Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Kjerand Evje

Automatic question generation for JavaScript programming courses

Master's thesis in Computer Science
Supervisor: Guttorm Sindre
June 2023

NTNU
Norwegian University of
Science and Technology

Kjerand Evje

Automatic question generation for JavaScript programming courses

Master's thesis in Computer Science
Supervisor: Guttorm Sindre
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

With the education system becoming increasingly digital and automated, the process of creating and grading assignments or practice questions has largely remained the same. Teachers are required to use valuable time and resources creating and grading assignments, exercises, quizzes and other forms of evaluations. This increasingly becomes an issue as the number of students in a course becomes larger, which further increases the workload. In order to reach their learning outcome goals, students might also wish for a larger quantity and variation of practice questions than are made available in the course. The development of automatic question generation technology makes it possible to easily generate a large number of relevant questions, allowing teachers to save resources and for students to have access to a continuous supply of questions.

This thesis explores the development of a question generation tool made for generating programming questions specifically for JavaScript programming courses. The goal of developing this tool is to create an artifact that might be used at universities and to evaluate how effective question generation tools are at improving learning outcomes in programming courses. Through an experiment evaluating the tool, it was found that the participants achieved a significant improvement in learning outcomes after using the tool for a short session. The results also showed that the question generation tool was just as effective in improving learning outcomes for both beginner and intermediate programmers.

Sammendrag

Ettersom utdanningssystemet har blitt stadig mer digitalisert og automatisert, har prosessen med å lage og vurdere oppgaver stort sett forblitt den samme. Lærere bruker verdifull tid og ressurser på å lage og vurdere oppgaver, quizer, innleveringer og andre former for evalueringer. Dette blir spesielt et problem etterhvert som antall studenter i et emne øker, som ytterligere øker arbeidsmengden for lærerne. For å oppnå sine læringsutbytte-mål kan studenter også ha ønske om å ha tilgang til en større mengde og variasjon av oppgaver enn det som er tilgjengelig. Utviklingen av teknologi for automatisk generering av variantoppgaver gjør det mulig å enkelt generere et stort antall relevante oppgaver, som tillater lærere å spare ressurser og for studenter å få kontinuerlig tilgang til nye oppgaver.

Dette prosjektet beskriver utviklingen av et verktøy for generering av variantoppgaver laget for å generere programmeringsoppgaver spesielt rettet mot JavaScript emner. Målet med å utvikle dette verktøyet er å lage et produkt som kan bli brukt ved universiteter og for å evaluere hvor effektivt et slikt verktøy er for å forbedre læringsutbytter i programmeringskurs. Gjennom et eksperiment for å evaluere verktøyet ble det funnet at deltakerne oppnådde en betydelig forbedring i læringsutbytter etter å ha brukt verktøyet en kort økt. Resultatene viste også at verktøyet var like effektivt for å forbedre læringsutbytter for både nybegynnere og videregående programmerere.

Acknowledgements

Thank you to Prof. Guttorm Sindre from the Department of Computer Science at NTNU for being my supervisor for this project. His insights and feedback relating to the development of the prototype, execution of the experiment and writing of the thesis have been extremely valuable, and have played an important role in how the project turned out. The weekly project meetings were really helpful in making sure I stayed on the right track throughout the project.

Additionally, I would like to thank all the students who took time out of their schedules to participate in the experiment and helping evaluating the project prototype. I would also like to thank Associate Prof. Ole Christian Eidheim for helping recruit students to participate in the experiment and for giving helpful feedback.

Lastly, I would like to thank my parents and my classmates for their support and help to make the process of carrying out this project easier and more enjoyable.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

At universities, programming is taught to students through a variety of methods. The students increase their programming skills by solving assignments, answering quizzes, working on group projects, reading the course book and attending lectures [1]. A fundamental part of learning a new skill is repeated practice and repetition over a long period of time, which is also true for learning programming [2], [3]. In programming courses, this is usually achieved by the lecturer manually creating programming assignments that are given to and solved by students throughout the course. The assignments are then usually manually graded by either the lecturer or the teaching assistants in the course. While this achieves the desired goal of creating assignments that the students can use to increase their competence, the process of manually creating and grading assignments can be time-consuming and prone to errors.

This has led to the development of question generation technology [4], which has the aim to automatically generate a large number of question variants that can be used as part of the teaching process. Question generation technology has the potential to be helpful to educators in many ways, including automatically generating assignments and assessments, reducing expenses due to saving time on creating and grading questions, and ensuring students a continuous supply of new questions. Another use-case for question generation technology is to use it for mitigating cheating at exams or assessments. Question generation can be used to generate unique exam questions which can make student cooperation during home exams more difficult. This has become increasingly relevant due to the increased number of cheating cases during the COVID-19 pandemic [5].

While previous question generation research has mainly focused on generating short answer questions and quizzes, this project is focused on the generation of more complex programming questions, specifically aimed at JavaScript programming courses taught at universities. While there are multiple different uses for question generation technology, the main focus of this project is the use of question generation as a tool for self-study and improving learning outcomes. The project is split into two main parts. The first is the process and methodology used to develop the new question generation tool, including an overview of previous research. The paper will also give an overview of all the functionality present in the final prototype. The second part of the project is an experiment conducted using the final question generation prototype, with the goal of evaluating the effectiveness of the prototype. The experiment was conducted on a group of university students with varying programming experience in JavaScript, with the goal of evaluating whether the question generation was effective in improving their learning outcomes in a chosen JavaScript topic. The aim of this project is to answer the following research questions:

RQ1. How effective is question generation in improving learning outcomes for JavaScript programming courses?

RQ2. Is question generation more effective in improving learning outcomes for beginner programmers than intermediate?

The first research question is aimed at finding out whether the question generation tool is actually an effective tool for learning. Learning is measured using learning outcomes, and the effect is limited to JavaScript programming courses. This limitation was added because finding that question generation is effective in increasing learning outcomes for programming in JavaScript does not necessarily carry over to other subjects or fields. The second research question builds on the previous question and is aimed at finding out if there is any difference in learning effectiveness between beginner and intermediate programmers. This can be used to determine the potential use cases of question generation, whether it is best suited for introductory programming courses or if it is better suited for more advanced courses.

The project report is split into six different chapters. Chapter 1, the current chapter, contains a short overview of the project. Next, Chapter 2 consists of the background for the project, including relevant research and a summary of the specialization project leading up to this project. Chapter 3 describes the methodology used to develop the question generation tool prototype and the methodology used in the experiment. Afterward, Chapter 4 contains the results from the project, both a presentation of the final prototype with a description of all the functionality and the results from the experiment. Chapter 5 is a discussion of the results, including discussing limitations of the tool and the experiment, potential use cases and interpreting the results from the experiment. Chapter 6 contains the conclusion of the project, summarizing the report and answering the research questions, and it contains further work, both with regard to the development of the tool and with regard to further experiments.

Chapter 2

Background

This chapter contains an overview of the background of the project and is split into two sections. The first section is a description of the specialization project that this project is a continuation of and the second section contains previous research relating to the topic of question generation.

2.1 Specialization project

The question generation tool developed for this project was a continuation of a prototype that was started in a specialization project the previous semester. The specialization project laid the groundwork for much of the functionality and research behind the question generation tool for this current project.

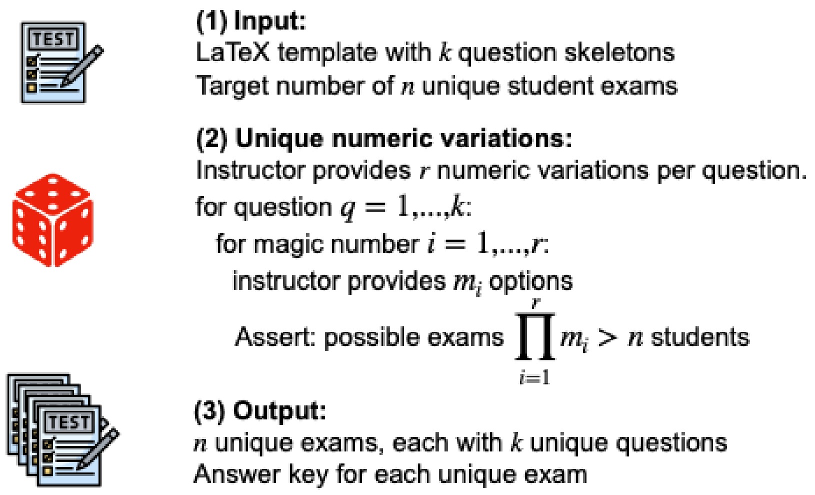
The specialization project consisted of a literature review of the current research on question generation methods, estimating the difficulty of questions and measuring question effectiveness. An initial prototype of the question generation tool was also developed, in addition to the literature search. The prototype consisted of some basic functionality with regard to being able to generate questions. It allowed for basic variable substitution to generate questions with unique numeric and text variables, which made it possible to some degree to generate questions. The prototype had no functionality for adding code to questions, for the user to write their own code solution to problems and there was no user interface for students to actually solve the generated questions.

The focus of the specialization project was broader than this project, in that it focused on question generation as a tool for self-study, in addition to how question generation can be used to reduce cheating at home exams. After completing the specialization project and having discussions with multiple educators teaching relevant programming courses, it became clear that using question generation as a self-study educational tool was more relevant than using it for exam generation. This was in part because of the shift from home exams during COVID-19 back to regular on-site exams, which largely eliminated the need for generating unique home exams. A decision was therefore made for this current project to narrow the focus on the educational self-study purposes of question generation instead of the parts relating to generating fair exam questions.

2.2 Previous research

This section is focused on the previous research that formed the base for large parts of the project. The main topics covered are different methods used for question generation, automated assessment of questions, evaluating the effect of question generation on learning outcomes and estimating question difficulty.

2.2.1 Question generation using templates and variables



(1) Input:
LaTeX template with k question skeletons
Target number of n unique student exams

(2) Unique numeric variations:
Instructor provides r numeric variations per question.

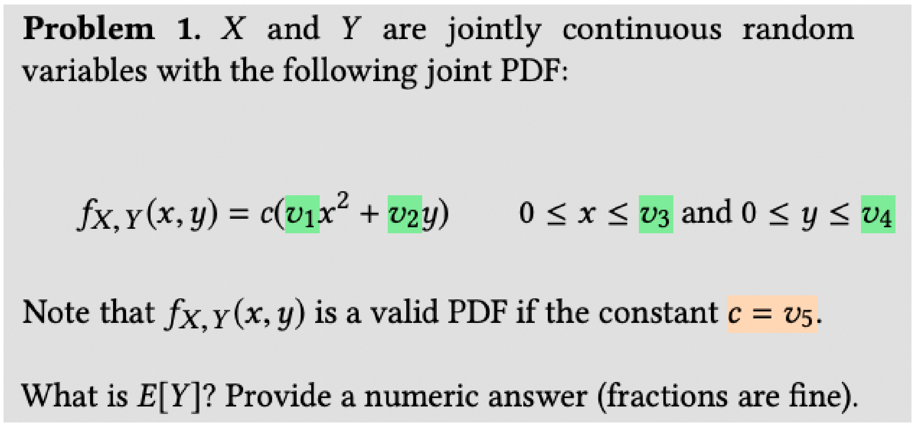
```
for question $q = 1,...,k$:
  for magic number $i = 1,...,r$:
    instructor provides $m_i$ options
    Assert: possible exams $\prod_{i=1}^{r} m_i > n$ students
```

(3) Output:
n unique exams, each with k unique questions
Answer key for each unique exam

Figure 2.1: Exam generation pipeline, copied from Rusak et al. [6, p. 2].

A major challenge faced by educators during the COVID-19 pandemic was the rapid change from physical lectures and assessments to digital ones. A shift to digital home exams entails some problems, one of them being that the exams are unsupervised and therefore more susceptible to cheating. This was demonstrated by a large increase in cheating cases at exams [5]. There are different strategies that can be used to mitigate cheating for home-exams [7], one of which is changing the exam design to make student cooperation more difficult. Other strategies includes using plagiarism checks, analyzing delivery times or using video conferencing. Changing exam designs could be done by having more essay-type questions instead of short-form, changing the order of questions or by giving unique questions. The unique exam framework is a framework developed by Rusak et al. [6] created to remedy this problem. The goal of this framework is to change the exam design thorough producing different exam questions for each student, and thereby mitigate cheating.

The framework is based around a pipeline depicted in Figure 2.1. The questions are designed using a LaTeX template. The pipeline is divided into three separate stages. The first stage consists of creating a question skeleton, which is manually done by the teacher. A question skeleton is a template for an exam question, with the question consisting of plain text that is constant throughout all the questions, in addition to variables that will vary between questions. An example of a question skeleton is provided in Figure 2.2. In this example, there is some plain text with the values v_1, v_2, v_3, v_4 and v_5 being variables. This process is repeated k times to produce the desired number of unique question skeletons. The next stage in the pipeline is for the teacher to define the different possible values for each variable in the question skeleton. These variables can either be independent or dependent, as seen in Figure 2.3.


