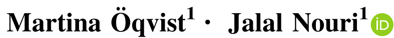
J. Comput. Educ. (2018) 5(2):199–219
https://doi.org/10.1007/s40692-018-0103-3

CrossMark

Coding by hand or on the computer? Evaluating the effect of assessment mode on performance of students learning programming

Martina Öqvist[1] · Jalal Nouri[1]



Received: 24 January 2018 / Revised: 23 March 2018 / Accepted: 27 April 2018 /
Published online: 3 May 2018
© The Author(s) 2018

Abstract Programming courses suffer from low retention rates, believed to be caused by difficulties in learning programming concepts. Another possibility relates to how programming ability is assessed. Although extensively used, pen-and-paper is arguably not the best way to assess a student's programming ability. Although previous studies have chosen to implement lab exams as a replacement for pen-and-paper examinations, little consideration has been made to evaluate the extent of the exam mode effect in relation to the use of computers during summative assessment in programming courses, and to what degree this could affect the student's performance. This study aims to answer to: *Are there any differences in performance between students using a computer, compared to students using pen-and-paper during summative assessment of programming ability among novice programmers?* This could aid teachers to better assess students' programming ability in novice courses, which could in turn aid in student retention. An experimental approach has been applied. 20 students participated and were divided in two groups that were giving the same programming problems that was either solved and examined through pen-and-paper assessment or through computerised assessment. While some differences between the groups were noted, the overall results demonstrate no statistical significant difference between the groups in terms of performance.

Keywords Programming · Coding · Assessment · Computer assessment

✉ Jalal Nouri
jalal@dsv.su.se

Martina Öqvist
maoq2907@su.se

[1] Department of Computer and Systems Sciences, Stockholm University, Kista, Sweden

Introduction

Since the first computer program was run on the Manchester Small-Scale Experimental Machine on the 21st June in 1948 (Enticknap 1998), to the first message sent through ARPANET in 1969 in the infancy of what was to become the Internet (Leiner et al. 2016), Computer and Information Technology (IT) has continued to advance and is now a huge part of the daily life for many. The expanded use of computerised systems has led to an increase in the demand of talented software engineers and developers (Masters in Software Engineering 2016; Statistics Sweden 2016), to help build and maintain both today's and tomorrow's software systems.

However, the demand of qualified workers outweighs its supply in various degrees, and the phenomenon is not only prevalent within the national borders of Sweden. The same predicament has been reported in the United States (Simpson 2016), in Canada (Arellano 2015), the United Kingdom (Anderson 2015), as well as various other countries in the European Union (van Heur 2016). The high demand for qualified personnel within the IT sector in Sweden cannot be accredited to an ageing group of practitioners (Statistics Sweden 2016), as the Swedish IT market displays a rather young workforce in comparison to e.g. the health-care sector. The U.S. Department of Labour came to the same conclusion that the high demand of workers is rooted in the expansion if the IT sector rather than the workforce ageing or leaving the industry (U.S. Bureau of Labor Statistics 2013).

In order to increase the amount of IT graduates in Sweden, efforts have been aimed at getting more students to enrol in Computer Science programs by e.g. lowering the programs' entry requirements (Swedish National Agency for Higher Education 2001), thus increasing the throughput. However, ensuring that students graduate, or at the very least obtain enough knowledge to ensure their own employability on the market, should be considered just as important. Although the number of students that confer degrees needs to increase, it ought to be done without passing students that do not possess the necessary tools to continue their education (Daly and Waldron 2004; Jacobson 2000), or who lacks the knowledge to work in the industry as a programmer (Atwood 2007).

Student retention is another issue related to the low number of graduates. Low student retention has been reported as being especially prominent for programming subjects in higher degree studies (Bennedsen and Caspersen 2007; Bergin and Reilly 2005; Jacobson 2000). Robins et al. even claim that programming courses have the highest dropout rates in comparison with other courses, despite the extended popularity of programming subjects in response to the increase in demand for educated programmers. The main reason that contributes to the low retention rates is believed to be the subject itself: learning to program (Begosso et al. 2012; Sarpong et al. 2013). As iterated by Gross and Powers (2005), programming is a skill that is considered difficult to attain, and generally requires up to 10 years of experience for a novice to become an adept programmer (Winslow 1996).

Although it is hard to dispute that learning to program is challenging, another important issue that has arisen as a response to the issues of low retention rates and

inadequate programming performance concerns how programming ability is assessed. There exist a variety of ways to evaluate programming ability, which makes it difficult to discern which method is the most appropriate one for a specific module (Chamillard and Braun 2000). Each assessment mode, besides carrying its own set of advantages and disadvantages (Chamillard and Joiner 2001), is also associated with a probable learning outcome (Biggs 2003). Hence, it is important to choose a method that leads to the most suitable learning outcome in terms of programming ability. The ones that are most commonly used in programming subjects, according to previous studies, are written exams, individual and/or group assignments, and practical lab assignments (Chamillard and Braun 2000; Chamillard and Joiner 2001; Daly and Waldron 2004; Jacobson 2000). Even though summative assessment methods such as written exams are commonly used, many studies show that it is arguably not the best way to assess a student's programming ability (Daly and Waldron 2004; Rajala et al. 2016; Sheard et al. 2013).

The reasons for having a written exam are various, as was made apparent by Sheard et al. (2013) during their study to investigate the process for creating a written exam. However, there is a common consensus that the regular written exam reduces the risk of plagiarism that is otherwise heavily associated with individual programming assignments (Bennedsen and Caspersen 2007; Daly and Waldron 2004; English 2002; Sheard et al. 2013). Even though a regular written exam is considered as an inappropriate way of measuring programming ability among students, it can still be used to assess a student's understanding of programming concepts (Sheard et al. 2013).

