specifically needed in programming courses?* ***RQ2:*** *In what ways could a digital ecosystem within e-assessment make for improved assessments?*

In the case study performed we look in most detail at the tools used in the authors' own university, which we had the opportunity to try out in detail, whereas other related tools were only studied via documentation available on the internet. The rest of the paper is structured as follows: Section 2 provides some background on question types in programming and identifies two question types for which the support (or lack of support) will be specifically investigated in the case study – namely Parsons problems [11] and code writing questions [12]. Section 3 then looks at the support for these question types in typical e-assessment/e-learning tools, with most detailed focus on the tools used in the authors' university, namely Blackboard Learn and Inspera Assessment. Section 4 then discusses whether the progress towards digital ecosystems with open API's could help improve the support for more diverse needs in e-assessment. Finally, section 5 concludes the paper.

2 Question types for e-assessment in programming

Programming exams may contain many different types of questions [13]. The below list provides some broad categories:

- Conceptual questions: These are questions that do not directly involve code, but focus on the recall and understanding of concepts, e.g., “What is a key difference between a list and a set?” (possibly a multiple choice question) or “Explain the concept of polymorphism and its utility?” (possibly a free text question)
- Code tracing: The code is given, and the candidate’s task is to explain what the code does. Within this category, questions may vary from those requiring only brief answers, e.g., “What will be the output of this program?”, to more detailed ones, e.g., “Explain what this program does, line by line.”
- Code writing: It is explained what a program is supposed to do, and the candidate’s task is to write the code.
- Code completion: It is explained what a program is supposed to do, and some code is provided, but not fully complete. The candidate’s task is then to fill in or select missing parts, or to rearrange code lines in the correct order.
- Error detection: It is explained what a program is supposed to do, and some faulty code is provided. The candidate’s task is then to identify the mistakes, possibly also to propose corrections.

As indicated by Sheard et al. [14], code writing appears to be the most used question type in programming exams, followed by code tracing. Writing and tracing tasks can be seen as opposites, i.e., write all the code vs. write no code (rather understand the code which is given). Completion and error detection tasks as somewhere in between those two extremes, requiring both understanding of the code already given, and ability to write some extra code: the missing parts to be added to completion tasks, the corrections to be proposed for error detection tasks.

A detailed analysis of all possible question types would be prohibitively time-consuming, so here we choose to focus on two specific question types, namely Parsons problems [11] and code writing questions [12]. The reason for choosing these two types is that they are quite specific for the discipline of programming, whereas other question types could more easily be supported by generic question types found in most e-assessment and e-learning systems. For instance, conceptual questions could be implemented as free-text short answer tasks or multiple choice questions. The same applies to code tracing questions, where the brief answer variety might typically be given as multiple choice, fill-in-number or fill-in-text depending on the output, while the longer variety could be a free-text answer or a sequence of fill-in fields showing the changes of variable content during execution. Code completion tasks (other than Parsons problems) could be implemented by e.g. multiple choice, fill-in, or pull-down menus for each missing code fragment, and error detection could again be short answer, fill-in (for proposed corrections) or multiple choice (selecting between real errors and distractors).

What are then the particularities of the two mentioned question types? *Parsons problems* [11] are coding problems where it is explained what some piece of code is supposed to do, and the code lines are given, but in jumbled order. It is then the candidates’ task to rearrange them in the right order. This question type has attracted a lot of research interest [15-18] because it reduces cognitive load for the students (e.g., recall of syntax, avoiding typing mistakes), yet still tests their visual-spatial abilities, constructive skills in solving a problem and constructing a solution from available building blocks. Since building blocks are larger (entire code lines rather than character by character on the keyboard), each question can be solved faster, thus potentially achieving better topical coverage in the exam set as a whole. Also, quick solution and automated feedback make such problems interesting for digital learning resources with self-testing features, for instance, the interactive e-book [19] makes extensive use of such problems among its exercises. Questions in Parsons problems can be made easier by providing hints [20] or more difficult by adding distractors [18], they can be one-dimensional (most common) or two-dimensional [21], the latter relevant with programming languages where indents have semantic significance (e.g., Python).