Problem 1. X and Y are jointly continuous random variables with the following joint PDF:

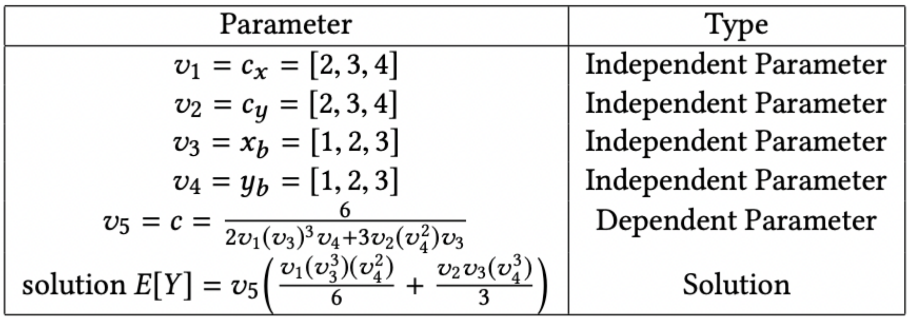
$$f_{X,Y}(x,y) = c(v_1 x^2 + v_2 y) \qquad 0 \leq x \leq v_3 \text{ and } 0 \leq y \leq v_4$$

Note that $f_{X,Y}(x,y)$ is a valid PDF if the constant $c = v_5$.

What is $E[Y]$? Provide a numeric answer (fractions are fine).

Figure 2.2: Example problem, copied from Rusak et al. [6, p. 3].

The variables v_1, v_2, v_3 and v_4 have a list of possible values which are completely independent, while the variable v_5 is calculated based on the previous variables. After the possible variable values have been defined, the teacher needs to define the question solution. The solution needs to be defined as a function that for a set of variables provides the correct question solution for the given combination of variables. This process needs to be completed for each unique question skeleton. After the solution is defined, the last stage of the pipeline is to generate the question variants. This is accomplished by randomly choosing a combination of the valid variable options and calculating the solution for the question. It is then possible to create n unique exams, each consisting of k unique question variants.



Parameter	Type
$v_1 = c_x = [2, 3, 4]$	Independent Parameter
$v_2 = c_y = [2, 3, 4]$	Independent Parameter
$v_3 = x_b = [1, 2, 3]$	Independent Parameter
$v_4 = y_b = [1, 2, 3]$	Independent Parameter
$v_5 = c = \frac{6}{2v_1(v_3)^3 v_4 + 3v_2(v_4^2)v_3}$	Dependent Parameter
solution $E[Y] = v_5\left(\frac{v_1(v_3^3)(v_4^2)}{6} + \frac{v_2 v_3(v_4^3)}{3}\right)$	Solution

Figure 2.3: Example solution, copied from Rusak et al. [6, p. 3].

While this framework is focused on automatic question generation for exams, the same framework could be utilized when the goal is to increase student learning outcomes. Instead of generating questions with the goal of creating unique exams, a similar pipeline can be used to generate a large quantity of quite similar, but still unique questions. This can be useful for students to get repeated practice on a given topic by solving a large number of relevant questions. This is also useful to ensure that students can get a continuous supply of new questions and not be limited by only having access to the manually created course questions.

A substantial difference when generating questions for formative assessments instead of summative assessments [8] is the need for questions of similar difficulty. When generating questions for a summative assessment it is very important that students get equally challenging questions to ensure that the students are all evaluated fairly. This can be a limitation when generating questions in that it reduces the variance possibilities, because of the need for equally difficult questions. As the variety between questions increases, the ability to judge the difficulty accurately increases. On the other hand, when generating questions for formative assessments, there are no such limitations on difficulty variance between questions. Creating question variants with large variance in difficulty may actually be a benefit, in that it allows the students to practice both fundamental and more advanced questions related to a topic. Because of this, the potential for question variation for formative assessments is higher.

2.2.2 Semantic-based question generation

While the question generation method described by Rusak et al. is focused on an educator providing a skeleton template for questions, there are also other strategies that can be used. Yao et al. [9] describe an implementation of semantic-based question generation. This process works by mapping natural language sentences into a representation of the meaning behind the sentence using Minimal Recursion Semantics (MRS). Using the sentence meaning, the system can generate relevant questions.

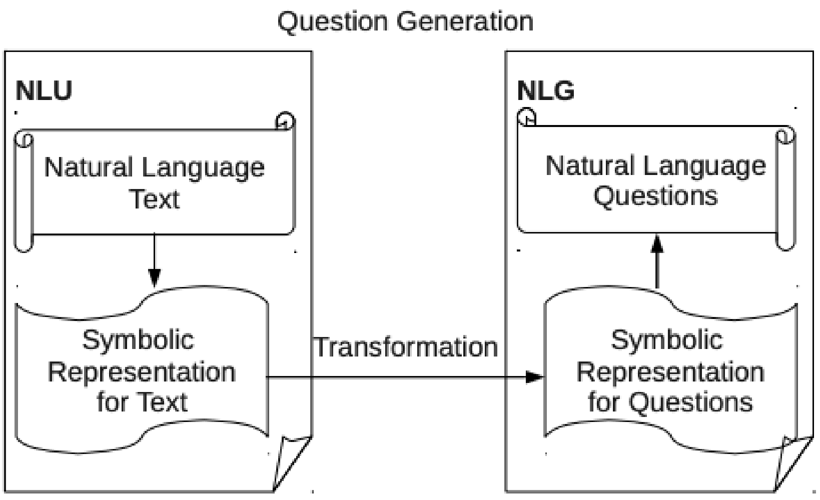

Question Generation
NLU
Natural Language Text
Symbolic Representation for Text
Transformation
NLG
Natural Language Questions
Symbolic Representation for Questions

Figure 2.4: Question generation model, copied from Yao et al. [9, p. 12]

.

A model of the process of generating semantic-based questions is shown in Figure 2.4. There are two pain parts of the system, focused on Natural Language Understanding (NLU) and Natural Language Generation (NLG). The part concerning NLU is concerned with understanding and processing the meaning behind the text, and the NLG part is concerned with generating questions based on the understanding. The process starts by defining some natural language text that the generated questions will be focused on. The natural language text will then be converted into a symbolic representation that tries to capture the meaning of the text using MRS. A transformation function is then applied to the symbolic representation of the initial text, converting it to a symbolic representation of a question. This representation can then be converted back into natural language, now as a question.

The question generation implementation by Yao et al. is quite different from the question skeleton method utilized by Rusak et al., but there are some similarities in the general process. Both implementations generate a symbolic representation of the given question, Yao et al. does this by using MRS while Rusak et al. does this by creating a skeleton with relevant variables representing the question. In both methods some transformation is then applied to the representation, converting it into a question. A limitation of semantic-based question generation is that it is mainly focused on processing natural language text and is therefore not suited to generating complex number-based questions, which are often used in math and programming courses. Rather it works best with generating understanding-based questions for a text.

2.2.3 Question generation for formative assessments

Using automatic question generation as an educational tool can have many benefits. It can reduce the amount of time teachers need to spend on creating questions, save resources and make sure that students have a large number of questions to solve. A relevant issue when creating a question generation tool is whether the tool is actually effective as a learning tool. Tsai et al. [10] attempts to evaluate the effectiveness of their question generation tool by evaluating the learning outcomes of students using the tool in a Python programming course.

The question generation model used is based on semantic-based generation, similar to Yao et al. in combination with syntax-based generation. Syntax-based generation differs from semantic-based generation in that it uses a syntactic tree to represent and transform the natural language text, instead of using a meaning-based symbolic representation. The quality of the generated questions is then evaluated using the Bilingual Evaluation Understudy (BLEU). This metric is based on the accuracy of the generated question, with accuracy being measured as the similarity between the question and the reference sentence, using N-gram matching rules.

$$BLEU = BP \cdot exp(\sum_{n=1}^{N} w_n \cdot log(P_n))$$

The goal of the study was to find out if engagement with the question generation tool correlated with improved learning outcomes across a range of learning outcomes. The tool was evaluated by calculating an engagement score $S_{RT}(i)$ for each student S_i. This score is a measure of how much each student utilized the tool. The score is calculated by dividing the number of correctly answered questions c_k^i by the total number of questions N_k for each learning outcome k, then summing this value for each learning outcome.

$$S_{RT}(i) = \sum_{k=1}^{n} \frac{c_k^i}{N_k}$$

This engagement score gives an indicator of how many questions each student answered and the ratio of correct answers. Four measures were then collected to measure the learning performance of each student. These measures were: students answering a short questionnaire for the learning outcomes, solving coding problems, their exam score and their final semester score. Comparing the learning performance scores with the engagement scores, it was found that a higher engagement score correlated positively with all four measures of learning performance. This finding can therefore give an indication that automatic question generation can be an effective tool in improving learning outcomes.

A potential limitation of this study is that even though it finds a correlation between engagement score and learning performance, this does not mean causation. It might be the case that high-performing students were more likely to utilize the tool than underperforming students, and they would have scored higher on learning outcomes performance anyway. It would therefore be relevant to do a controlled experiment where the learning outcomes are directly linked to the usage of the question generation tool.

2.2.4 Assessing question difficulty

When utilizing a question generation tool it might be useful to have an estimate of the difficulty of each question compared with other questions. This can be beneficial in many situations. If the tool is used to generate questions for an assessment it is paramount that the questions given to the students are of similar or equal difficulty to ensure that all get a fair assessment. It can also be useful to know the question difficulty when using a question generation tool as an educational tool, as it can allow students to more easily find questions that match their current abilities for the given topic.

In 2013 a research group proposed a method for estimating question difficulty using a self-study tool [11]. The researchers made available a studying tool that students could voluntarily use as part of a course to practice certain topics. The tool consisted of a pool of Multiple Choise Questions (MCQ) that the students could answer. By collecting data on how many students answered each question right or wrong, they could estimate a difficulty rating for each question.

$$w(x) = \frac{\sum_{i=0}^{n} c(i,x)}{n(x)}$$

The difficulty rating $w(x)$ for each question x is defined as the sum of all the times a question was answered correctly divided by the total number of answers for that question. $c(i,x)$ in the formula returns the value 0 if the answer is wrong and 1 if it is correct. This rating will therefore give an estimate of difficulty based on the ratio of correct and wrong answers to a given question. The rating is a value between 0 and 1, where 0 means all students got the question right and 1 means all students got the question right. This rating can then be used to group together questions of similar scores.

A limitation of this method is that requires quite a lot of data before an accurate estimate can be extracted. After new questions have been generated, they need to be answered by a large number of users before the questions will have an accurate score.

2.2.5 Automated assessment

An important aspect to consider when creating a question tool for self-studying is how the students get feedback on their work. Feedback is necessary for the students to know whether their work is correct or not. Without getting good feedback, students can risk learning things the wrong way or misunderstanding the material. Usually in university programming courses, students get feedback on their assignments from teachers or student assistants manually grading their work. This is not feasible when utilizing question generation tools, as the number of questions can be very large and one of the major advantages of using such tools is for the teachers to save time. These factors make it necessary to have some sort of automated assessment method to accompany the question generation tool.

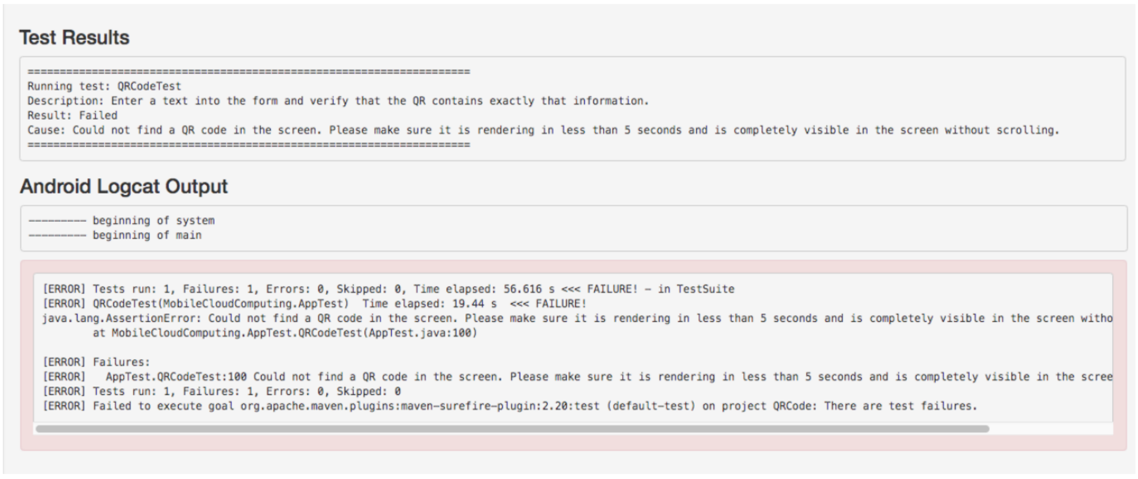
In 2020 Bruzual et al. [12] developed a system for automated assessment of exercises in a mobile application development course. The goal of creating the system was to make mobile application coding exercises scale more easily to large classrooms, as previously the exercises needed to be manually graded by teaching assistants.

The system works by the teacher or teaching assistant designing a test case for each exercise. An example test case is provided in Figure 2.5. The test cases work by creating a set of actions that will be performed on the mobile application, then checking if the application produces the correct output in response. The students

```java
public class AppTest {

  AndroidDriver driver;

  public void HelloUserTest(){
    String currId = null;
    try {
      // Input random value into EditText
      Random rand = new Random();
      int n = rand.nextInt(25000) + 1;
      currId = "txtInput";
      driver.findElement(By.id(currId)).sendKeys(""+n);
      // Press button
      TouchAction t = new TouchAction(driver);
      currId = "btnSubmit";
      t.tap(driver.findElement(By.id(currId))).perform();
      // Check if input text is appended to Hello
      currId = "txtResult";
      Assert.assertEquals(driver.findElement(By.id(currId)).
        ↪ getText(), "Hello "+n, "The TextView contains the
        ↪  wrong text.");
    } catch (NoSuchElementException e) {
      Assert.fail("Could not find element with ID '"+currId+
        ↪ "'. Please check the assignment instruction and
        ↪ use the provided IDs.");
    } catch (Exception e) {
      Assert.fail(e.getMessage());
    }
  }

}
```

Figure 2.5: Test case, copied from Bruzual et al. [12, p. 43].

will produce their mobile application solution, then upload it to the system. The system will then run the test cases and assess whether the submission passes all the cases. If some test cases do not pass or there is an error when compiling the code, the students will get feedback in the system user interface. This is demonstrated in Figure 2.6. The students are then able to make changes to their code based on the feedback, then resubmit their solution.