One solution applied by others to counter the inherently negative effects of written exams, have been to introduce a computerised examination where students are allowed access to appropriate programming environments and the ability to use the compiler to allow students to verify their solutions (Chamillard and Joiner 2001; Haghighi et al. 2005; Rajala et al. 2016). In a study by Haghighi et al. (2005), the lab exam approach was evaluated using a set of students. Although the students believed that one of the most useful aspects of the lab exam was the access to the compiling and debugging facilities, if the solution that they were writing did not compile properly, this added to the overall stress of the exam. Haghighi et al. (2005) also made the assumption that some students might be more comfortable using pen and paper during an exam, regardless of topic. Rajala et al. (2016) make the opposite assumption that students are likely more familiar with writing code using a keyboard and a proper editor, and that using a pen and paper for coding is a slow and tedious process. Although these studies come to the same conclusion, they do not evaluate the actual effect of the exam mode. By just using a computer for the exam, rather than pen and paper, this change might have a proficient effect on performance, without the added stress of possible compilation errors. There are also other types of questions that cannot be assessed correctly by giving students access to compilers during an exam, such as being able to predict output of a program based on already written code (Haghighi et al. 2005).

The overall purpose of this study is to evaluate whether changing the exam mode to allow for the use of a computer, with access to a supported editor, in itself would have a positive effect on novice programming students' exam grades. The objective

of this study is therefore to evaluate the exam mode effect when assessing programming ability among students.

The research question this study aims to answer is: *Are there any differences in performance between students using a computer, compared to students using pen-and-paper during assessment of programming ability among novice programmers?.*

Background

Previous studies with computerised summative assessment

As many studies have pointed out, being able to test their solutions during a written exam could inherently aid the student in providing a proper solution, and support the teachers during their assessment process of the student's programming ability (Chamillard and Joiner 2001; Haghighi et al. 2005; Rajala et al. 2016). However, adding the criteria that a program must run in order to submit it, as the test was implemented by Jacobson (2000), puts extra pressure on the student during an already stressful situation such as a time-constrained written exam (Haghighi et al. 2005). Although it has been mentioned that using pen-and-paper to code is an artificial situation and is therefore inappropriate (Bennedsen and Caspersen 2007), the effect of merely changing the environment from using pen-and-paper to computer has not been measured.

Studies in the area that supports the assumption that a digital or practical exam more accurately assesses a student's programming ability, in comparison with pen-and-paper exams, make this claim by using either exam marks or the number of students that pass the course before and after the implementation, as the main measurement for comparison (Bennedsen and Caspersen 2007; Chamillard and Joiner 2001; Daly and Waldron 2004). Though Bennedsen and Caspersen (2007) acknowledge that there are other factors that could be attributed to having improved the exam rates after an implementation, the studies do not directly compare the exam mode effect.

In a study by Lappalainen et al. (2016), the pen-and-paper exam was compared to a computer exam through the use of the rainfall methodology. This allowed students to rework their initial paper solutions on a computer after the exam had concluded. The aim of the study was to see if the use of an Integrated Development Environment (IDE) would help students to correct the errors in their previous answers. An IDE contains the means to debug and compile code, allowing the students to both see what was erroneous in their initial solution, as well as give them the opportunity to fix these errors by themselves. The students were also given access to a test suite in order to verify that their program generated the correct output. The results indicate that the compiler can help the student correct their own mistakes, and that using both a computer and paper granted the students the means to provide better solutions, than what they had done previously by solely using pen-and-paper. (Lappalainen et al. 2016) But as a major part of the solution was already written by hand when the student was given access to a computer, it is difficult to directly compare the two modes. Besides providing the opportunity to debug code,

an IDE contains several other tools and features that can aid the students when programming, such as auto-completing code and providing code suggestions.

Haghighi et al. (2005) claimed to directly compare paper exams to that of a computerised exam in their study, by allowing students to first write part of the exam on paper and then the other part using a computer. Their results show that the students perceived that it was more difficult to answer the exam questions using a computer in comparison with using pen-and-paper (Haghighi et al. 2005). However, these results can be explained by the inherently different question types given during the two instances of the exam. In order to accomplish what Biggs (2003) call *constructive alignment* between the mode of assessment and the learning goals, the paper exam consisted of questions that aimed to evaluate the students' ability to explain programming concepts and theory, the students' ability to predict output from a given set of code, and their ability to design and draw a solution. The computer exam, however, aimed to evaluate the students' ability to write a small program, make modifications to existing code, and to add functions to an already pre-written program. (Haghighi et al. 2005) In this context, any comparison between the two given tests will reflect the differences in what is being assessed. Another conclusion that can be drawn from this study is that the students perceived the computer exam as being more difficult, perhaps considering the process of writing code to be more demanding of the student than that of answering questions related to programming concepts.

Correlation of memory and environmental context

Though memorisation is considered an inappropriate learning strategy in programming courses (Begosso et al. 2012), it is still used during assessment of programming skills (Thompson et al. 2008). In a study that aimed to provide a new interpretation of categorising programming assessment according to the cognitive domain, Thompson et al. (2008) found that memory retrieval (including both recall and recognition) was related to programming assessments as a means to recall syntax rules. This included the knowledge of how to write and use these rules, as a way to recognise programming constructs in new contexts, such as interpreting a given code paragraph. Hence, while not being recommended as a learning strategy (Begosso et al. 2012), recall is still used in order to retrieve information about syntax, algorithms, design patterns, as well as their role in previously seen solutions (Thompson et al., 2008).

The pressure of recalling and applying correct syntax has been reported to interfere with the novices' ability to provide semantically correct code (Pane and Myers 1996). While knowing, or rather understanding that syntax is important in order for the students to be able to practically display their knowledge, it should not remove focus from the student's ability to communicate their problem solving ability. For this purpose, and within this department, whether a student choose to use size() or length() in order to determine the number of characters in a String, is not considered as being important in order to understand the solution provided by the student. While it might not compile in a regular IDE, their ability to solve the presented problem is still apparent and should remain the focus of the exam, no

matter if it is completed digitally or using pen-and-paper. The assumption that is made here is that in a regular non-exam environment, the student would be able to correct this mistake effortlessly (either through the IDE's own code correction, or using a simple Java API look-up). That the student understands what it is that they are trying to achieve, and knowing that a method exists that performs this very operation, is what is considered relevant in order to assess their programming ability. This assumption is supported by Fay (as cited in Pane and Myers 1996), who suggests that the students' syntactic ability should take second place in order to direct their focus to use correct programming semantics.

Limited working memory has also been suggested as a reason that novices do not perform well in programming courses, and that novices would perform better if the dependency on their working memory is lessened. This could be achieved by e.g. allowing necessary information to be visible to the novice (Anderson as cited in Pane and Myers 1996). This premise is also supported by Haghighi et al. (2005), who concluded that minimising reliance on memory and giving more access to resources resulted in better performance during assessment.