A common way of implementing Parsons problems digitally would be as drag-and-drop questions – a featured question type in many e-learning / e-exam applications. Drag and drop questions may test students' higher order thinking skills, i.e., algorithmic problem-solving skills [22, 23]. The recent research has been progressed more towards the visual programming language (VPL) that allows users to create programs using drag-and-drop genre [24]. However, its use in e-exam applications will normally not have been made with programming tasks in mind, rather tasks such as placing names in the correct positions on a background picture (e.g., Latin names of body parts for an anatomy exam, names on countries on a map for a primary school Geography exam). Hence, standard tool support for drag-and-drop questions may not be ideal for Parsons problems in programming.

Code writing tends to be a key element of programming exams, and most would agree that doing these tasks with pen and paper is not particularly authentic. Switching to a digital interface will make the task more similar to real work – but not necessarily fully authentic, as there may be various ambition levels to the tool support. For instance, students may be able to type the code in the test interface, but this could be in an editor with specific support for code writing (more authentic) or in a generic text input window with few functional features (less authentic). Also, students might be able to compile and run the code (more authentic), or not (less authentic). Sometimes, the more authentic, the better – but not always. A problem with the ability to compile, run, and test the code during an exam, for instance, is that students will then spend more time on each programming task – due to the need to debug and rerun if something was not working. More time on each task would give poorer coverage of the learning outcomes, especially if tool usage was not among the specified learning outcomes for the course. An ideal e-exam tool should therefore have a wide range of support for code writing tasks, anything from writing in a fairly simple editor without the ability to run, to professional tool support for code editing, testing and debugging.

3 Analysis of mainstream tool support

As shown in [25], there are many tools for e-assessment of programming, but many of these are standalone applications or cloud tools not integrated with official university information systems. This section looks at mainstream tool support for Parsons problems and code writing problems, with special focus on Blackboard Learn and Inspera Assessment, which happen to be the mandatory tools in the authors' university for formative and summative e-assessment, respectively. The first subsection looks at Blackboard Learn, the second at Inspera Assessment, and the third makes a quick review of some other tools.

3.1 Blackboard Learn

Blackboard Learn is the current LMS for the authors' university. It is used for communication between teaching staff and students during the semester, e.g., course info and announcements, learning resources, exercises (if not graded), etc. It is not compulsory to use it for everything, so teaching staff could use supplementary tools, in addition, for instance, for students' automated self-testing. However, it would be convenient both for teachers and students if course tasks are seamlessly supported through Blackboard, so that they avoid confusing and time-consuming switches between tools [26].

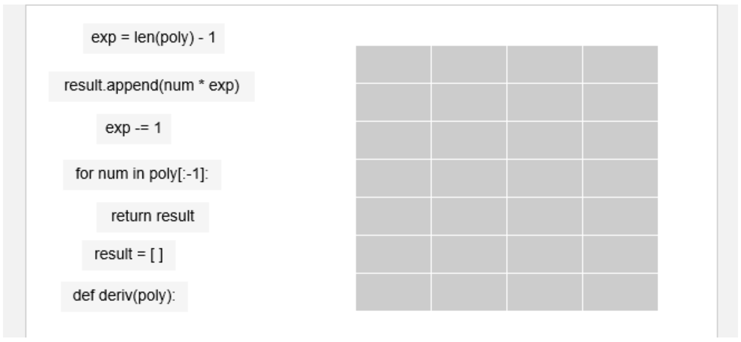
Support for Parsons problems in Blackboard turns out to be limited. Drag and drop questions do not exist, so such questions would instead have to be approximated by other question types. Obvious candidates might be *ordering questions* or *jumbled sentence questions.* Ordering questions would show the code lines in a shuffled order, then let the user assign ordinal numbers to each in input fields beside the code lines. This is not entirely ideal for the purpose. For instance, code lines are not repositioned, so the resulting code is not easily read. Reordering requires changing the ordinal numbers of all code lines affected, whereas a modern drag and drop interface might solve this by repositioning fewer lines. Jumbled sentence questions would give a series of input fields, where each would yield a drop-down menu when clicked, with all the code lines as alternatives. The student would then have to make a multiple choice selection for each input field. This would appear somewhat better than the ordering question since at least the code would be shown in the wanted order when selections had been made. However, reordering would have the same issues as with the ordering questions, and if the task contains many code lines, the drop-down menus will be long and clumsy.

Specific support for Code writing problems in Blackboard does not exist, beyond generic essay and short answer question forms meant for natural language text, or using file upload questions (e.g., student could write the code in a separate tool more fit for programming, and then upload the file to Blackboard).