Test Results

```
======================================================================
Running test: QRCodeTest
Description: Enter a text into the form and verify that the QR contains exactly that information.
Result: Failed
Cause: Could not find a QR code in the screen. Please make sure it is rendering in less than 5 seconds and is completely visible in the screen without scrolling.
======================================================================
```

Android Logcat Output

```
--------- beginning of system
--------- beginning of main
```

```
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 56.616 s <<< FAILURE! - in TestSuite
[ERROR] QRCodeTest(MobileCloudComputing.AppTest)  Time elapsed: 19.44 s  <<< FAILURE!
java.lang.AssertionError: Could not find a QR code in the screen. Please make sure it is rendering in less than 5 seconds and is completely visible in the screen witho
        at MobileCloudComputing.AppTest.QRCodeTest(AppTest.java:100)

[ERROR] Failures:
[ERROR]   AppTest.QRCodeTest:100 Could not find a QR code in the screen. Please make sure it is rendering in less than 5 seconds and is completely visible in the scree
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.20:test (default-test) on project QRCode: There are test failures.
```

Figure 2.6: Feedback output, copied from Bruzual et al. [12, p. 43].

A potential limitation of this system is that the test cases are only able to look at the application output produced by the students. This means that it can evaluate the end result, but not the process of how the result was obtained. This opens up the possibility for students to take shortcuts in their solutions by producing the correct output for the assignment, but not using the correct approach. This is not a problem when assignments are manually graded, as the teacher or teaching assistants can evaluate both the output and the process of producing the output. While this might be a problem when automatically assessing exercises that count as part of the grade in a course, it is not necessarily a problem if the system is used as a tool for self-study. If the goal for the students is to learn as much as possible, there is no motivation for students to take shortcuts to produce the correct output.

Chapter 3

Method

This chapter describes the methodology used in this project, consisting of two main sections. The first section is focused on the methodology used to develop the question generation tool prototype and the second section is focused on the methodology used in the experiment. The methodologies for this project were chosen based on the research questions ”How effective is question generation in improving learning outcomes for JavaScript programming courses?” and ”Is question generation more effective in improving learning outcomes for beginner programmers than intermediate?”. There have been multiple previous implementations of question generation tools, but none were found that are specifically focused on more complex programming questions that could answer these research questions. The research questions are based on the evaluation of an artifact, being a question generation tool. This made using design science as a research paradigm a natural choice. The research questions also require an evaluation of learning outcomes to be able to determine the effectiveness of the artifact. A pretest-posttest design experiment was chosen as this would make it possible to evaluate the learning outcomes of the participants before and after exposure to the artifact, thereby directly seeing if the artifact was effective or not.

3.1 Question generation tool

This section will describe the methodology used to develop the tool, including the chosen research paradigm, the technologies used and the development process.

3.1.1 Design science

The research paradigm chosen for developing the question generation tool was design science, due to the main focus of the research being the development of a new artifact. This is one of the two main research paradigms in information system research, with the other being behavioral science. While behavioral science focuses on verifying and predicting human and organizational behavior, the goal of design science is to extend human and organizational capabilities by creating new and innovative artifacts. Hevner et al. [13] defines seven guidelines for performing design science, as seen in Figure 3.1.

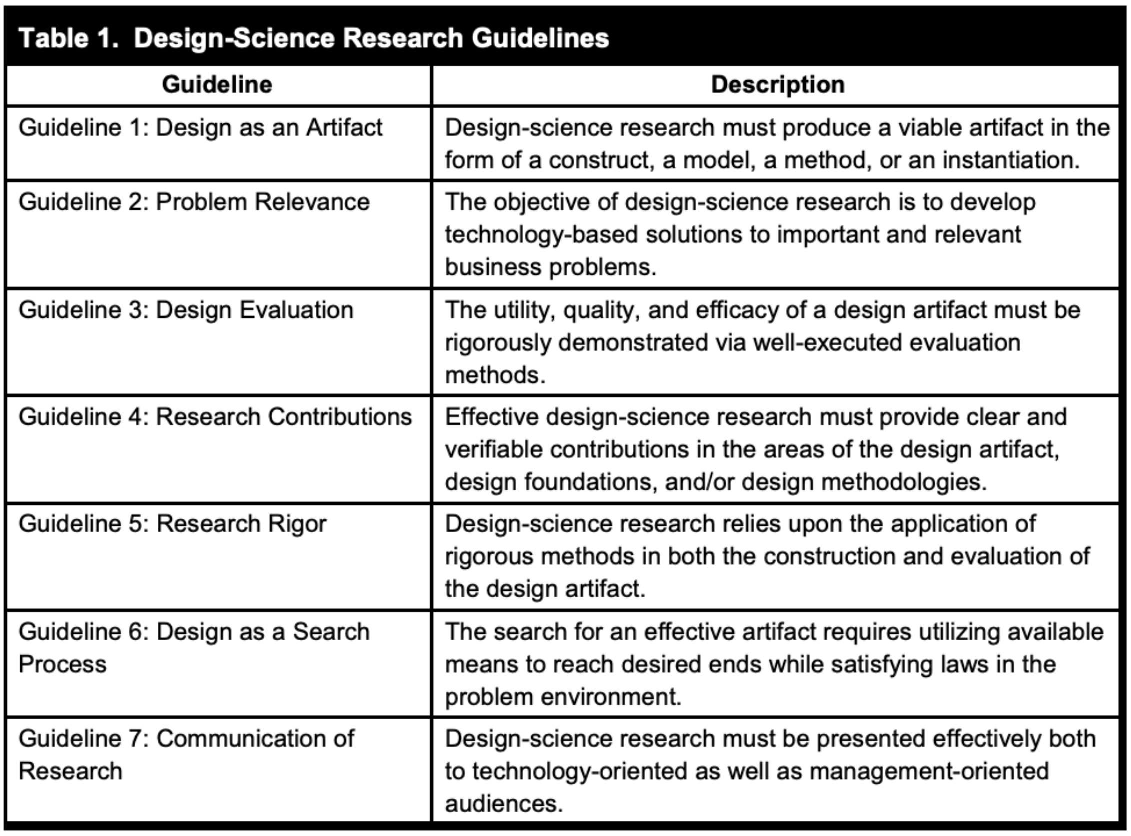


Table 1. Design-Science Research Guidelines

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Figure 3.1: Design science guidelines, copied from Hevner et al. [13, p. 83].

Design as an Artifact

An integral part of the design science methodology is to produce an actual artifact that can be utilized in a specific domain. This artifact can come in different forms, whether it is the design of a new model, method, construct or instantiation. Artifacts developed through the design science process are very rarely fully completed systems that can be used in practice. It is most often an innovation that defines an idea or a new product that can later be efficiently implemented into a full information system.

The artifact developed for this project is the question generation tool method and instantiation. The artifact represents a new method for question generation, combining aspects from different tools described in the literature. The artifact instantiation is represented by the deployed website, in which the new question generation method can be utilized in practice. While the instantiation of the question generation tool is an initial implementation, it is close to being ready for use in a classroom setting. This was demonstrated in the experiment, in which the instantiation was used in practice by students. The goal of the instantiation is to demonstrate that the tool is feasible and that it achieves the object that it was created for.

Problem Relevance

The ultimate goal of using design science to produce an artifact is to solve a relevant business or organizational problem. The Technology Acceptance Model is a method often used in design science to evaluate the problem relevance of an artifact. This model can be used to evaluate to what degree an artifact is relevant for a given group by evaluating the perceived usefulness, the perceived ease of use, and the user acceptance of an artifact.

The problem relevance for the artifact produced in this project was ensured in a couple of ways. Firstly, extensive feedback on the artifact was received from university teachers in both this project and the specialization project. This is important because teachers are the main group for utilizing the tool to generate questions, and they have the relevant classroom experience to know what problems are important and which are not. Feedback was also received from students. Students are the main target group for utilizing the tool to answer the generated questions, so their experience is also important for the problem relevance. Secondly, the problem relevance was ensured by evaluating the artifact using the Technology Acceptance Model and thereby ensuring that the artifact actually is useful at solving the problem.

Design Evaluation

A crucial part of the design science process is evaluating the artifact to make sure it satisfies the criteria for quality, utility and effiacy. The environment the artifact will be used in defines what metrics are relevant when evaluating the artifact. Some qualities that artifacts might be evaluated for could be completeness, functionality, usability, reliability, performance, etc. There are five different methodologies typically used to evaluate artifacts, as seen in Figure 3.2. These are observational, analytical, experiment, testing and descriptive methodologies. Which methodology to use will depend on the art of the artifact and which evaluation metrics are relevant. As design science is an iterative process, evaluation is important in that it gives feedback relating to what parts of the system should be further developed and which should be discarded.

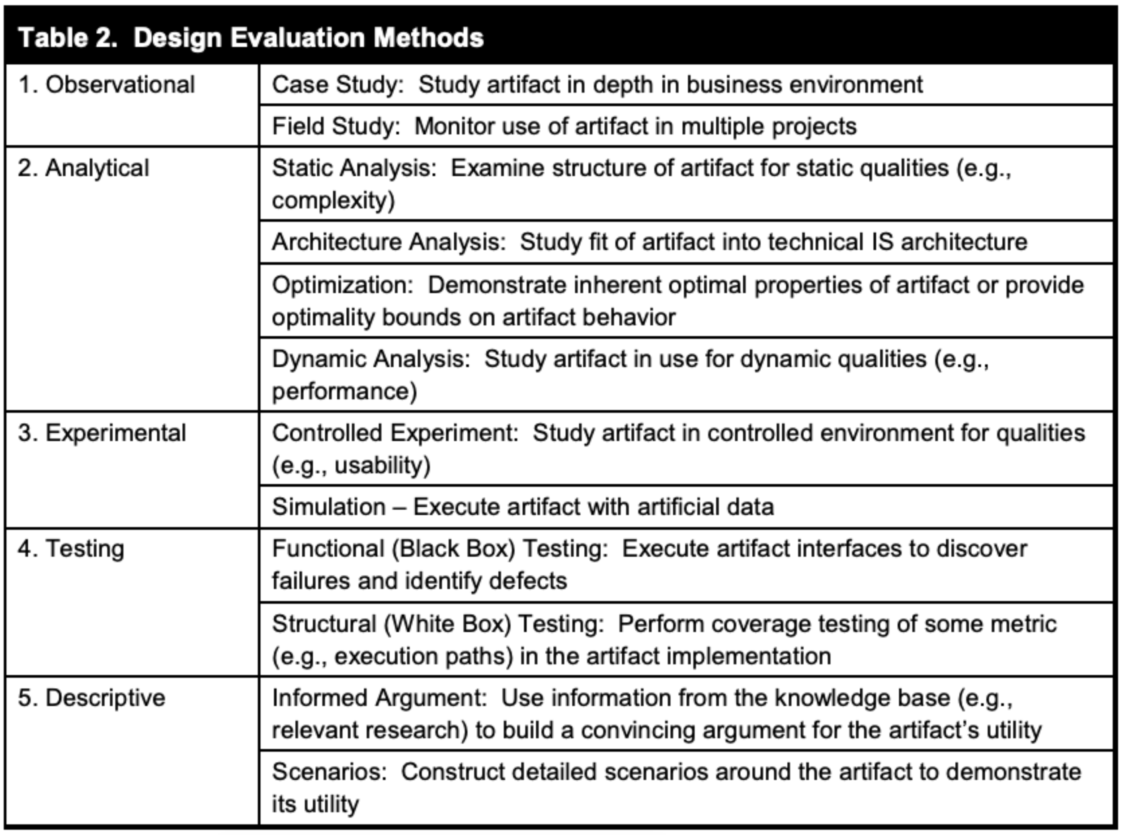


Table 2. Design Evaluation Methods

1. Observational	Case Study: Study artifact in depth in business environment
	Field Study: Monitor use of artifact in multiple projects
2. Analytical	Static Analysis: Examine structure of artifact for static qualities (e.g., complexity)
	Architecture Analysis: Study fit of artifact into technical IS architecture
	Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
	Dynamic Analysis: Study artifact in use for dynamic qualities (e.g., performance)
3. Experimental	Controlled Experiment: Study artifact in controlled environment for qualities (e.g., usability)
	Simulation – Execute artifact with artificial data
4. Testing	Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects
	Structural (White Box) Testing: Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Descriptive	Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility
	Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

Figure 3.2: Design science evaluation methods, copied from Hevner et al. [13, p. 86].

The evaluation methodology used to evaluate the artifact in this project is the experiment methodology, specifically a controlled experiment. This method was chosen, as it is the most relevant method to how the artifact would be used in a real-world setting. A controlled experiment makes it possible to evaluate the question generation tool on a variety of relevant evaluation metrics. Usability is one of these, as the participants in the experiment need to learn how to use the artifact. Reliability, as the artifact needs to be able to handle many concurrent users without any errors. Usefulness, in that it produces the desired improvement in learning outcomes. These metrics are evaluated using the Technology Acceptance Model, collecting usage data from the tool and learning outcomes questionnaires. Another possibility for choice of methodology would be an observational methodology, but a controlled experiment was chosen as this allowed for being able to collect more detailed data relating to the usability and effiacy of the system.