Another reason to believe there potentially is an effect of using computers during the exam, instead of pen-and-paper, can be derived from the theory of environmental context-dependent memory recall (Smith and Vela 2001). Smith and Vela (2001) show that *the environmental context of which memory testing occurs affects memory in a context-dependent manner*. Lieberman (2011) declared that an explanation to this phenomenon could be that the contextual cues in the environment becomes part of the memory during the process of recording it. However, as argued by Isarida and Isarida (2007), the context would only facilitate recall if there was a relationship between the context and the memory to retrieve, and that the strength of the context-effect would increase the more a memory was processed within the associated context. If the changes in context between the time of recording the memory and the time of retrieving it are comparably large, this would make it more difficult to retrieve the memory (Lieberman 2011).

Kolers and Ostry (as cited in Lieberman 2011) found evidence that suggests that even some time after reading a passage, the font in which the passage was written could still be accurately recalled, and as such, worked as a contextual cue for retrieving the information earlier read. This is supported by Wilson (as cited in Clariana and Wallace 2002), who reported that text font accounted for measured differences in exam mode effect between computers and pen-and-paper. Besides font, the colour of the background has also been reported to have an effect on memory retrieval, though to a limited degree (Lieberman 2011).

Memory recall is the process of retrieving and reproducing a memory without cues, whereas recognition is based on retrieval of memory after being given association cues (Lieberman 2011). In relation to exam questions, recall (or rote) is associated with essay types of questions, whereas recognition is associated with multiple-choice questions (Biggs 2003). In experiments, recognition has been found to be more accurate than recall, especially if some time has passed between the moment of learning and the time of testing (Lieberman 2011). Lieberman (2011) proposes that, in order to recall a specific piece of information, the given cue for retrieval (e.g. the exam question) is not always enough in order to fully recall the

memory. However, when given more ques (e.g. the proposed alternatives for a multiple-choice question), this might be enough in order to retrieve the sought after information from memory. Hart (as cited in Atkinson and Shiffrin 1968) calls this the "tip-of-the-tongue" phenomena, where someone who could not previously recall certain information, in many cases, were still able to choose the correct answer given a set of alternatives. The information was still retained, although unreachable at the initial time of retrieval. This effect was described by Perkins and Martin, who reported that programming novices, while they initially appeared to not possess certain knowledge, could still evoke the information through the use of probes. Hence, while the novices were not able to correctly recall the answer, they were able to provide an answer through recognition. Knowledge that novices were unable to access, had forgotten, or applied in an incorrect manner, were termed as "fragile knowledge" by Perkins and Martin.

For the purposes of this study, the most important aspect of environmental context regards that of the environment in which the actual programming is taking place. When students learn and practice programming they are doing so with access to a computer with an IDE, including the internet and various other resources. Later, when students' programming ability is assessed, they are presented a pen-and-paper exam. These two environment differ greatly, which in turn could have a negative impact on their ability to perform. The code the student will write during the exam might, in itself, not provide the right contextual cue in order to retrieve all the information needed in order to write a complete solution. While it might be inconsequential, the answers written by the student would not be in the same font as the code the student would be used to reading.

Support for readability

As stated by Rajala et al. (2016), the students are most likely used to writing code by using a computer, where access to a keyboard and a proper editor support their programming through automatic indenting, and with the added support for readability through features such as code highlighting. Although a computer could potentially provide enough support in itself, using a normal text editor, such as notepad for Windows, is not enough in order to support for novices' when assessing programming ability (Dillon et al. 2012).

In a study by Sarkar (2015), syntax highlighting, i.e. meaningful colouring of text in coding environments, was reported to increase readability and comprehension speed, particularly for novice programmers. Using technology for eye-tracking, it was concluded that syntax highlighting reduced context switches, as the colouring permitted the participant to focus on other important aspects of the code rather than the key words. The key words were instead perceived through their peripheral vision (Sarkar 2015). In comparison, when using the same text without colouring, the participants eye fixation was not directed, and all text were considered as of equal importance. This increased the number of contextual switches needed by the participant to read and interpret the code. (Sarkar 2015) The use of syntax highlighting is also supported by Pane and Myers (1996), who use the term

signalling. According to Pane and Myers (1996), *signalling improves comprehension of the signalled material, at the expense of non-signalled material.*

While reading speed was reported to increase through highlighting, the effect is only noticeable if the colouring is recognised by the reader (Green and Anderson as cited in Sarkar 2015). As an implication if this, using colour coding that the novice programmer is unused to, might not provide the same favourable results as was reported by Sarkar (2015).

Support for structuring solutions

Whether there is a lack of knowledge or that the students' current knowledge could be considered as fragile, it has been suggested that these limitations are made apparent through difficulties in expressing, planning and designing problem solutions. Though these difficulties should not be related to issues in understanding features pertaining to specific programming languages. In order to structure a solution, a novice programmer needs to make use of strategies which they have often been found to lack. According to Biggs (2003), the most likely kind of learning associated with regular written exams (called essay exams by Biggs (2003)) relates to memorisation, or rote, and the ability to effectively structure a written answer. Assuming that the process of writing code by hand resembles that of writing an essay by hand, it is likely that the student's ability to structure their answer is in focus, rather than their ability to provide a probable solution for the task at hand.

In order to solve a programming problem, a student has to first gain a thorough understanding of the problem itself, and then understand what type of solution would best be suited to solve this problem (Begosso et al. 2012). Winslow (1996) described the problem solving process as a series of steps, starting with (1) understanding the problem, (2) determining a solution in some form, before determining how to apply it in a more computer compatible form, (3) translating the previously designed solution into a computer language, before (4) testing and debugging the program code. According to Begosso et al. (2012), this requires the student to be able to rearrange their thoughts, which is believed to require great diligence from the student in order to succeed. Assuming that the student needs to rearrange both thoughts and, most likely, the code being written during their problem solving process, any need for editing will halt the student's progress. Thus, their problem-solving process is not supported by the current pen-and-paper exam mode, as it imposes a top-to-down approach to solving a problem. This conclusion is supported by Sarpong et al. (2013), who named "lack of programming planning" as one of the many identified reasons to why students fail programming courses. This conclusion is also supported by Robins et al. who noted that novices in comparison with experts did not spend much time planning their solutions before starting to code. Winslow (1996) reported that novice programmers in particular look at code from a line-by-line viewpoint, rather than any other type of meaningful chunk. To be able to properly assess students' programming ability, these novice strategies need to be taken into account when choosing an appropriate exam mode.