3.2 Inspera Assessment

When it comes to Parsons problems, Inspera Assessment does support drag and drop questions. The resulting interface for the student while solving the task is therefore more elegant than what can be achieved in Blackboard, though there are some issues with the user interface. The task has to be made with separate drop areas for each code line, rather than one big drop area where the order is given by relative positioning. This means that the student still has to reposition several code lines in cases where a better interface might have gotten away with just repositioning one line and having other lines yield place. Especially, if trying to make two-dimensional Parsons problems, the snapping feature may behave a little counter-intuitively, since it is not determined by the position of the mouse pointer, rather the middle of the drag object (mouse pointer would be more natural, or the left edge of the drag object). Parsons problems become very time-consuming for the teacher to develop in Inspera, since all the drag areas must be created manually one by one and filled with solution (and possibly distractor) code lines, and then linked to the correct drop areas, also manually created one by one. Especially for two-dimensional Parsons problems, this takes quite a lot of time. An illustration of a two-dimensional Parsons problem for Python, as implemented in Inspera, is shown in Figure 1. For space reasons, the natural language explanation of what the code was supposed to do is omitted, showing only the interactive part of the screen. The candidate's task would be to drag each code line into the correct position in the grid (the function heading `def deriv(poly):` going upper left), both concerning vertical order and horizontal indenting, as indents have semantic significance in Python. In Inspera Assessment, the 28 drop areas must be created one by one, hand positioned in the grid and adjusted for size, hence quite time-consuming for the question author.

.



```
exp = len(poly) - 1
result.append(num * exp)
exp -= 1
for num in poly[:-1]:
return result
result = [ ]
def deriv(poly):
```

Fig. 1. Two-dimensional Parson problem for Python.

For code writing tasks, Inspera has a dedicated question type called "Programming". Notably, the student is not able to compile and run the code during the exam, nor is staff able to run it afterwards in connection with grading, so this type of task is manually graded. However, it does support the following features:

- A monotype font suitable for code, and syntax highlighting for some much used programming languages
- Other syntax related support, such as automatically giving an end parenthesis for each start parenthesis, and automatically making indents where appropriate, for instance in Python if the previous code line ended with a colon.

All in all, then, Inspera Assessment has better question type support both for Parsons problems and code writing than what Blackboard has, but still with substantial limitations. The user interaction for drag and drop questions is somewhat tedious for students, especially if reordering, and for teacher authoring of questions it is even more tedious. For code writing questions, both have the shortcoming that the code will not run and must be manually graded, and Blackboard does not even have syntactic support. Hence, both Blackboard and Inspera could clearly be made much more usable for handling these question types if there were plugins specifically targeting them.

3.3 Other tools

Table 1 gives a summary of the possible support for Parsons problems and code writing problems in various tools. In addition to Inspera and Blackboard, other tools worth looking at are the e-exam tool WISEflow (a competitor to Inspera) and general LMS tools Canvas and Moodle (competitors to Blackboard). The authors gathered information about these tools from web-documentation since they do not have direct access for these tools in their institution. Our findings show that Blackboard does not support drag-and-drop functionality while all the other tools support this feature. However, these tools only support the basic functionality of drag-and-drop into text and image, which is not ideal for Parson problems. Code writing is supported in Inspera and Moodle, moreover it seems Moodle has better support than Inspera. Both Moodle and Inspera support code writing with syntactic support (e.g., indentation and code highlighting). In addition, Moodle has an external plugin, Coderunner that allows students to run their programs during exams and teachers to run programs in order to grade student's answers. Limitations of the functionalities in tools can be improved further by third-party extensions and plugins with the adoption of digital ecosystems.

Table 1. Tool support summarized.

Tool	Parsons problems	Code writing	Import/export questions	Plugins
Blackboard	Lacks drag&drop	No specific support (free text)	QTI, LTI	LTI, Google Apps SafeAssign
Inspera	Has drag&drop, but not ideal	Only syntactic support for code [27]	QTI, LTI	Atlassian Jira
Canvas	Has-drag&drop but not ideal	No specific support (free text)	QTI, LTI	LTI, Facebook, Google Drive, Twitter, Tinychat Google Docs, Kaltura, LinkedIn, Canvasdocs
WISEflow	Has-drag&drop but not ideal	No specific support (free text)	QTI, Canvas, Moodle XML, Blackboard V6-9	
Moodle	Has-drag&drop but not ideal	Syntactic support, Code runner support	QTI, LTI, GIFT Moodle XML, XHTML, LTI	SEB Quiz Access, Coderunner Rule, LTI, Turnitin, Plagiarism