Research Contributions

For design science to be effective it must contribute something new to the research. There are three main types of research contributions, the design artifact, foundations or methodologies. The design artifact itself is often the contribution in itself. The contribution is a new artifact that solves a previously unresolved problem. Foundations refer to a research contribution that involves creating new or improving current methods, models or instantiations in the field. Finally, methodologies refer to a research contribution that creatively develops and uses evaluation methods and metrics in a new manner.

The research contribution for this project can be classified into both the design artifact and the foundation category. The project contributes a new design artifact in the question generation tool itself, and it also provides new methods for question generation by combining previous methods in a new manner.

Research Rigor

When conducting research using design science it is important to do it rigorously. This includes both rigor when developing the artifact and when evaluating it. In design science rigor is derived from using the knowledge base effectively, meaning the theoretical foundations and methodologies. Performance metrics are usually used to make claims about an artifact, and it is therefore important to make sure that the measurements are appropriate to ensure rigor in the metrics.

The development of the artifact was conducted rigorously by carefully reviewing the results from the relevant research papers when implementing the functionality. The experiment was conducted rigorously by making sure that participants all had access to the same information, and that the questionnaires were designed consistently with the use of standardized questions, such as questions usually found when using the Technology Acceptance Model. The consistency of the technology acceptance model ensures that the performance metric of the artifact is appropriate.

Design as a Search Process

An important part of design science is to use an iterative search process to discover an effective solution. This means iteratively generating potential solutions, then testing these solutions against the relevant constraints or requirements of the project and then making changes based on the evaluation. This is a necessary process, as the design problem is only a starting point that can change or expand throughout the design process. This process makes it possible to rapidly make changes based on feedback to make the artifact more relevant, instead of implementing a hard-set design created at the start.

An iterative design search process was employed in this project by creating multiple different iterations of the tool and evaluating it by getting user feedback from different sources, including the project supervisor, teachers, and students. Many of the feedback suggestions were then implemented in the tool, before being evaluated again in the next iteration. This was an important process, as the initial problem and functionality ideas were greatly expanded throughout this process, which led to a much more relevant final artifact. This part of the project is expanded on in the development process section below.

Communication of Research

The design science research should be able to be presented to either management-oriented or technology-oriented audiences. Presenting the research for technology-oriented audiences should be such that it is possible to implement the artifact. For the management-oriented audience, the presentation should focus on the artifact, but also the knowledge required to apply the artifact for organizational gain.

The artifact for this project is presented in this report. The report includes enough detail about the artifact so that technology-oriented audiences should be able to implement the artifact themselves. The report should also contain enough information for management-oriented audiences to demonstrate the potential organizational benefits of the artifact.

3.1.2 Development process

As mentioned previously, the question generation tool in this project is a continuation of a tool that was started in the previous semester as part of a specialization project. This project produced an initial prototype with some basic functionality but with the majority of the functionality not yet implemented. This was mainly due to the specialization project being focused on reviewing the literature, with the goal of only producing an initial prototype. During the specialization project, multiple meetings with programming teachers at the university were held to find out what functionality they wished to see implemented in a question generation self-studying tool. This feedback laid the groundwork for much of the functionality implemented in the final prototype. A wish from the interviewed teachers was the ability to have programming questions where students could submit their coding solution, and then have the code compile and run in the cloud and output a solution. In combination with this, they also requested to have the coding submission automatically assessed in the cloud.

The development of the tool for the master project started by implementing the major functionality requested by the university teachers from the last semester. This included mainly the implementation of the code submitting and compiling, automatic assessment, and designing an interface for the system. Weekly meetings were held with the master supervisor, in which feedback on the implementation was received. The implementation of the functionality was an iterative process, in that parts of the functionality were implemented, then feedback was received, then changes were based on the feedback. Feedback was received either from the project supervisor, from the programming teachers, or from having a user test the system. A new meeting was arranged with one of the programming teachers from the specialization project, where feedback was received on the current functionality implemented, in addition to some more wishes for new functionality.

The list of functionality for the question generation tool changed throughout the semester as previous ideas were discarded and new and improved ideas were added. All the major functionality defined at the start of the project ended up being implemented in some form in the final product, in combination with multiple new additions. In the experiment evaluation, the participants had many good suggestions for functionality that could be implemented. As this was towards the end of the semester, there was not enough time to implement these suggestions.

3.1.3 Technologies

This section will describe the main technologies utilized to develop the question generation tool and the motivation for choosing the different technologies. This includes the programming languages used, frameworks and cloud technology.

TypeScript

TypeScript was the programming language used to write the question generation tool code. A decision was made early to make the question generation tool available as a website instead of an application, as this would make it more easily accessible for the user. The language was chosen because JavaScript/TypeScript is one of the most popular languages used for developing web applications, and is well suited for this purpose. This popularity also means that there is a large number of frameworks and libraries available that made the development of the tool easier. TypeScript was also a natural choice due to the main focus of the tool being JavaScript programming courses, and it would therefore make it easier for JavaScript educators to potentially further develop the tool.

JavaScript was also an option, but TypeScript was chosen as the ability to define types was really helpful when defining the generated question data. TypeScript also makes the code more robust, as it helps to avoid type-errors. In addition, it makes the code more readable, which is useful if other programmers are to further develop the project.

React

React was the chosen framework for developing the web-application. React is a web-framework for developing web user interfaces for JavaScript and TypeScript, and is based around creating interfaces using components. React is one of the most popular frameworks in the industry, and is utilized in many of the most used websites in the world. It is therefore a very robust language, with a very large range of available libraries.

A major reason for using React was the availability of libraries, as this allowed for saving a lot of time on developing components that are already available in React. Another reason is the simplicity of use, and how easy it is to create a simple working website. The main objective of this project is the functionality relating to question generation, and not necessarily the user interface of the application. Using lots of time on developing the user interface was therefore not prioritized. React also works very well in combination with the other technologies used in the project, including Azure, Judge0 and GitHub Pages.

Judge0

Judge0 is an open-source online code execution system [14] and it supports a wide range of programming languages, including JavaScript and TypeScript. The system allows users to compile and execute code in the cloud. It works by the user writing a piece of code, which can then be submitted via their API. When the code has been executed, the user receives the output of the code, whether it be a text or an error message.

Judge0 plays an important role in the functionality of question generation tool in multiple ways. It allows for the teacher to write code that defines the solution for generated questions and it is used to compile and run the code that submits answers to the questions. Originally the plan for this project was to develop an independent code execution system that could execute code in the cloud to achieve these same tasks. After discovering that Judge0 already implemented all the functionality that was necessary for the question generation tool, a decision was made to utilize Judge0 instead of creating a new framework with the exact same functionality. This made it possible to use more time on coding the parts relating to question generation.

Azure Cosmos DB

Cosmos DB is a globally distributed database service provided by the cloud computing platform Azure. It allows users to create and use databases in the cloud, supporting many different database models. Cosmos DB is used as the database solution for this project and is used to store the generated question data in addition to usage data.

Cosmos DB was chosen as the database solution for a couple of reasons. Firstly is that it supports NoSQL, which is convenient for storing objects. Secondly, that it integrates very well with React with the Cosmos DB React library. Thirdly is that it supports auto-scaling. This means that the database can expand and shrink depending on how much it is used. While this was not an issue during this project due to the relatively low activity, it would be useful if the tool were to be used in practice. While Azure was chosen for this project, the same functionality could also have been provided by Amazon Web Services or Google Cloud Services.

GitHub Pages

GitHub Pages is a tool developed by GitHub to deploy GitHub projects as websites. This tool was used in this project to deploy the question generation tool web application. This made the tool easily accessible without having to run the code locally, and it was used as the deployed site during the experiment. GitHub Pages was chosen because the project was already stored on GitHub, and it made it easy to quickly push new changes to the deployed version when developing.

3.2 Experiment

This section will describe the methodology used to conduct the controlled experiment that was carried out as part of this project. The goal of the experiment was to evaluate the effectiveness of the question generation tool across a range of evaluation metrics.

3.2.1 Pretest-posttest design

The experiment for this project was designed based on a pretest-posttest design, as described by Dimitrov et al. [15]. Pretest-postest designs are widely utilized in behavioral research and are used to measure a change in results based on a treatment or exposure. The main idea is to have participants take a test to measure some results, then expose the participants to some treatment, then test to participants again to see if the treatment led to a change in results.

The pretest-posttest design was utilized in the experiment by having the participants take a test measuring their learning outcomes, then exposing them to the question generation tool for a period of time, then measuring their test results again after the exposure. Ideally, the participants would be divided into a control group, similar to the randomized control-group pretest-posttest design described in Dimitrov et al. [15]. With this design, there would be a group exposed to the question generation tool and a control group not exposed to the tool. Due to the low sample size of this experiment, this was not feasible, and there was therefore no control group.

3.2.2 Participants

The participants in the experiment (n=10) consisted of a group of students from the Norwegian University of Science and Technology. The group consisted of students from different study programs, some first-year students (n=2), some second-year students (n=4), and some fifth-year students (n=4). All the participants had programming experience from university, including JavaScript experience. The participants were recruited through different methods. Through contacting a teacher at the university teaching JavaScript, it was agreed that the author could give a short presentation and try to recruit students from the JavaScript classes. In addition to this, an email with information about the experiment was sent to all the students partaking in both courses. Students were also recruited from the fifth-year Computer Science class at the university. All participants received a gift card of 250 NOK to participate in the experiment.

3.2.3 Technology Acceptance Model

The Technology Acceptance Model was developed by Davis et al. [16] in 1989. The goal of this model was to create a valid measurement for predicting whether a new technology would be accepted by users. Two specific variables were found to be fundamental determinants of users accepting new technologies. These two variables are the perceived usefulness of the technology and the perceived ease of use. These attributes were found to be highly correlated with current usage and self-predicted future usage.

Perceived usefulness

An important factor when users are evaluating whether to use a technology or not is to what extent the user believes it will help them perform their job better. In the Technology Acceptance Model, this is defined as perceived usefulness. This refers to the degree to which a user believes that using a technology would enhance his or her job performance.

Perceived ease of use

After considering whether a technology would help improve job performance, the user may consider how hard the technology is to use. If the technology is too hard to use this might outweigh the potential benefits the user may have from the technology. This is referred to as the perceived ease of use in the Technology Acceptance Model. The ease of use is defined as the degree to which the user believes using the technology would be free of effort.

The Technology Acceptance Model was utilized in this project when evaluating the question generation tool in the experiment. Here the participants answered standardized questions based on the Technology Acceptance Model evaluating the perceived usefulness and the perceived ease of use of the tool. While perceived usefulness and perceived ease of use are highly correlated with user acceptance, user acceptance was also measured directly by asking the participants how likely it is they would use or recommend the tool to others.

3.2.4 General information

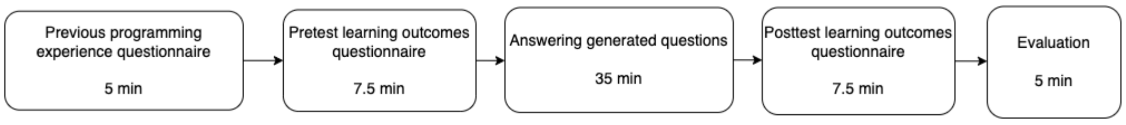

Previous programming experience questionnaire
5 min
Pretest learning outcomes questionnaire
7.5 min
Answering generated questions
35 min
Posttest learning outcomes questionnaire
7.5 min
Evaluation
5 min

Figure 3.3: Pretest-posttest experiment timeline.

The experiment was split into four distinct parts and lasted about 1 hour. The first part was a questionnaire regarding the previous programming experience of the participant, with the students having 5 minutes to answer. The second part was a questionnaire testing their learning outcomes for a chosen programming subject. This questionnaire was answered twice, once before and once after using the question generation tool. The students had 7.5 minutes to answer each time. The third part was using the question generation tool, lasting 35 minutes. The fourth part and last part of the experiment was answering a questionnaire evaluating the tool, having 5 minutes. All the questionnaires were created and answered using Google Forms. The students received no information about the outline of the experiment before starting. The timeline for the experiment can be seen in Figure 3.3.

The topic of the experiment was functional array methods in JavaScript. This topic was chosen because it can be well suited both for participants with no previous experience and participants with more experience. If you have no previous experience it is possible to learn the basics quite fast, and if you have some experience there are more advanced methods to learn. The topic is also highly focused on JavaScript, which is the main target language for the question generation tool. The experiment was conducted over Google Meets. All participants were required to have their cameras on, to make sure that everyone used the allotted time.