When reading the code for a computer program, it is easy, and even common, to misinterpret what a program does based solely on the formatting. The use of incorrect formatting invites the reader to provide an erroneous interpretation of the written code. (Pane and Myers 1996) Similarly, using incorrect indentation could lead the reader to confuse which section a coding paragraph might belong to, by either making the assumption that it belongs to different coding structures, or misinterpret what should correctly be read as a separate entity (du Boulay as cited in Pane and Myers 1996).

When writing by hand, it might be considered more difficult to write code with correct formatting, as this is a feature normally performed automatically by the IDE. Similarly, if the student has used incorrect formatting, this might invite them to read their own written code in an incorrect way. As stated by Davies (as cited in Pane and Myers 1996), the end product seldom conforms to the sequence in which it was generated, and will most likely be subjected to several revisions, as is common when writing code. Even if the student has been able to provide an adequately indented answer, if there is any need for alterations, editing the written text on paper will subject the solution to possible re-indentation and increase the probability of introducing errors related to structure of the program.

Methodology

By using an experiment, quantitative data will be collected from students by comparing test results between the test group (digital exam mode) and the control group (pen-and-paper exam mode). As previous studies have already proven that access to compiler and test suites have a positive correlation with students' test results (English 2002; Jacobson 2000; Rajala et al. 2016), these features will not be included as variables in this study. Instead, this study aims to see the effect of the exam mode if only minimal computer support is given.

The experiment was carried out in the department's educational facilities in Kista (Stockholm, Sweden). Before the experiment began, the participants were informed about the structure of the test assignment and what was expected of them. It was thoroughly communicated that it was the method that was being tested and not their individual ability, both in order to discourage cheating and to relieve them of any potential test anxiety. Once this was done, the control group was led away to the room in which they were supposed to execute the test. One test guard was assigned in order to assure that the student did not try to cooperate in order to solve the programming tasks. Once a student had finished the test, the guard ensured that they had received all written answers, before recording their hand-in time. Once this was done, the student was allowed to leave.

After the control group had started writing their tests, the test group was led to one of the computer rooms where they were told to log into their individual computer accounts, and to open a new tab in Chrome in order to enter the URL for the test platform. Upon entering the site, the students were asked to log into the test session using their participant ID found in the assignment hand-out. They were told to answer the tasks using the online editors, one for each question, and that their

answers were saved continuously and when they navigated to another page. Another guard was assigned to observe this group to discourage cheating. Once a student had finished the test, the guard ensured that all of their answers had been recorded in the database, and recorded their hand-in time as 3 min after their lastly saved answer. The student was then allowed to leave the room.

Two questionnaires were distributed to the participant students. One before the experiment for capturing programming experience to be used for randomised strata sampling, and another after the study to collect data in regard to the students' impression of the programming test.

Research design

Study setup

The independent variable that was examined was the exam mode. The control and the test group differed in respects to how they provided their solution to programming tasks during the test. Whilst the control group used pen-and-paper to write their code, the participants of the test group were given access to a digital programming environment on a computer. The environment, in the form of a simple editor, was mainly meant to support editing, structuring and readability of code. The features that provided this support allowed the students the ability to both write and delete written code, support auto-indentation and the ability to move and remove coding paragraphs, and support syntax highlighting, line numbering, and the use a familiar type-face.

Both groups were also provided a hand-out consisting of the programming tasks, a java class, and a small booklet with an extract of the necessary classes and methods as a way to simulate access to the Java API during the test.

Other common features that are prevalent in regular IDE's, for example auto-completion of code, debugging, and code suggestions, were not included in the test. These features were considered to provide too much support in comparison with the control group, which could have led to skewered results in favour of the test group.

The dependent variable that was measured and compared was the test results of the two groups, to see if there were any major differences that could be correlated to the exam mode. This included the overall grade of the test, and potential variance in results for different types of questions/tasks.

The new factor that was introduced was the digital programming test, which included a computer and an editor. As most "off-the-shelf" IDE's support a series of features that will not be tested during experiment (such as code completion, compiler and debugger), a simple open source editor was used. CodeMirror is an open source project under the MIT licence that allows for a JavaScript written editor to be implemented as a text area in a website. The editor allows for some customisation when it comes to supported features and the colouring schema used for syntax highlighting. As it is an editor (and not an IDE, the code cannot be compiled or debugged through the application itself.

For the purpose of this test, and since other options were limited, a small web application was created. The application allowed for the students' answers to be

periodically saved to a database, as a way to ensure that no loss of data would occur should something happen to crash the application. The students were able to go back and forth between questions, until they chose to submit their answers at the end of the exam.

Test platform

Since there is a lack of studies in the area of assessing programming ability among students without the support of a compiler, no tool could be found that sufficiently supported the aim of this study. While tries were made to remove any and all functionality from Eclipse to support the specific functions that this study aimed to test, it was not possible to completely disable all parts without allowing access to turn them back on again. The same goes for other IDE's that were tried. While certain editors, such as Sublime Text (2017), allow for the type of functionality sought after, they require costly academic licenses. For that reason, the choice was made to create an implementation that would not only fulfil the required support for readability and structuring code, but also support the actual test administration.

The test platform that was created for this purpose consisted of a website developed using JavaScript and a web framework called Sails (Sails.js). Sails.js is an open source Model-View-Controller (MVC) web framework developed by The Sails Company, used for building service-oriented architectures in Node.js, and allows for fast development of these types of applications. Node.js is a JavaScript runtime that is developed by Node.js Foundation, and is used to build asynchronous and scalable network applications. Sails.js has several built-in services that allows for easy database implementation, secure user authentication and prevention of cross-site scripting. Other services needed for the application, such as encrypted passwords, was easily implemented using third-party software modules.

While the researchers for this study have experience in a number of different languages, full-stack web development in JavaScript and Node.js was found to provide the most efficient tool-set in order to quickly produce a working application.