4 Towards a digital ecosystem

Tools like those discussed in section 3 can import/export questions in the QTI (Question and Test Interoperability) format [28]. So for authoring of drag-and-drop questions (which was somewhat cumbersome in Inspera), a possible way to improve the support would be to make a stand-alone authoring tool that could generate questions as QTI files, then to be uploaded to Inspera, for instance as suggested by [29]. In Blackboard, such an authoring tool would not be of much use, since the question type is not supported. Hence, Blackboard would need an integrated plugin supporting the question type, and an integrated plugin would probably appear better for the user of Inspera, too, especially for students solving the tasks, since the user interface could then be improved with custom features for Parsons problems. A plugin might also be a possible solution for better support of code writing questions in both tools (e.g., for the student, ability to compile and test the code during the exam; for the teacher, support for automated testing and grading of delivered code).

Currently, Inspera offers REST-based APIs to enable the third-party developers to integrate the additional functionalities and a Custom Interaction API that allows customers to build specialized question types. It supports stimuli elements with JavaScript and mathematical tools such as Geogebra and Desmos. These specialized question types can still be exchanged through QTI specification and the IMS Global Assessment Custom Interactions specification.

As the digitization of the exams increased, the need for technology for exams is also rapidly increasing. However, the usability of a digital exam system highly depends on the simplicity of the system. Also, users are sometimes forced to use several systems, not well integrated. For instance, in the authors' own university Blackboard is actively used as an LMS while Inspera is used as an assessment tool. The key requirements from teachers in the computer science department at our university that are ecosystems related include:

- Teachers want to have some exercises using the Inspera UI rather than Blackboard's, to give the students more accurate exam practice. Preferably, students should then be able run Inspera via Blackboard, so that Blackboard could still automatically register who has delivered the exercise.
- Concerning the import and export of contents, teachers may want to use last year's exam questions as exercise questions the next year. However, while Inspera can export questions in QTI 2.1 format, Blackboard (at least the version in our university) for some reason only seems to support the older QTI 1.2 standard.

In a well functioning software ecosystem, the platform system would have open APIs for external third-parties to develop plug-ins on top of the platform. This type of solution has several advantages over monolithic exam systems. García-Holgado and García-Peñalvo [30] explained that technological ecosystems could be considered as a framework to develop technological solutions where information and the human factor are the centre of the system. One of the main advantages with such an ecosystem is the flexibility it provides to institutions to integrate new software components within their workflows to support emerging needs.

The key requirements from teachers could be fulfilled to some extent with the current plug-in support by Inspera: (i) Integrate contents and external tools into LMS. Inspera supports sharing of the contents through the IMS Learning Tools Interoperability™ (LTI) plugin. LTI is an interoperability specification which facilitates full integration between Inspera and Blackboard. With LTI support, Inspera can be launched as a tool from Blackboard, which allows students to take exams directly through Blackboard. This feature is currently supported in Canvas, Blackboard, and Moodle [31]. (ii) Sharing of the contents across e-learning platforms. Issues with import and export questions can be reduced with more updates in the versions of interoperability specifications of platforms and tools [32]. In [30], the authors addressed the problem of sharing questions across departments in university in the e-learning context. They argued that although the technological ecosystem provides tools to facilitate communication between departments, employees are not utilizing the tools.

Presently, Inspera only supports sharing questions among teachers in the same university – for wider sharing, one must export and import. Of course, one deterrent against easier sharing could be increased fear for question leakage, i.e., confidential exam questions being disclosed to candidates before the exam. However, it mostly seems to come down to lacking features, and a natural tendency to prioritize the basic features first: support for each autonomous teaching staff for making the exam in their course, rather than to support a wider community of teachers within a discipline in making larger question bases that can be shared and continuously quality assured and updated.