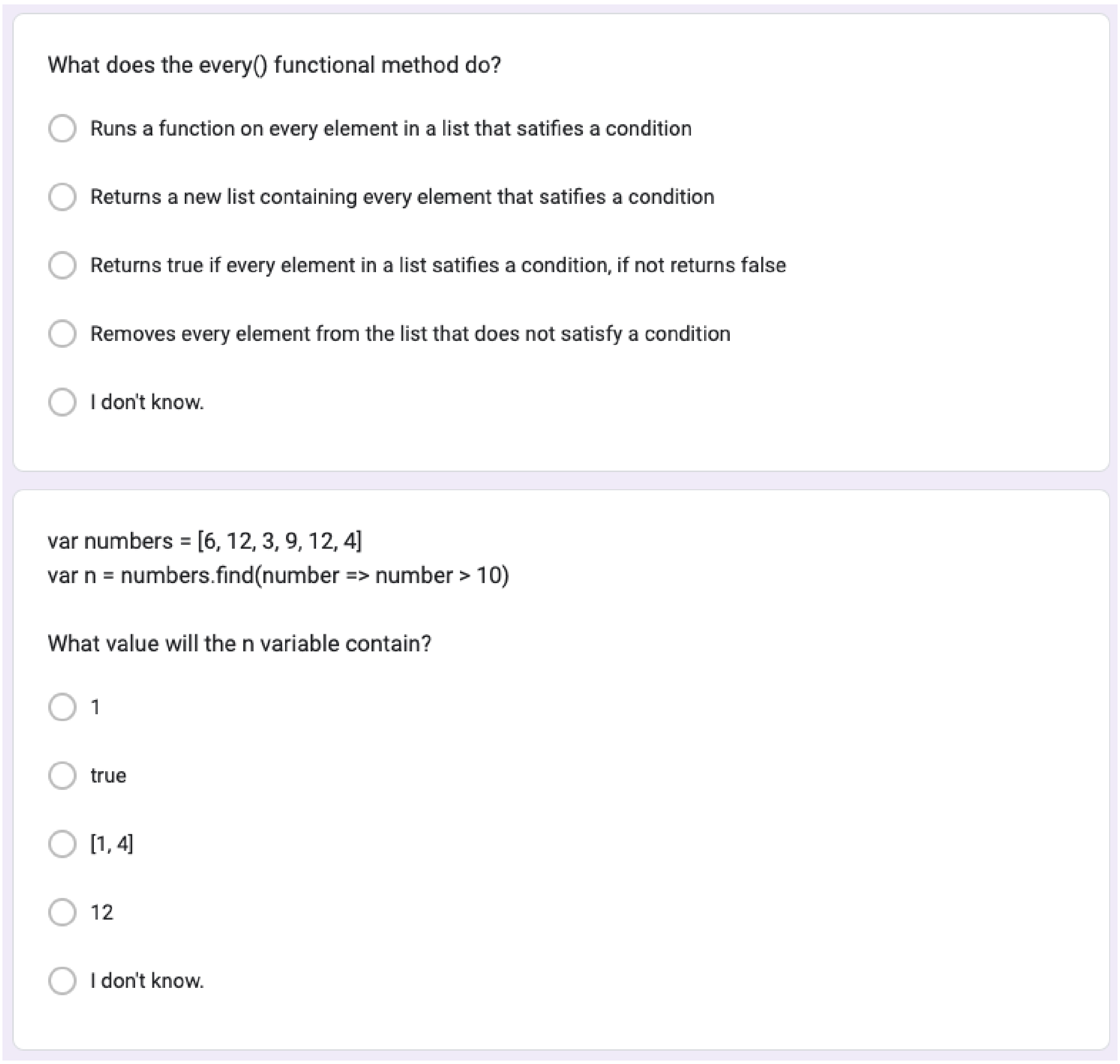
The anonymity of the participants was ensured by generating a unique ID for each participant. The participants then provided this ID in each separate part of the experiment, to make it possible to group the answers by the participant afterward. The only data personal to the participants collected was their study program and year of study, but this data was only attached to their unique ID and not by name. Because of the fact that the experiment was conducted over Google Meets, all the participants could see who the other participants in the experiment were, but there was no way of seeing what the other participants responded in the questionnaire.

3.2.5 Participant experience data

The first part of the experiment was collecting data about each participant's previous programming experience. The participant stated the name of their study program, approximately how many programming course credits they had taken in university and how many JavaScript course credits they had taken. Participants were also asked to rate their overall programming abilities, how much experience they had with JavaScript outside of their studies and state how much experience they have had with functional array methods in JavaScript previously. These were all rated on a 7-point Likert scale. All these questions have the aim of evaluating the participant's programming abilities, JavaScript abilities and array functional methods knowledge.

3.2.6 Learning outcomes

The second part of the experiment is focused on evaluating the participant's learning outcomes for a given topic. As mentioned, the topic chosen for this topic was array functional methods. This part consisted of a questionnaire of 12 questions relating to array functional methods. The questions were multiple choice questions with four different options, in addition to an ”I don't know” option. Participants were informed to choose the ”I don't know” option if they did not know the answer, as guessing the correct answer would not give a real indication of the participant learning outcome. The questions were designed such that all the options could be a plausible answer, and the participant needed a good understanding of the topic to get it correct. Figure 3.4 shows two examples of the questions used in the questionnaire. The questions covered three main topics related to the array methods: an understanding of what each method does, an understanding of when



What does the every() functional method do?

- ◯ Runs a function on every element in a list that satifies a condition
- ◯ Returns a new list containing every element that satifies a condition
- ◯ Returns true if every element in a list satifies a condition, if not returns false
- ◯ Removes every element from the list that does not satisfy a condition
- ◯ I don't know.

```
var numbers = [6, 12, 3, 9, 12, 4]
var n = numbers.find(number => number > 10)
```

What value will the n variable contain?

- ◯ 1
- ◯ true
- ◯ [1, 4]
- ◯ 12
- ◯ I don't know.

Figure 3.4: Example of two of the questions in the learning outcomes questionnaire.

to use each method and knowing the correct syntax for each method. The first ten questions were of medium difficulty, with the two last ones being harder. These last questions were for the participants who already had quite a lot of knowledge of the topic so that they would not get everything correct in the first round. After the participants answered questions from the question generation tool, they were asked to answer the same questionnaire again, to evaluate the learning outcomes after using the tool. They were not informed of this beforehand, to avoid them deliberately focusing on only answering questions on the topics covered in the questionnaire. The participants were also told to not use any external assistance when answering the learning outcomes questions.

3.2.7 Answering generated questions

The third part of the experiment is the participants answering questions generated by the question generation tool. The participants were given 35 minutes to utilize the tool. No instructions were given on how to use the tool, and the application contained no tutorial explaining how to use it. This was a deliberate decision, to evaluate how intuitive the user interface of the tool is. The participants were asked to not use any assistance during the 35 minutes, with the exception of link to a site containing documentation for all the array functional methods [17]. This was done to ensure that all the participants had the same information available to solve the questions. All the questions were designed to be able to be answered using only the documentation link. It was necessary to have some sort of documentation available, as some participants might not be familiar with a given array method.

When using the tool during the experiment, the participants would use the tool to answer programming questions relating to array functional methods in JavaScript. There were 55 unique questions generated in total by the author, divided into two categories: standard and intermediate, with 40 of the questions being standard and 15 intermediate. Before starting using the tool the participants were informed they could choose which difficulty they wanted, but it was recommended to start on the standard questions and then potentially move towards the intermediate questions later. The generated questions covered a range of topics within the subject of array functional methods, with all of the questions being generated using the question generation tool. Examples of the type of questions the participants answered are shown in Figure 4.9, Figure 4.10 and Figure 4.11.

3.2.8 Evaluation

The fourth and last part of the experiment was for the participants to answer a questionnaire evaluating the question generation tool. This evaluation was based on the Technology Acceptance Model, as described previously in this chapter. The evaluation questions were split into three categories, perceived usefulness, perceived ease of use and user acceptance. The questions were answered on a 7-point Likert scale, with there being 10 questions in total. Perceived usefulness evaluates the usefulness of the tool in achieving the goal of the tool, and consisted of the first four questions in the form. These questions referred to how effective the tool was in learning new programming concepts and how useful the tool would be as a part of their studying routine. The next category was perceived ease of use, referring to how understandable and easy to use the tool was, consisting of three questions. The last category was user acceptance, meaning how likely it is the user would actually utilize or recommend the tool in a real-world setting, consisting of three questions. Lastly, there were two open-text questions were the participants could give more in-depth answers. The first question was about the learning experience of using the tool and the second asked for any suggestions for improvements to the tool.

Chapter 4

Results

This chapter is split into two sections and contains the results from the question generation tool and the results from the experiment. The question generation tool results section is a presentation of the final prototype with all the functionality and the experiment results section contains the results from the evaluation of the tool.

4.1 Question generation tool results

This section will be a presentation of the final question generation tool prototype, including a description of all the functionality included in the tool. The question generation tool developed for this project is split into two major components. The first component consists of functionality for generating question variants, to be used by teachers. The second component is an intuitive interface for students to answer the question variants generated by the teacher.

4.1.1 System architecture

An overview of the system architecture is provided in Figure 4.1. The architecture is relatively simple, with there being four main technologies used as a part of the system. The majority of the system consists of the React-application, which includes all the code for the user interface, generation of questions, answering questions and communication with the other parts of the system. The React application communicates with the Azure Cosmos Database by fetching and uploading documents, which is used to store generated question documents and user metrics. The React application also communicates with the Judge0 API, which is the API used

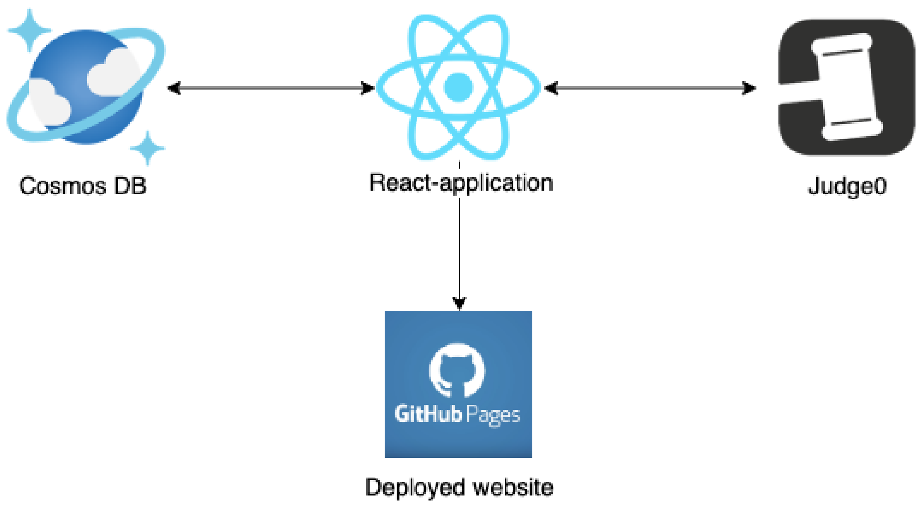


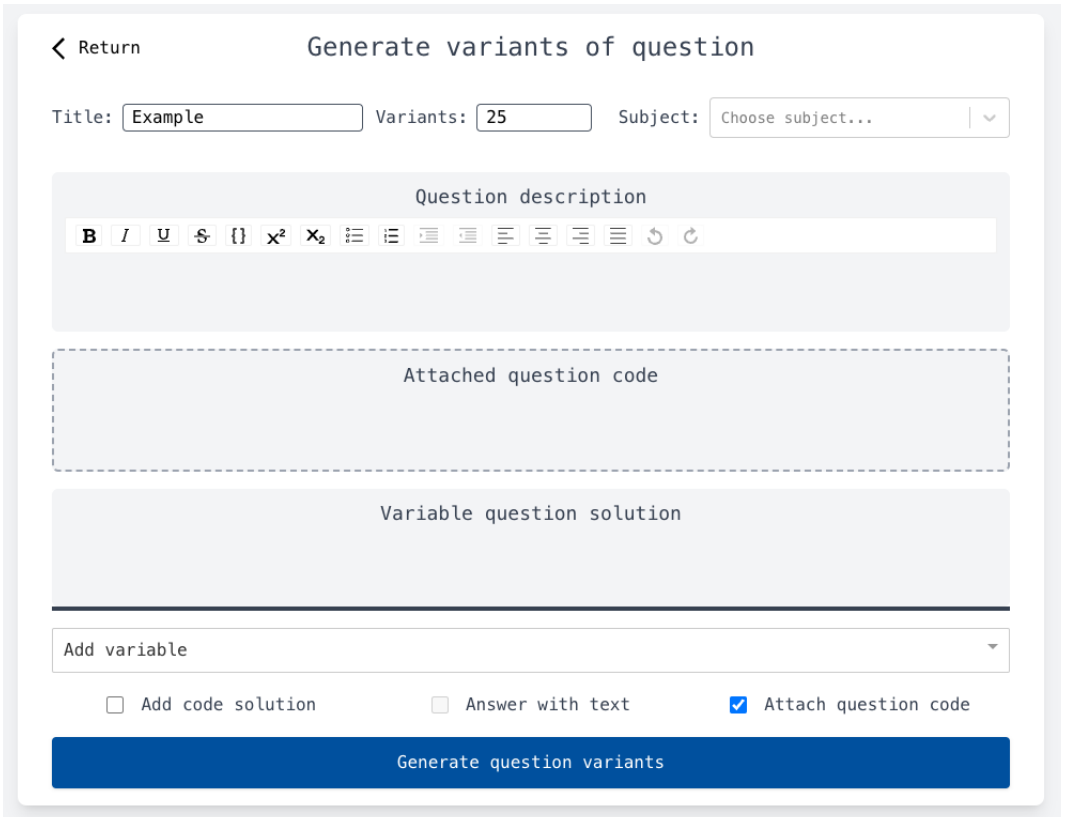
Figure 4.1: Overview of the system architecture.

to compile and execute code in the cloud. The React application will send code submitted by the users to the Judge0 API, which will then execute and return the output of the submitted code. Lastly, GitHub Pages is used to actually deploy the React application to a hosted website.

4.1.2 Question generation

The core of the tool developed for this project is the functionality related to question generation. The user interface for the question generation capabilities of the application is presented in Figure 4.2. The goal of this part of the application is to be able to create a question template, which then can be used for generating variants based on some variables provided by the user. There are three main question types, all of which will be described in detail later.

As seen at the top of the figure, the user can provide some information about the given question. Firstly it is possible to give the question a descriptive title. This is mainly for being able to identify the unique questions and to give the students answering the question an idea of the question topic. Next, the user can provide the number of unique variants they wish to generate. There is no upper limit for how many unique variants can be generated, so the user can generate as many variants as needed. Next, it is possible to categorize the subject for the question. This makes it possible to group together questions that belong to the same subject or class. Here it is also possible to categorize the question on difficulty.