In order to support continuous saving of students' answers, a database needed to be implemented. MongoDB is a NoSQL database that is schema-less (Chodorow 2013). This means that the structuring of the data is done on an application level rather than on database level, and allows for quick changes of attributes without disrupting the already written data in the database (Chodorow 2013). As the platform needed to be developed within a tight time-frame, a flexible database was favoured in front of other stricter SQL alternatives.

In order to allow the student to code in an environment similar to that of an IDE, a third-party software module was implemented. CodeMirror is a JavaScript plugin used to implement a front-end editor similar to that of other coding environment. The module supports configuration and implementation of syntax for several different languages, as well as a number of features that were not used for this study (such as code completion). Since it is only an editor (in comparison with a regular IDE), it did not allow students to compile or otherwise test their solutions once written.

Test assignment

The assignment consisted of three tasks: create a java constructor, create a java method, and write a loop. The students were also provided with an example class to conceptualise and aid their process, which added the additional challenge of being able to read and understand already written code.

The first two tasks evaluated the level of existing knowledge the student had in regard to Java, its terminology, and how to apply it. The tasks were inspired by the study by Thompson et al. (2008). In accordance to this study, the tasks aimed to test the students' knowledge in what could be categorised as belonging to both the *remember* as well as the *apply* category. Given a code example, the student was asked to both recognise the terms used in the task description (remember) and how these concepts were to be implemented using code (apply). They were also asked to recall material from their course and apply these in a way that should be familiar to them, although perhaps not in the exact context as what they were used to. Since these tasks covered basic concepts in Java that students should have some grasp of, their solutions for these two tasks were corrected quite stringently.

The third task was a simplified (or otherwise altered) version of the FizzBuzz test, which is said to be commonly used by organisations in order to test programming ability among job applicants for programming positions (Quora Contributor 2016). The solutions for the last task were corrected on both a syntactic and a semantic level: if the student appeared to understand what they were trying to achieve, points were given for each concept that the student managed to apply. If the solution that was provided would run or not was not the main focus, since the compiler would assist in correcting any minor issues the student might have made (spelling mistakes, missing parenthesis or curling bracket, and so on), as has been proven by Lappalainen et al. (2016). This approach to correcting assignments is also supported by English (2002) who claimed that it is *unfair to expect students to produce working programs* in the same type of context that was tested as a part of this study (English 2002, p. 51).

Test group To compare the current method of assessment with that of a computer supported assessment, the test group was to be allowed access to a computer with a programming supported editor, through which they would provide their answers. The editor would afford minimal support for programming tasks in order to not pose more of a hindrance than an aid for the students, but not more support than that it would still be comparable to that of a pen-and-paper exam. The effect of the computer was the focus of the experiment, though it was not possible to test it in complete isolation, as software (operating system, applications) makes up the computer's environment as much as the hardware (keyboard, mouse, screen etc.).

In order to lessen the reliance on working memory, any information needed in order to solve the tasks needed to be made visible for the students. To accommodate this, access to a syntax summary was required. This was considered as an appropriate aid for the students as it would lessen the need for them to recall complete syntax. While it is entirely possible to provide this summary using the

digital platform, the syntax summary was decided to be provided in similar format to both groups to lessen the number variables that separated the two.

Control group The control group was to be assessed using a method similar to what they were used to having previously attempted regular written exams at the department. While the same assignments were to be given to both groups, the control group provided their answers using pen-and-paper.

Study context

This experiment was conducted using participants either currently or previously registered for the novice Java course at Stockholm University's department of Computer and System Science. The introductory programming course for novices in the Bachelor program is scheduled to proceed in late autumn for all first year students. The course is a mandatory object oriented programming course in Java for 7.5 credits (10 weeks of studies), where the exam constitutes 4 credits, and the last 3.5 credits comprise assignments (some completed individually, others in pairs). The following text is an extract from the course segment's informational page.

After completing the course, the student should be able to:

- Explain, and correctly use, basic concepts in imperative (e.g. data type, variable, selection, iteration, recursion and sub-routine) and object oriented programming (e.g. class, object, encapsulation).
- Construct algorithms for solving programming problems and implement these algorithms in Java. This includes correcting any errors encountered.
- Construct an executable object-oriented program out of a conceptual model (for example UML).
- Use classes and methods from a Class library.

For the purpose of this study, constructing algorithms for solving programming problems and to implement these in Java were the main focus of the test. The students understanding of how to correctly interpreting program codes written in Java were evaluated, as well as the use of classes and methods from a Class library, though to a very limited extent. In a sense, a student also needed to understand what happened during program execution in order to correctly interpret their own solutions, but this part of the last goal is difficult to examine in isolation due to the test's focus on writing code rather than mere code interpretation.

Sampling method and participants

An important factor that was considered when choosing an appropriate sampling method in the context of this research was that of potential biases. Since it is important that other factors are excluded, besides the one that is the focus of the experiment, it was preferable that the composition of test group and that of the control group were similar. To ensure similarity between groups, a randomised

strata sampling method was chosen. This prevented introducing researcher bias into the study but still allowed for a certain amount of similarity of participants in both groups. Applying a randomly selected sampling method is also supported by previous literature (Clariana and Wallace 2002). 20 students partook in the final experiment, 10 using pen-and-paper, and 10 using the digital platform. Questionnaire data about students' experience of programming were used to create randomised strata sampling.

Results

Test results

The average total score for the entire set of participants, and for all tasks, was 11.70 (SD = 8.59), where max possible score that could be achieved was 25.00, 5 points for each of the first two tasks, and 15 points for the last one. The results of conducted independent sample t-tests showed no statistically significant differences between students examined by paper and pen ($M = 8.60$, SD = 8.32) and students examined on a computer ($M = 14.60$, SD = 8.15), $t(18) = -1.63$, $p > 0.05$. See Tables 1 and 2 for the control and test groups scores on the different tasks.

Task 1: create a constructor

Based on the questionnaire sent after the experiment was conducted, most students found Task 1 to be of little challenge (see Fig. 1). Given a Likert scale, where 1 represented "Easy" and 5 "Challenging", the majority of the participants answered either 1 or 2. While none of the participants of the control group answered above 2, the maximum given answer for the test group was 4.