However, Inspera also has some frustrating shortcomings on the single course level. In Norway, the law says that complaint graders shall not know the grades or viewpoints of the original graders. However, in Inspera it was impossible to hide given scores on the tasks. This meant that complaint graders could not do their grading in Inspera, but instead had to receive pdf screenshots of student answers, and then had to score manually even tasks like multiple choice, that would have been auto-scored in Inspera – with higher work-load and increased risk of error as a result. Fixing such issues will of course have a higher priority for the next release than more ambitious ideas supporting disciplinary communities. In Norwegian universities, Inspera must also be integrated with FS (Common Student System), a legacy system used for the administration of students in universities. Both the LMS and the e-exam system will fetch information from FS (e.g., which students are enrolled, registered for the exam, etc.) and send information back to FS (e.g., grades). The legacy system is not directly seen by students or teachers, but by administrative personnel – for instance it also contains the link between anonymous candidate numbers used during exams and the students' identities.

The implementation of digital exam ecosystems involves a higher degree of complexity due to the integration of different components that should evolve both individually and collectively. Although the REST APIs aids the developer, lack of the framework and design patterns makes the integration with plug-ins more difficult. A framework for technological ecosystems will consider all aspects related to integration, interoperability, and the evolution of the components [33] thus forms the well-developed open ecosystem. Several frameworks and methods were discussed in the literature. For instance, A framework can be designed using architectural patterns using the Business Process Model and Notation (BPMN) [30]. García et al. proposed a service-based framework connecting Moodle LMS and Basic LTI (BLTI) [33]. Consequently, it could ease commercial vendors and free software developers to make plugins supporting the authoring, solving, and grading of various question types.

5 Conclusion

Several advantages have been discussed in literature about e-assessments, and today many tools are available for course management and assessments. Although many e-learning/ e-assessment tools are available, only a few support programming exams. In the paper, we discussed our practical experience with programming questions in Blackboard Learn and Inspera Assessment tools, particularly focusing on Parsons problems (i.e., drag- and–drop questions) and code writing questions. Our observations revealed that currently Inspera, Moodle, Canvas, WISEflow supports drag-and-drop questions but not ideal for programming using Parson problems. Also, there is a low-level integration between Inspera and Blackboard for programming exams. The improvements can be made further with the transition of a monolithic digital exam system to digital exam ecosystem by opening APIs though it requires a lot of changes internally and outside institutions. However, open APIs alone cannot be able to improve e-assessments, without the support of frameworks, and architectural designs that explain software updates, security policies, access permissions etc. Though many papers discussed ecosystem phenomenon in e-learning, its implementation on the digital exam is still in infancy. This paper has initiated the concept of the ecosystem in the digital exams area focusing on programming exams.

The paper still has some limitations: It discussed only details of the tools used in authors' university, Inspera and Blackboard, since they have direct access to only these tools. Currently, there are many tools available for digital assessment; the study of every tool would require more time for research and cost (to buy licenses for tools). Moreover, students and teachers are adapted to the tools they use, so it is more convenient to receive their feedback. The findings from this study are based on the author's practical experience. Hence, this study can be improved in the future by more quantitative and qualitative research in academia and industries, especially on the perspective of a digital ecosystem.

References

1. Fluck, A., *An international review of eExam technologies and impact.* Computers & Education, 2018. **132**: p. 1-15.
2. Spector, J.M., et al., *Technology enhanced formative assessment for 21st century learning.* 2016.
3. Veiga, W., et al. *A Software Ecosystem approach to e-Learning domain*. in *Proceedings of the XII Brazilian Symposium on Information Systems on Brazilian Symposium on Information Systems: Information Systems in the Cloud Computing Era-Volume 1*. 2016. Brazilian Computer Society.