Return

Generate variants of question

Title: Example Variants: 25 Subject: Choose subject...

Question description

Attached question code

Variable question solution

Add variable

☐ Add code solution ☐ Answer with text ☑ Attach question code

Generate question variants

Figure 4.2: Generation of question variants.

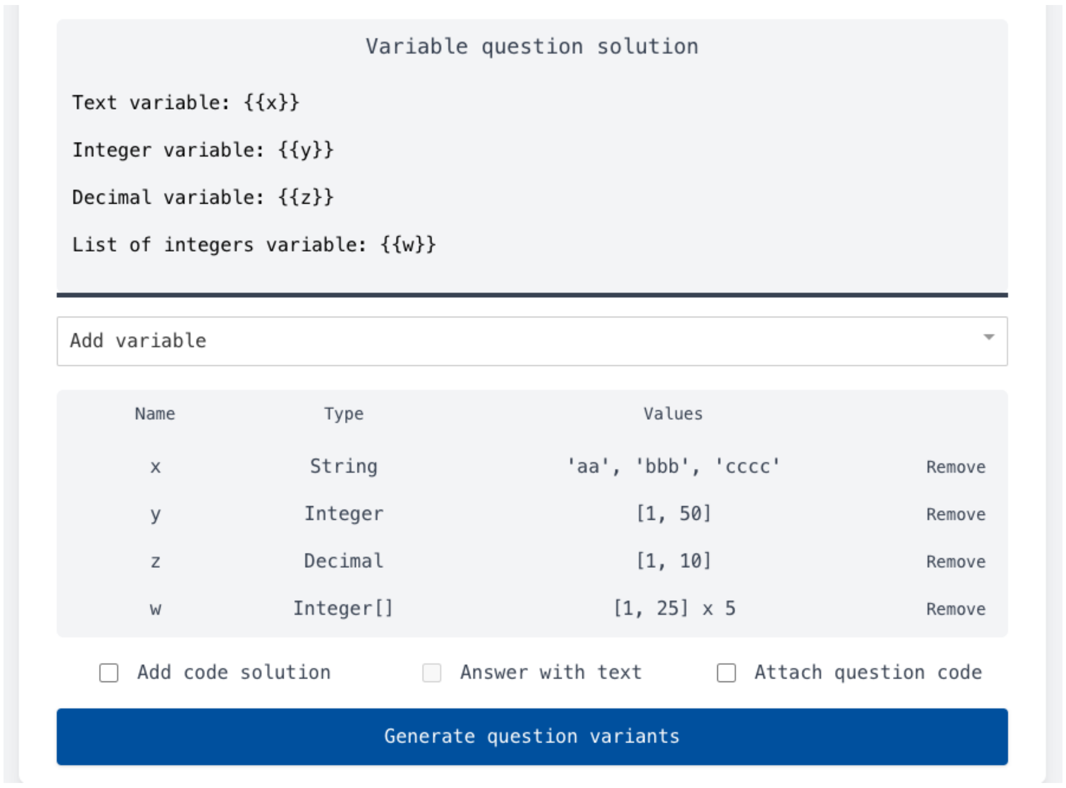
Below there is a field for the user to provide a question description. This is a text description that will be constant for all the unique variants, so here the user could potentially give some background information for the question or some general information about the question. Because this description is constant across all variants, it is important that the information applies to all the variants. This part is optional, so if there is no general description for the question it can be left empty.

After the general question description, there is a field for attaching some code that is relevant to the question. This part is also optional and can be added or removed by checking the "attach question code" checkbox. This field might be used to attach code that is relevant, for example, a class or method that might be used as a basis for the question. This code will be set as the initial code in the code editor when answering questions.

The next field is the variable question solution field, which is the field responsible for creating the variants of the question. This field contains the parts of the generated questions that vary. This field allows for adding normal plain text, in addition to being able to add question variables. Variables can be added through the "Add variable" drop-down, with four different types of variables available:

- **Text**: a list of possible string values
- **Integer**: a number interval defined as a minimum and maximum value
- **Decimal**: a decimal number interval defined as a minimum and maximum value
- **List of integers**: a number interval defined as a minimum and maximum value, with the length of the array

For text variables, the user can provide a list of the different possible values the variable can contain. For integer and decimal values, the user must define a minimum and maximum value. For a list of integer variables, the user must define a minimum and maximum value and the desired length of the list. The user must also define a unique variable name for each variable added.



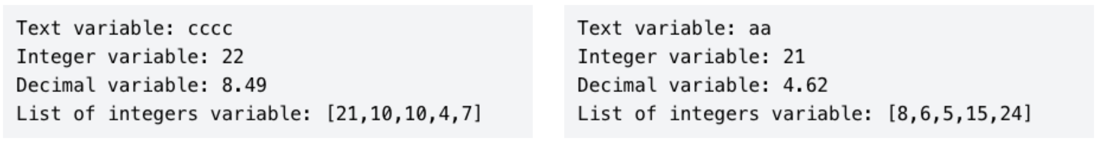
Variable question solution

```
Text variable: {{x}}

Integer variable: {{y}}

Decimal variable: {{z}}

List of integers variable: {{w}}
```

Name	Type	Values	
x	String	'aa', 'bbb', 'cccc'	Remove
y	Integer	[1, 50]	Remove
z	Decimal	[1, 10]	Remove
w	Integer[]	[1, 25] x 5	Remove

Figure 4.3: Example of using the four different variable types.

When a question is generated all the instances of a variable in the variable question solution field will be replaced with an actual value based on the parameters provided for the given variable. The variables are contained in double curly brackets, so the tool will search the text for any valid instance of double curly brackets and replace it with a valid value. For a text variable, a random value in the provided list of strings will replace the variable. For an integer variable, a random integer in the



```
Text variable: cccc
Integer variable: 22
Decimal variable: 8.49
List of integers variable: [21,10,10,4,7]
```

```
Text variable: aa
Integer variable: 21
Decimal variable: 4.62
List of integers variable: [8,6,5,15,24]
```

Figure 4.4: Two example variants generated from the template in Figure 4.3
.

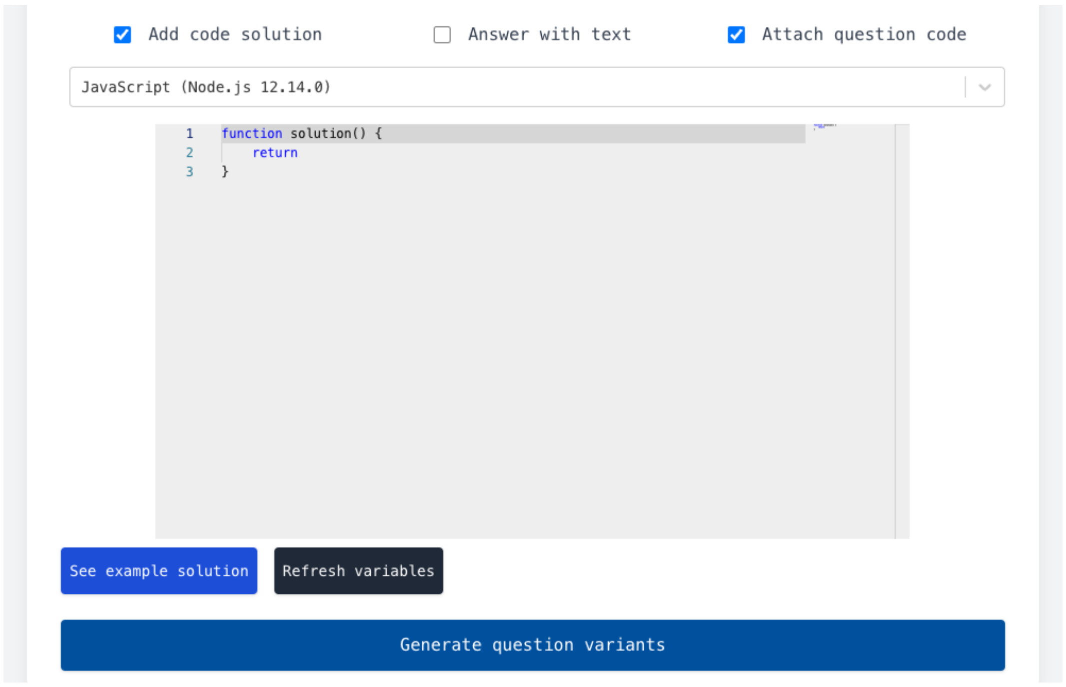
provided interval will be generated and substituted for the variable. For a decimal variable, a random decimal value in the interval will be generated. For the list of integer variable, a list with the provided length will be created, with each value in the list containing a randomly generated integer in the interval.

An example of how the variables are defined is provided in Figure 4.3. Here a template is created containing some text, including an example of each variable type. Figure 4.4 is an example of two variants generated from the template, where each variable is replaced by a value based on the parameters.

Below the variable question solution field there are three different check-boxes. These can be toggled to add different functionality which can be used to create different question types. As mentioned, there are three main question types that can be generated:

Coding questions

The first type of question is simply called coding questions and is the main method for creating questions using the tool. It is the question type that allows for the most amount of variation, and the most creativity. To create a coding question, the user needs to check the ”add code solution” checkbox, which will make a code editor window appear, as seen in Figure 4.5. Below are two buttons, ”see example solution” and ”refresh variables”. What defines the coding question type is the ability of the teacher to write a question solution that depends on the added variable questions. The code editor has a function called *solution*(), which defines the solution to the question. When pressing the ”refresh variables” button, all the question variables will be added as parameters in the function. The teacher can then write their desired code, using the question variables. By pressing the ”see example solution” button, the tool will randomly generate a set of valid variable values and display the solution for those variable values. This is useful for checking that the solution function provides the correct answer for many different combinations of variables. An example of how a coding question might be designed is provided in Figure 4.6, which was one of the question templates used in the experiment. An example of a



JavaScript (Node.js 12.14.0)

```
function solution() {
    return
}
```

Figure 4.5: Editor for adding a code solution to a question.

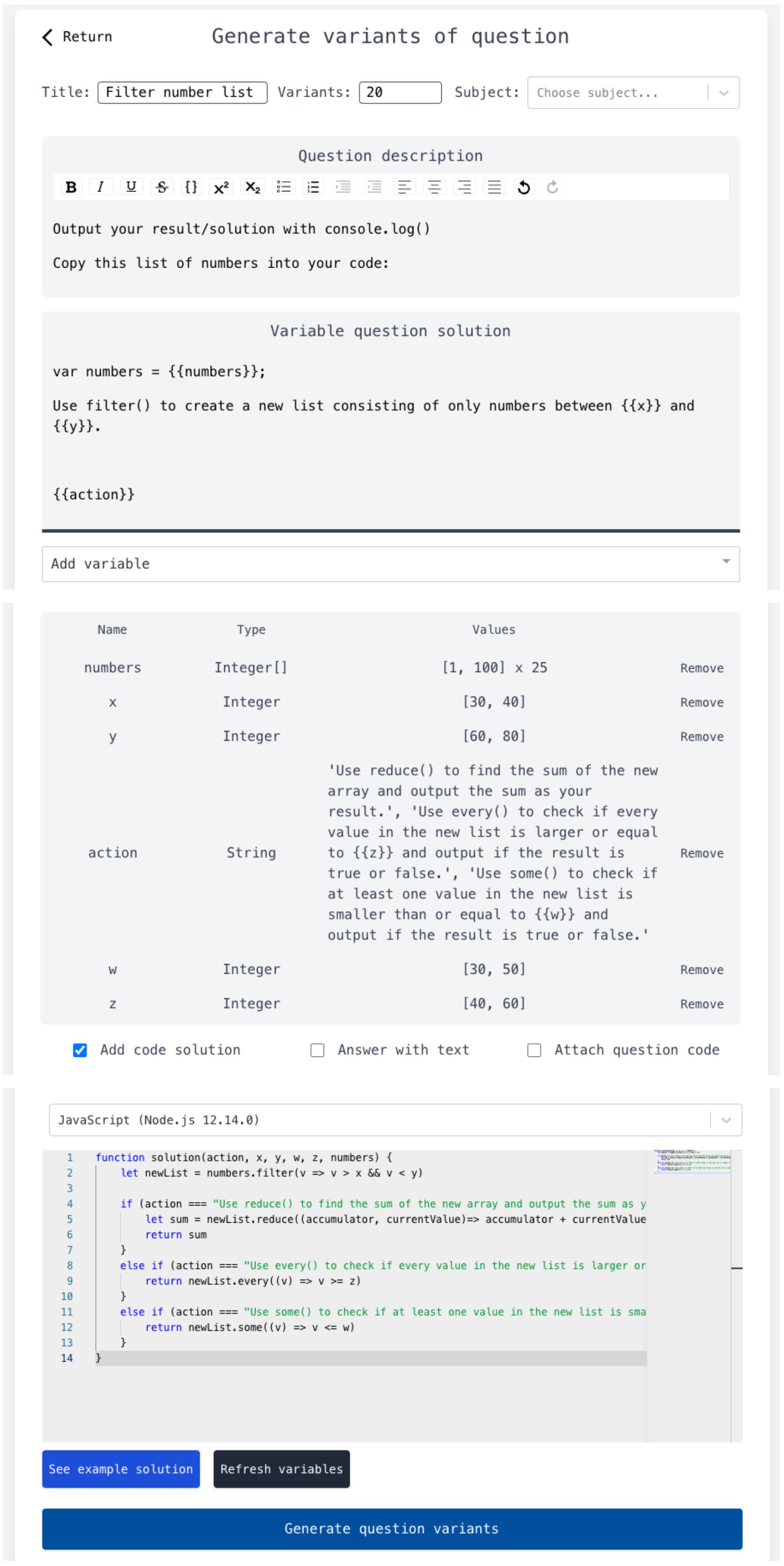
question generated by this template can be seen in Figure 4.9.