The expected answer for this task was:

```
public Character (String firstname, int age) {
    this.firstname = firstname;
    this.age = age;
    lastname = "Svantesson";
}
```

Though most students managed to contribute with a solution that solved either the entire task or part of it, one of the more common mistakes made by the participants for this task included missing to add the keyword this. The keyword is used to help the compiler to distinguish from locally declared variables (e.g. those declared as parameters for the constructor) and attributes declared in the class, when these share the same name. This included 2 contributions provided by participants in the test group and 1 in the control group.

Other mistakes included using the equality operator (==) instead of the assignment operator (=), and either misspelled or missing keywords for the data

Table 1 Scores per task for the control group

Task 1 points	Task 2 points	Task 3 points	Total
5	5	14	24
5	5	11	21
5	5	2	12
5	4	1	10
4	3	0	7
3	3	0	6
3	2	0	5
1	1	0	2
0	0	0	0
0	0	0	0

Table 2 Scores per task for test group

Task 1 points	Task 2 points	Task 3 points	Total
5	5	14	24
5	5	12	22
5	5	12	22
5	5	11	21
5	5	10	17
5	5	3	13
5	4	3	12
4	1	0	5
4	0	1	5
4	0	0	4

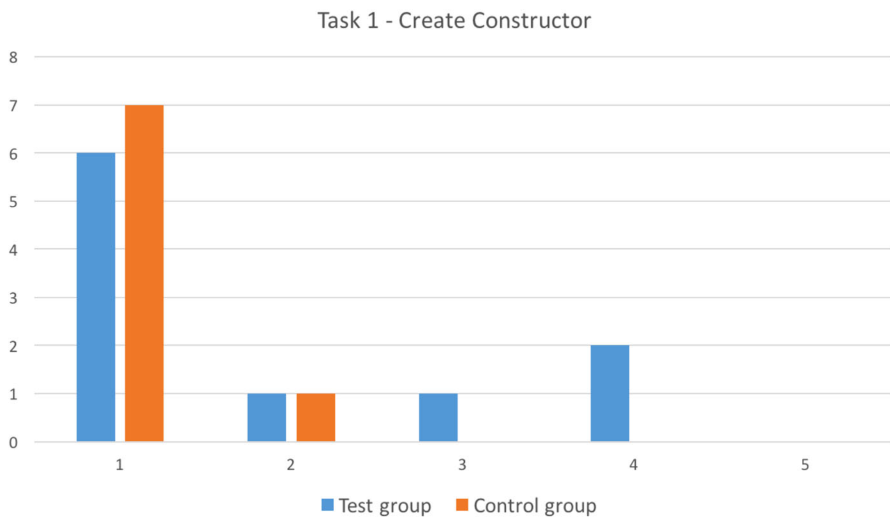


Fig. 1 Chart of the perceived difficulty of Task 1, split by group

types (String, int). While missing keywords were equally as common for both groups, misspelled data types were only found in the control group. Missing to assigning the lastname variable to "Svantesson", not setting or incorrectly setting either of the attributes firstname or age, missing or adding extra parameters to the constructor (e.g. to set the lastname attribute), were more common in the test group than the control group, which can be seen by looking on the lower calculated average for the two groups. With regard to task 1, an independent sample *t* tests showed a statistically significant differences between students examined by paper and pen ($M = 3.1$, SD = 2.07) and students examined on a computer ($M = 4.07$, SD = 0.48), $t(18) = -2.37$, $p < 0.05$.

Task 2: create a method

While over half of the students still found task 2 to be easy, the other half of the student found the task to be either somewhere between easy and challenging, or somewhat challenging. Of the students that found the task to be more challenging than the first, most of these belonged to the test group (see Fig. 2).

The expected answer for this task was:

```
public void fliesAgain () {
        onTheRoof = true;
age += 2;
}
```

Again, most students managed to contribute with a solution that solved the task or parts of it. While there were more contributions that received 0 points on this task

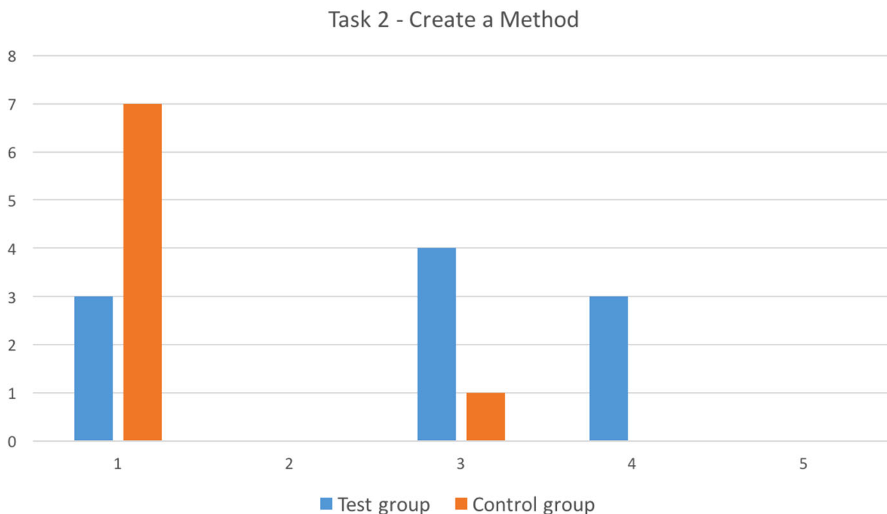


Fig. 2 Chart of the perceived difficulty of Task 2, split by group

than that of the previous one, most students received at least one or more points on the task.

Several of the students used either boolean as a return value (instead of void) for the method, or did not declare a return type at all. This mistake was equally as common in contributions provided by both groups. As an example is given for how a method call should look like, points were also deducted from several contributions that added parameters to the method. This included three contributions from the control group, and two from the test group. Eight of the solutions either did not set, or incorrectly set, the attributes onTheRoof or age. Five of these contributions belonged to the test group, and three of these to the control group. With regard to task 2, an independent sample t test showed no statistically significant differences between students examined by paper and pen (M = 2.8, SD = 1.98) and students examined on a computer (M = 3.40, SD = 2.37), $t(18) = -0.61$, $p > 0.05$.

Task 3: create a loop

The distribution of answer on the task difficulty is more distributed for task 3 than for the previous tasks. While a third of the students still found the task to be either easy or somewhat easy, one third found the task to be either somewhat challenging or challenging. Of those that found the task easy or somewhat easy, most of these belonged to the control group. Of those that thought the task to be somewhat challenging or challenging, two thirds belonged to the test group (see Fig. 3).