4. Stanley, J. and G. Briscoe, *The ABC of digital business ecosystems.* arXiv preprint arXiv:1005.1899, 2010.
5. Nachira, F., P. Dini, and A. Nicolai, *A network of digital business ecosystems for Europe: roots, processes and perspectives.* European Commission, Bruxelles, Introductory Paper, 2007. **106**.
6. Jansen, S. and M.A. Cusumano, *Defining software ecosystems: a survey of software platforms and business network governance.* Software ecosystems: analyzing and managing business networks in the software industry, 2013. **13**.
7. Kallinikos, J., Aleksi Aaltonen, and Attila Marton, *The ambivalent ontology of digital artifacts.* Mis Quarterly, 2013. **37**(2): p. 357-370.
8. Uden, L., I.T. Wangsa, and E. Damiani. *The future of E-learning: E-learning ecosystem*. in *Digital EcoSystems and Technologies Conference, 2007. DEST '07. Inaugural IEEE-IES*. 2007.
9. Oskar, P., *Software ecosystems and e-learning: recent developments and future prospects,* in *Proceedings of the International Conference on Management of Emergent Digital EcoSystems %@ 978-1-60558-829-2*. 2009, ACM: France. p. 427-431.
10. Marti, R., M. Gisbert, and V. Larraz. *Technological learning and educational management ecosystems. Thirteen characteristics for efficient design*. in *EdMedia+ Innovate Learning*. 2018. Association for the Advancement of Computing in Education (AACE).
11. Parsons, D. and P. Haden. *Parson's programming puzzles: a fun and effective learning tool for first programming courses*. in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. 2006. Australian Computer Society, Inc.
12. Sheard, J., et al. *Assessment of programming: pedagogical foundations of exams*. in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. 2013. ACM.
13. Simon, et al. *Introductory programming: examining the exams*. in *Proceedings of the Fourteenth Australasian Computing Education Conference-Volume 123*. 2012. Australian Computer Society, Inc.
14. Sheard, J., et al. *Exploring programming assessment instruments: a classification scheme for examination questions*. in *Proceedings of the seventh international workshop on Computing education research*. 2011. ACM.
15. Denny, P., A. Luxton-Reilly, and B. Simon. *Evaluating a new exam question: Parsons problems*. in *Proceedings of the fourth international workshop on computing education research*. 2008. ACM.
16. Helminen, J., et al. *How do students solve parsons programming problems?: an analysis of interaction traces*. in *Proceedings of the ninth annual international conference on International computing education research*. 2012. ACM.
17. Ericson, B.J., L.E. Margulieux, and J. Rick. *Solving parsons problems versus fixing and writing code*. in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. 2017. ACM.

18. Harms, K.J., J. Chen, and C.L. Kelleher. *Distractors in Parsons problems decrease learning efficiency for young novice programmers*. in *Proceedings of the 2016 ACM Conference on International Computing Education Research*. 2016. ACM.
19. Guzdial, M. and B. Ericson, *CS Principles: Big Ideas in Programming*. 2014: RuneStone Academy.
20. Morrison, B.B., et al. *Subgoals help students solve Parsons problems*. in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 2016. ACM.
21. Ihantola, P. and V. Karavirta, *Two-dimensional parson's puzzles: The concept, tools, and first observations.* Journal of Information Technology Education, 2011. **10**: p. 119-132.
22. Kalelioğlu, F., *A new way of teaching programming skills to K-12 students: Code.org.* Computers in Human Behavior, 2015. **52**: p. 200-210.
23. Lee, Y.Y., N. Chen, and R.E. Johnson, *Drag-and-drop refactoring: intuitive and efficient program transformation*, in *Proceedings of the 2013 International Conference on Software Engineering*. 2013, IEEE Press: San Francisco, CA, USA. p. 23-32.
24. Tsai, C.-Y., *Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy.* Computers in Human Behavior, 2019. **95**: p. 224-232.
25. Gupta, S. and A. Gupta. *E-Assessment Tools for Programming Languages: A Review*. in *Proceedings of the First International Conference on Information Technology and Knowledge Management*. 2018.
26. Forment, M.A., et al. *Interoperability for LMS: The Missing Piece to Become the Common Place for Elearning Innovation*. 2009. Berlin, Heidelberg: Springer Berlin Heidelberg.
27. Assessment, I. *Programming - Knowledge Base - Inspera*. Available from: https://inspera.atlassian.net/wiki/spaces/KB/pages/57311556/Programming.
28. Global, I. *Question and Test Interoperability (QTI): Overview*. Available from: https://www.imsglobal.org/question/qtiv2p2/imsqti_v2p2_oview.html.
29. Jørgensen, J. and S. Kvannli, *Efficient generation of Parsons problems for digital programming exams in Inspera*, in *Department of Computer Science*. 2019, NTNU: Trondheim.
30. García-Holgado, A. and F.J. García-Peñalvo, *Architectural pattern to improve the definition and implementation of eLearning ecosystems.* Science of Computer Programming, 2016. **129**: p. 20-34.
31. Inspera, A. *Assessment technology standards*. Available from: http://www.inspera.com/standards.
32. Dagger, D., et al., *Service-oriented e-learning platforms: From monolithic systems to flexible services.* IEEE Internet Computing, 2007. **11**(3): p. 28-35.
33. García Peñalvo, F.J., et al., *Opening learning management systems to personal learning environments.* Journal of universal computer science: J. UCS, 2011. **17**(9): p. 1222-1240.