Code understanding questions

The second type of question is code understanding questions. Unlike the coding questions, where the students are asked to write code that provides the correct answer, the coding understanding questions are based on being able to understand and debug existing code. This type of question is generated by checking the "add code solution" and the "answer with text" checkboxes. The teacher can then write a coding solution using the question variables. The difference here is that the student will be given the code solution as part of the question, and then asked questions about what the code outputs. An example of a code understanding question is provided in Figure 4.7, with an example of a generated question in Figure 4.11.

Produce output questions

The third type of question is the produce output questions. This is the question type that allows for the least amount of variance, but it can be useful in some situations. The idea behind this question type is that the student will be asked to write some code that produces a given output, where the output varies between questions. This question type is created by un-checking both "add code solution" and "answer with



Title: Filter number list Variants: 20 Subject: Choose subject...

Question description

Output your result/solution with console.log()

Copy this list of numbers into your code:

Variable question solution

var numbers = {{numbers}};

Use filter() to create a new list consisting of only numbers between {{x}} and {{y}}.

{{action}}

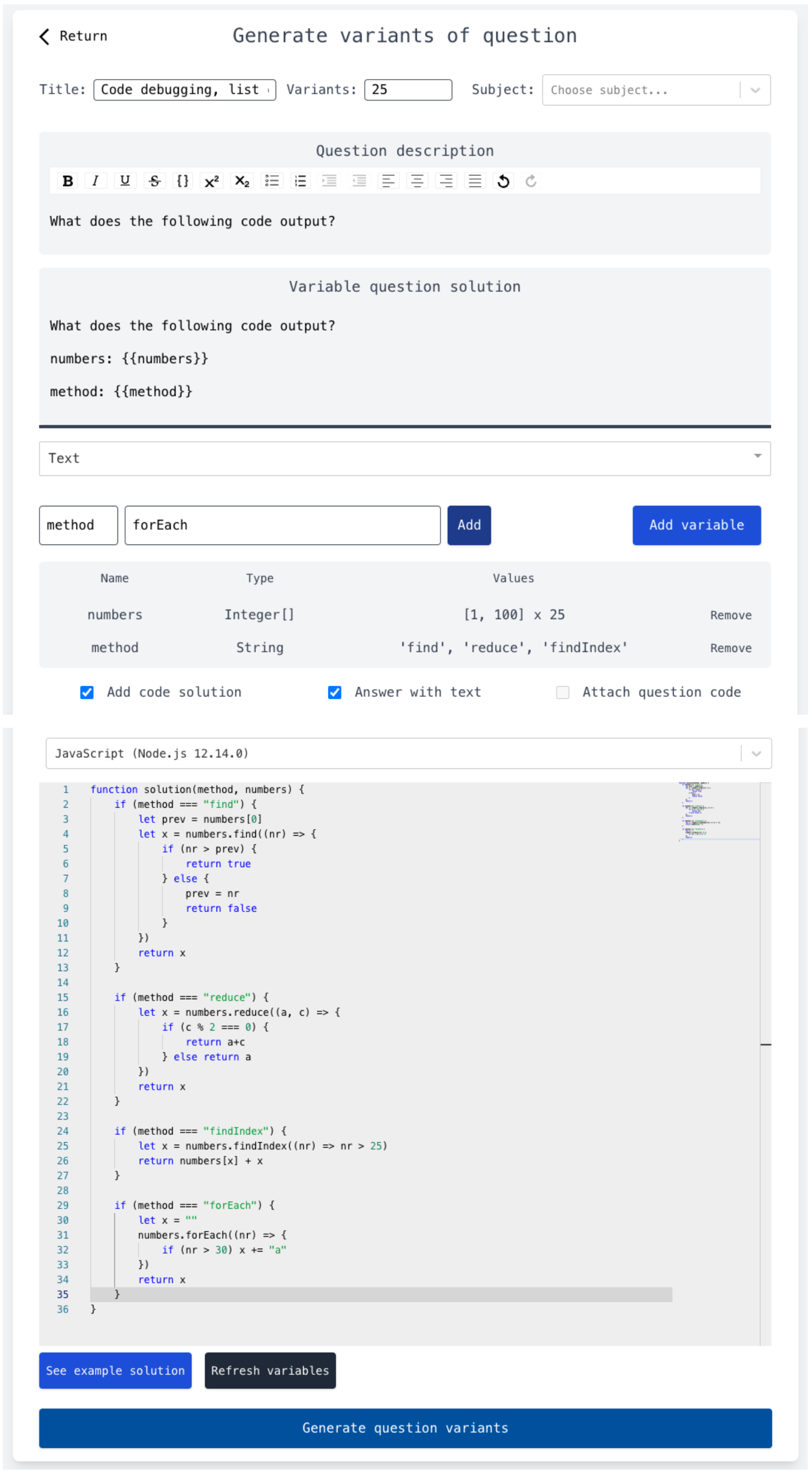
Name	Type	Values	
numbers	Integer[]	[1, 100] x 25	Remove
x	Integer	[30, 40]	Remove
y	Integer	[60, 80]	Remove
action	String	'Use reduce() to find the sum of the new array and output the sum as your result.', 'Use every() to check if every value in the new list is larger or equal to {{z}} and output if the result is true or false.', 'Use some() to check if at least one value in the new list is smaller than or equal to {{w}} and output if the result is true or false.'	Remove
w	Integer	[30, 50]	Remove
z	Integer	[40, 60]	Remove

[✓] Add code solution ☐ Answer with text ☐ Attach question code

JavaScript (Node.js 12.14.0)

```
function solution(action, x, y, w, z, numbers) {
    let newList = numbers.filter(v => v > x && v < y)

    if (action === "Use reduce() to find the sum of the new array and output the sum as y
        let sum = newList.reduce((accumulator, currentValue)=> accumulator + currentValue
        return sum
    }
    else if (action === "Use every() to check if every value in the new list is larger or
        return newList.every((v) => v >= z)
    }
    else if (action === "Use some() to check if at least one value in the new list is sma
        return newList.some((v) => v <= w)
    }
}
```

Figure 4.6: Example of a coding question template.



Title: Code debugging, list
Variants: 25
Subject: Choose subject...

What does the following code output?

What does the following code output?

numbers: {{numbers}}

method: {{method}}

Text

method | forEach

Name	Type	Values	
numbers	Integer[]	[1, 100] x 25	Remove
method	String	'find', 'reduce', 'findIndex'	Remove

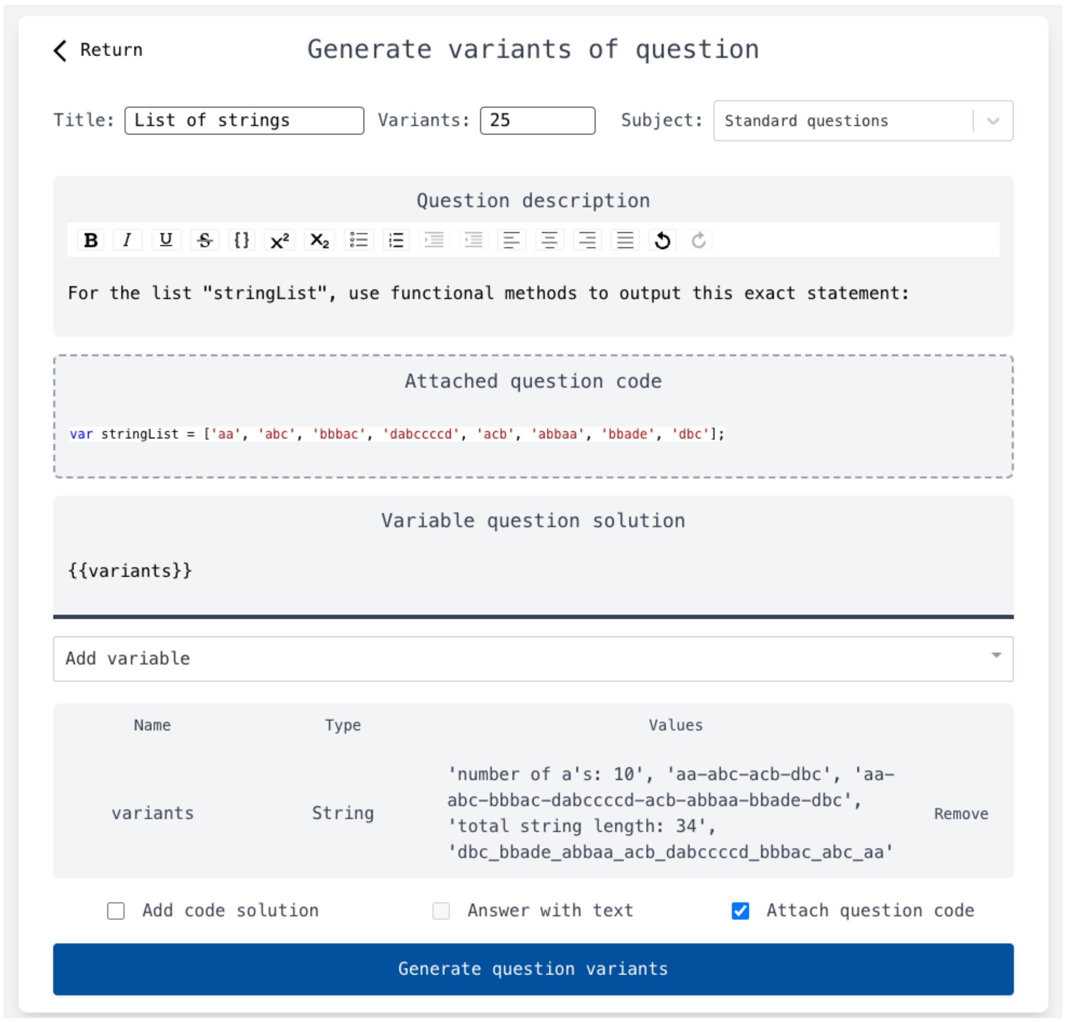
[✓] Add code solution [✓] Answer with text [ ] Attach question code

JavaScript (Node.js 12.14.0)

```javascript
function solution(method, numbers) {
    if (method === "find") {
        let prev = numbers[0]
        let x = numbers.find((nr) => {
            if (nr > prev) {
                return true
            } else {
                prev = nr
                return false
            }
        })
        return x
    }

    if (method === "reduce") {
        let x = numbers.reduce((a, c) => {
            if (c % 2 === 0) {
                return a+c
            } else return a
        })
        return x
    }

    if (method === "findIndex") {
        let x = numbers.findIndex((nr) => nr > 25)
        return numbers[x] + x
    }

    if (method === "forEach") {
        let x = ""
        numbers.forEach((nr) => {
            if (nr > 30) x += "a"
        })
        return x
    }
}
```

Figure 4.7: Example of a code understanding question template.

text". Attaching question code is optional. The teacher will write the desired code output in the "variable question solution" field, which can contain both text and variables. An example of a produce output question template is provided in Figure 4.8 with a generated question in Figure 4.10.



Title: List of strings Variants: 25 Subject: Standard questions

Question description

For the list "stringList", use functional methods to output this exact statement:

Attached question code

```
var stringList = ['aa', 'abc', 'bbbac', 'dabccccd', 'acb', 'abbaa', 'bbade', 'dbc'];
```

Variable question solution

{{variants}}

Name	Type	Values	
variants	String	'number of a's: 10', 'aa-abc-acb-dbc', 'aa-abc-bbbac-dabccccd-acb-abbaa-bbade-dbc', 'total string length: 34', 'dbc_bbade_abbaa_acb_dabccccd_bbbac_abc_aa'	Remove

☐ Add code solution ☐ Answer with text ☑ Attach question code

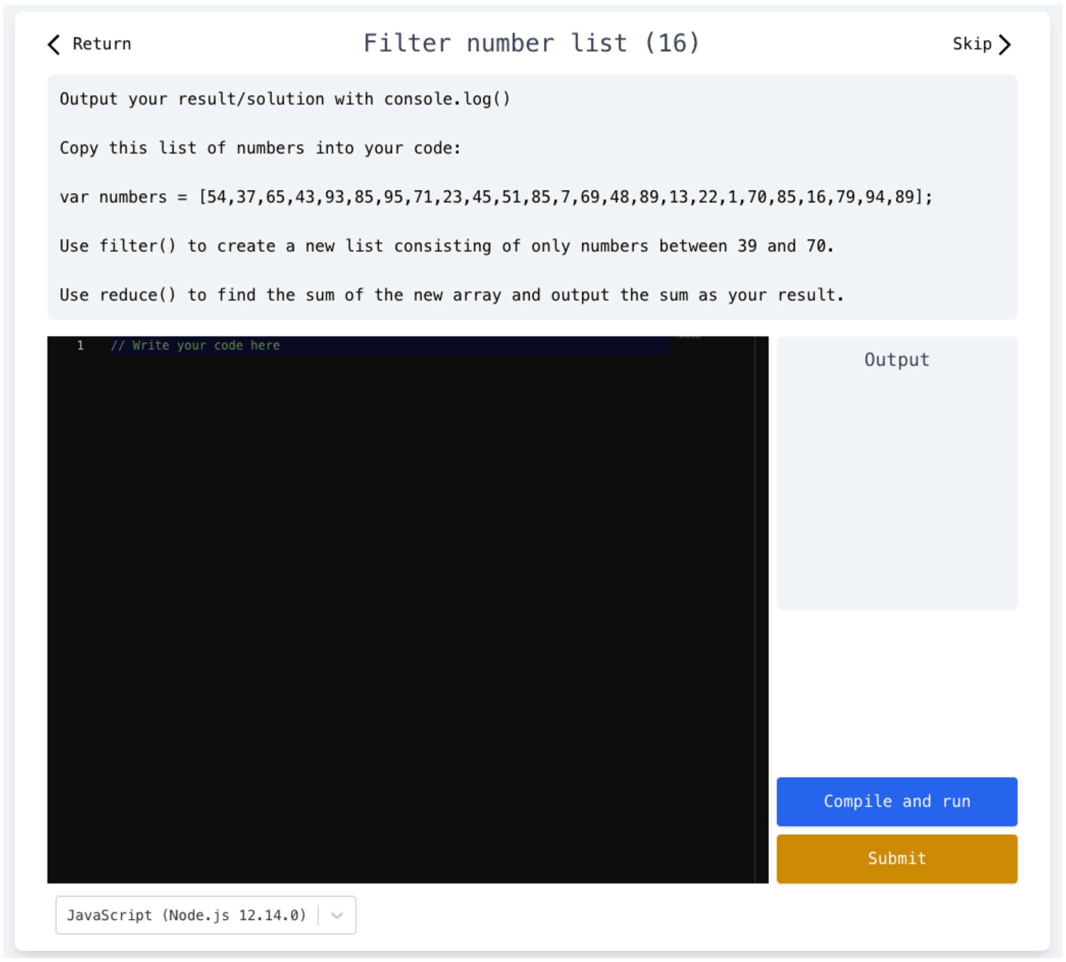
Figure 4.8: Example of a produce output question template.