The expected answer for this task:

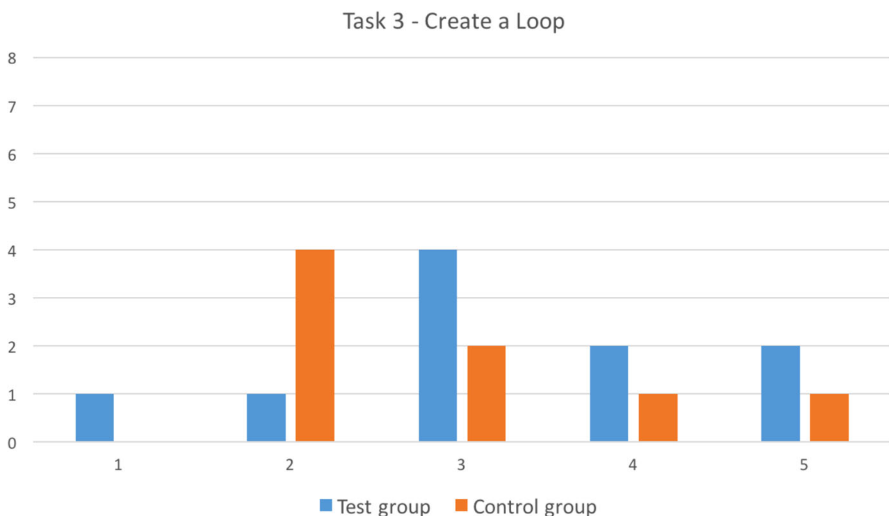


Fig. 3 Chart of the perceived difficulty of Task 3, split by group

```
for (int i = 0; i < 10; i++) {
    if (i == 0) {
        Character svante = new Character();
        svante.fliesAgain();
        family.add(svante);
    } else if (i \% 2 == 0) {
family.add(new Character("Bosse", 15);

    } else {
        family.add(new Character("Bettan", 14);
} }
```

As expected from a task with a bit higher difficulty, more of contributions were awarded few or no points. Eight of the participants were given no points for task 3, six of these from the test group and two from the control group. In order to get points for the task, one or more objects must be created and correctly added to the privately declared ArrayList family. If no object were added, the student had to either provide a sleek solution or try to follow the bonus instructions in order to be given a maximum of 5 points. The most common mistake found in the solutions was the use of the assignment operator (=) in the if-statement blocks, instead of the equality operator (==). This mistake was found on seven of the contributions, three from the test group and four from the control group. Another common mistake regarded the middle argument for the loop, i.e. the end condition of the loop. In five of the contributions, the loop either went past ten, or not up to ten, causing too many or too few objects to be added to the array. 2 of these belonged to the test group, and 3 to the control group. In regard to added bonus points, 11 of the contributions used modulo to achieve equal distribution of the objects in the array. Four of these belonged to the test group, and seven to the control group. Seven of the contributions used references sparsely or economically, two from test and five from the control group.

In two of the solutions, object "Svante" was added to the array though not on index 0. Both these belonged to the control group. Eight of the contributions followed the bonus instructions; two of these from the test group, and six from the control group. Only one student in the control group used the standard constructor to create an object of type "Svante". With regard to task 3, an independent sample t-tests showed no statistically significant differences between students examined by paper and pen ($M = 2.8$, SD = 5.20) and students examined on a computer ($M = 6.60$, SD = 5.66), $t(18) = -1.56$, $p > 0.05$.

Discussion and conclusions

While the results are inconclusive in regard to a definite answer in terms of exam mode effect with the introduction of a digital exam using a computer, it was mentioned in Haghighi et al. (2005) study that the students found the ability to test

their solution to be the most useful aspect of using a digital platform in comparison with a pen-and-paper approach. Although this study does not provide any proof in opposition to this claim, it cannot be disproved that the access to a compiler is what gave these studies the inherently positive outcome.

As mentioned by Pane and Myers (1996), the ability for students to provide semantically correct code is hindered by their ability to recall and apply correct syntax. While precautions were taken in order to make information visible to the students by allowing students an extract of the Oracle's Java Docs, as recommended by Pane and Myers (Anderson as cited in 1996), perhaps this information was not enough to alleviate the dependence on students' working memory in order for them to be able to perform well.

One aspect the students did seem to appreciate was the ability to restructure their solutions once written, as indicated by several students in the test group when asked' what was useful about the exam format?'. Allowing editing of the answers was likened to "what an actual programmer does", or "being more natural" as going back and changing parts is closer to the iterative process the students learn when taking their first steps to learning to program. The previous essay format would not allow this approach to solving programming problems. As stated by Pane and Myers (Davies as cited in 1996), the end product is rarely a result of the order in which it was created. Although this could not be measured in terms of the results of this study, the feature was still expressed as a welcome addition to the exam format by the students in the test group.

Furthermore, although an assumption was made that the context of learning programming (using a computer) and method of assessment (using pen-and-paper) would affect memory, this potential difference could not be measured using the results. This could indicate that programming ability and the material learnt is closer to meaningful material as it was mentioned by Lieberman (2011), which is not effected by environmental context to the same extent as presumed.

Ensuring fair grading of the students' contributions by applying measures to ensure anonymity of each paper, and by using the same grading template for all test results the researcher's influence were minimised as much as possible, adding credibility to the findings of this study. However, for future work, in order to draw any definite conclusions, and to dismiss or prove the hypothesis of this paper, the experiment would need to be repeated, preferably using a larger, if not much larger sample. Although 20 students partook in the experiment of this study, perhaps double of that amount would be an adequate number to aim for. Furthermore, perhaps a better choice of strata would be to divide students by grade rather than self-reported programming experience.

Moving forward, it is still important to continue to evaluate the assessment method used in programming courses as the problem of low attaining rates persists, in order to understand whether the exam mode is partly to fault for the high rate of students failing these types of courses.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

Anderson, E. (2015). Here are the workers most in demand in the uk. *Telegraph Media Group Limited*. Retrieved September 19, 2016 from http://www.telegraph.co.uk/finance/jobs/11602670/Here-are-the-workers-most-in-demand-in-the-UK.html.

Arellano, N. E. (2015). Canada needs 182,000 people to fill these IT positions by 2019. *IT World Canada*. Retrieved September 19, 2016 from http://www.itworldcanada.com/article/canada-needs-182000-people-to-fill-these-it-positions-by-2019/287535.

Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. *Psychology of Learning and Motivation, 2,* 89–195.

Atwood, J. (2007). Why can't programmers.. program? Retrieved September 19, 2016 from: https://blog.codinghorror.com/why-cant-programmers-program/.

Begosso, L. C., Begosso, L. R., Gonçalves, E. M., & Gonçalves, J. R. (2012). An approach for teaching algorithms and computer programming using greenfoot and python. In *Frontiers in education conference (FIE), 2012* (pp. 1–6). IEEE.

Bennedsen, J., & Caspersen, M. E. (2007). Assessing process and product: A practical lab exam for an introductory programming course. *Innovation in Teaching and Learning in Information and Computer Sciences, 6*(4), 183–202.

Bergin, S. & Reilly, R. (2005). Programming: factors that influence success. In *ACM SIGCSE bulletin* (Vol. 37, pp. 411–415). ACM.

Biggs, J. (2003). Aligning teaching and assessing to course objectives. *Teaching and learning in higher education: New trends and innovations, 2,* 13–17.

Chamillard, A., & Braun, K. A. (2000). Evaluating programming ability in an introductory computer science course. *ACM SIGCSE Bulletin, 32*(1), 212–216.

Chamillard, A. & Joiner, J. K. (2001). Using lab practica to evaluate programming ability. In *ACM SIGCSE Bulletin* (Vol. 33, pp. 159–163). ACM.

Chodorow, K. (2013). *MongoDB: The definitive guide*. Newton: O'Reilly Media, Inc.

Clariana, R., & Wallace, P. (2002). Paper-based versus computer-based assessment: Key factors associated with the test mode effect. *British Journal of Educational Technology, 33*(5), 593–602.

Daly, C. & Waldron, J. (2004). Assessing the assessment of programming ability. In *ACM SIGCSE bulletin* (Vol 36, pp. 210–213). ACM.

Dillon, E., Anderson, M., & Brown, M. (2012). Comparing feature assistance between programming environments and their effect on novice programmers. *Journal of Computing Sciences in Colleges, 27*(5), 69–77.

English, J. (2002). Experience with a computer-assisted formal programming examination. In *ACM SIGCSE bulletin* (Vol. 34, pp. 51–54). ACM.

Enticknap, N. (1998). Computing's golden jubilee. In *Resurrection: The Bulletin of the Computer Conservation Society* (Vol. 1(20)).

Gross, P. & Powers, K. (2005). Evaluating assessments of novice programming environments. In *Proceedings of the first international workshop on computing education research* (pp. 99–110). ACM.

Haghighi, P. D., Sheard, J., Looi, C.-K., Jonassen, D., & Ikeda, M. (2005). Summative computer programming assessment using both paper and computer. In *ICCE*, (pp. 67–75).

Isarida, T., & Isarida, T. K. (2007). Environmental context effects of background color in free recall. *Memory & Cognition, 35*(7), 1620–1629.

Jacobson, N. (2000). Using on-computer exams to ensure beginning students' programming competency. *ACM SIGCSE Bulletin, 32*(4), 53–56.

Lappalainen, V., Lakanen, A.-J., & Högmander, H. (2016). Paper-based vs computer-based exams in cs1. In *Proceedings of the 16th Koli calling international conference on computing education research* (pp. 172–173). ACM.

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., et al. (2016). Brief history of the internet. Retrieved September 19, 2016 from http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet#Origins.

Lieberman, D. A. (2011). *Human learning and memory*. Cambridge: Cambridge University Press.

Pane, J. & Myers, B. (1996). Usability issues in the design of novice programming systems. Technical report, Carnegie Mellon University - Institute for Software Research.

Rajala, T., Kaila, E., Lindén, R., Kurvinen, E., Lokkila, E., Laakso, M.-J., et al. (2016). Automatically assessed electronic exams in programming courses. In *Proceedings of the Australasian computer science week multiconference* (p. 11). ACM.

Sarkar, A. (2015). The impact of syntax colouring on program comprehension. In *Proceedings of the 26th annual conference of the psychology of programming interest group* (*ppig 2015*) (pp. 49–58).

Sarpong, K. A.-M., Arthur, J. K., & Amoako, P. Y. O. (2013). Causes of failure of students in computer programming courses: The teacher-learner perspective. *International Journal of Computer Applications, 77*(12), 27.

Sheard, J., Carbone, A., D'Souza, D., Hamilton, M. (2013). Assessment of programming: pedagogical foundations of exams. In *Proceedings of the 18th ACM conference on innovation and technology in computer science education* (pp. 141–146). ACM.

Simpson, I. (2016). Developers in demand — tech talent doomsday. *Clearcode*. Retrieved September 19, 2016 from: http://clearcode.cc/2016/01/developers-in-demand-tech-talent-doomsday/.

Smith, S. M., & Vela, E. (2001). Environmental context-dependent memory: A review and meta-analysis. *Psychonomic Bulletin & Review, 8*(2), 203–220.

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom's taxonomy for cs assessment. In *Proceedings of the tenth conference on Australasian computing education* (Vol. 78, pp. 155–161). Australian Computer Society, Inc.

U.S. Bureau of Labor Statistics. (2013). Occupational employment projections to 2022. Retrieved September 19, 2016 from http://www.bls.gov/opub/mlr/2013/article/occupational-employment-projections-to-2022.htm.

van Heur, R. (2016). Fears of software skills shortage in Germany and the Netherlands. *Computer Weekly, TechTarget*. Retrieved September 19, 2016 from http://www.computerweekly.com/news/4500269840/Fears-of-software-skills-shortage-in-Germany-and-the-Netherlands.

Winslow, L. E. (1996). Programming pedagogy—A psychological overview. *ACM Sigcse Bulletin, 28*(3), 17–22.

Martina Öqvist has a master in Computer and System Sciences from the department of Computer and Systems Sciences, Stockholm University.

Jalal Nouri is an associate professor at the department of Computer and Systems Sciences, Stockholm University, and the coordinator of the Technology-enhanced learning group at the IDEAL unit.