4.1.3 Answering questions

The next major component of the tool is the functionality relating to answering the generated questions. While the generation of questions is the main research topic of the project, it is also important to have a user interface for solving the questions. The formatting of the questions is quite specialized, so it was necessary to create a system that supported the question formats and provided all the necessary functionality.

As described in the previous section, there are three main question types. As the nature of the question types requires different solutions from the student, there are two different interfaces for viewing and answering questions.

Coding questions

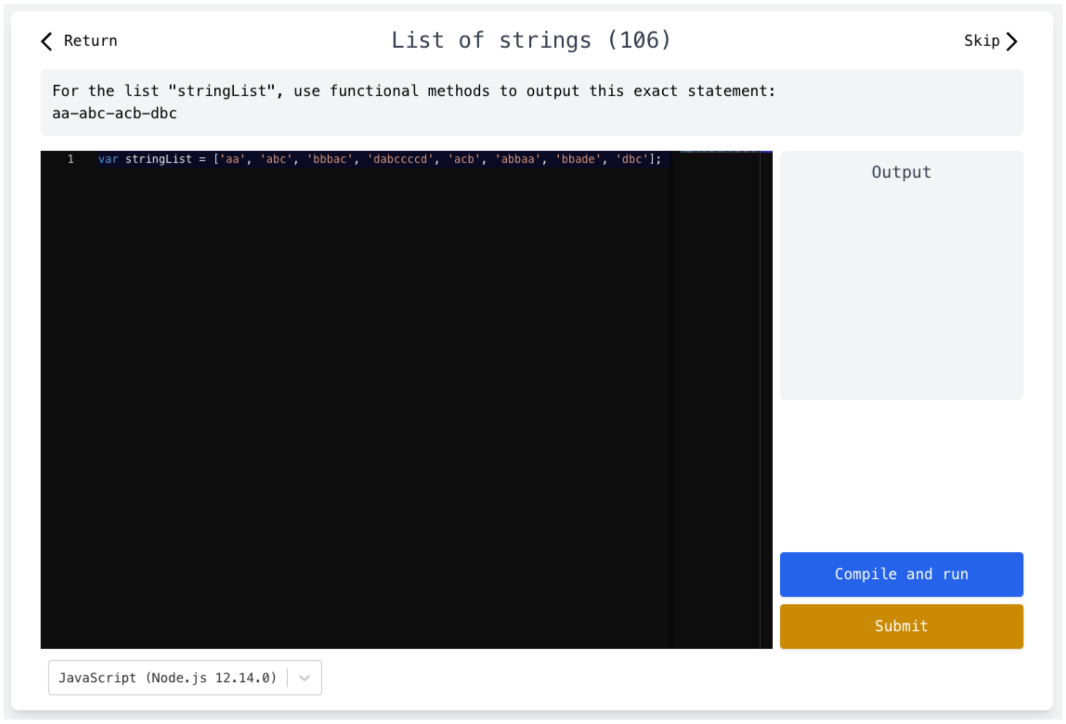

Return
Filter number list (16)
Skip

```
Output your result/solution with console.log()

Copy this list of numbers into your code:

var numbers = [54,37,65,43,93,85,95,71,23,45,51,85,7,69,48,89,13,22,1,70,85,16,79,94,89];

Use filter() to create a new list consisting of only numbers between 39 and 70.

Use reduce() to find the sum of the new array and output the sum as your result.
```

```
// Write your code here
```

Figure 4.9: User interface for solving coding questions.

The first interface is created for the questions that require the student to write and submit code, as depicted in Figure 4.9 and Figure 4.10. This interface is used for answering the coding question type or the produce output question type, as both these require the student to submit code. At the top of the screen is the title of question, including the ability to return to the home menu on the left or to skip the current question to the right. Below is the question description and the variable question solution. After the question description is the code editor, where the student can choose between writing JavaScript or TypeScript. If the question contains some attached question code, the code will appear in the code editor when loading the question. If not the editor will be empty. To the right of the code editor is a code output window, where the output of the code is presented, or an error if something goes wrong. Below are the options for either compiling and running the code, or for submitting. Compiling and running the code can be used by the student to test their code to see if it works as they intended. When submitting the code, the code will be executed and the output of the submission will be compared to the question solution defined by the question template. If the answer is correct, the student can proceed to the next question.



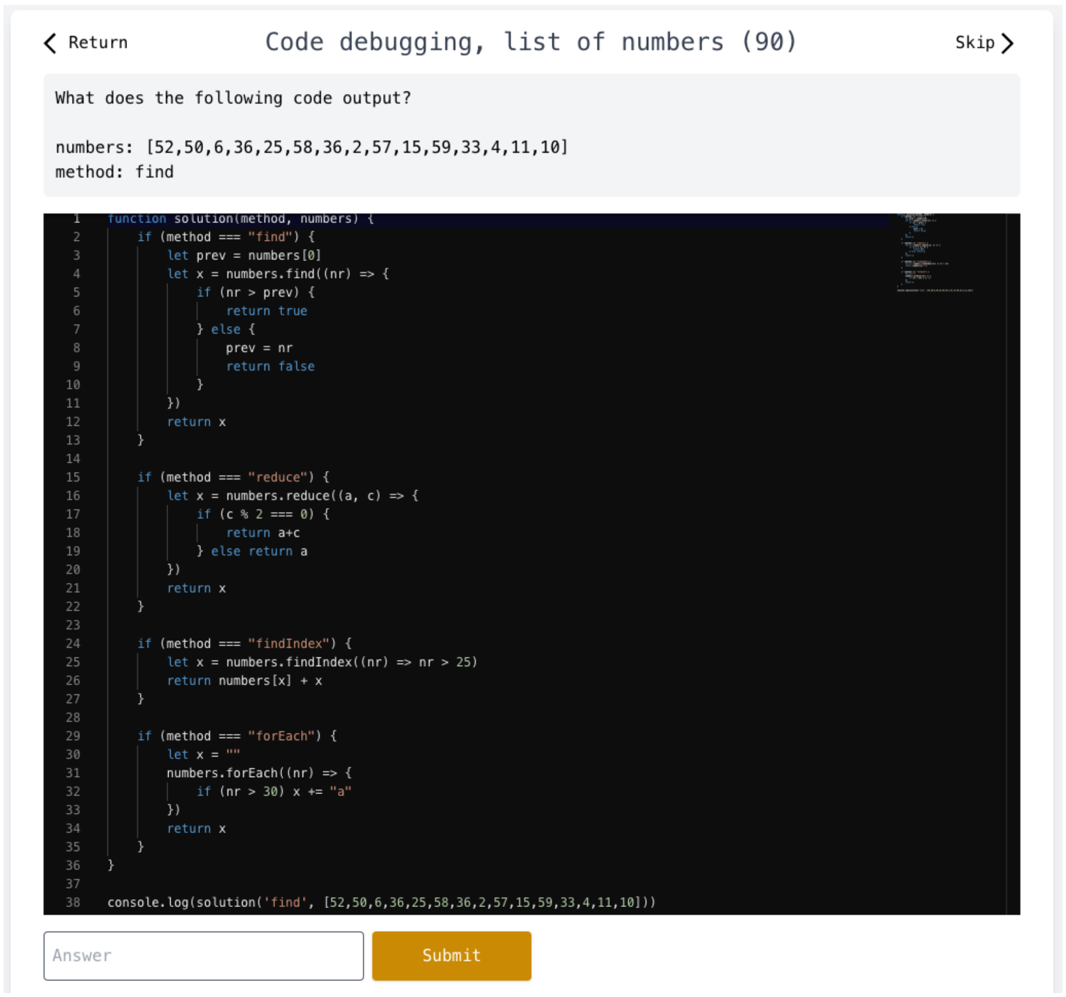
For the list "stringList", use functional methods to output this exact statement:
aa-abc-acb-dbc

```
var stringList = ['aa', 'abc', 'bbbac', 'dabccccd', 'acb', 'abbaa', 'bbade', 'dbc'];
```

Figure 4.10: User interface for solving produce output questions.

Code understanding questions

The second interface is for the code understanding questions. For these questions, the user does not need to write code, and there is therefore no need for a code editor. This interface is seen in Figure 4.11. The top contains the same parts as the previous interface: title, return and skip button and question description. Below this is a read-only code window containing the code question solution defined by the teacher. The student will be asked to read the code and answer what the output of the code is, given some information provided in the question. The student provides their answer in a text field beneath the code window. When submitted, the answer will be compared with the correct output and evaluate whether it is correct or wrong.



What does the following code output?

numbers: [52,50,6,36,25,58,36,2,57,15,59,33,4,11,10]
method: find

```
function solution(method, numbers) {
    if (method === "find") {
        let prev = numbers[0]
        let x = numbers.find((nr) => {
            if (nr > prev) {
                return true
            } else {
                prev = nr
                return false
            }
        })
        return x
    }

    if (method === "reduce") {
        let x = numbers.reduce((a, c) => {
            if (c % 2 === 0) {
                return a+c
            } else return a
        })
        return x
    }

    if (method === "findIndex") {
        let x = numbers.findIndex((nr) => nr > 25)
        return numbers[x] + x
    }

    if (method === "forEach") {
        let x = ""
        numbers.forEach((nr) => {
            if (nr > 30) x += "a"
        })
        return x
    }
}

console.log(solution('find', [52,50,6,36,25,58,36,2,57,15,59,33,4,11,10]))
```

Figure 4.11: User interface for solving code understanding questions.

4.1.4 Navigating the tool

The tool has a simple menu system for navigating the different parts of the system. There is a main menu to be used by students for navigating the questions, and there is a separate admin menu for navigating the admin functionalities.

Menu system

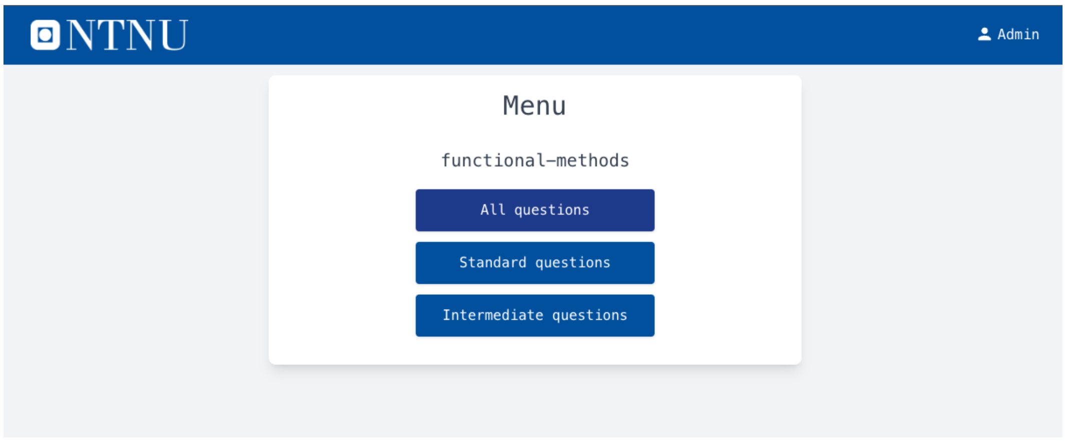


Figure 4.12: Main menu user interface.

The main menu can be seen in Figure 4.12. The menu is simply an overview of all the different question subjects, with each subject having a list of under-categories. When navigating to a topic from the main menu, the student will be presented with questions tagged with that category and can start solving the questions right away. It is also possible to access "all questions" within a subject, meaning all questions from the under-categories for that subject. When students have navigated to a subject, all the questions they receive when either skipping or moving to the next question will be from that subject. If a student wishes to solve questions relating to another subject, they can navigate back to the main menu and choose another subject. The under-categories for a subject can be used for categorizing into more specific subjects within the main subject, or they can be used to categorize questions based on the difficulty, which is the case in Figure 4.12.

Admin

The admin menu is accessible by clicking the admin button on the top right of the screen and logging in with a specific username and password, as seen in Figure 4.13. After logging in, the admin has access to the admin menu, seen in Figure 4.14.

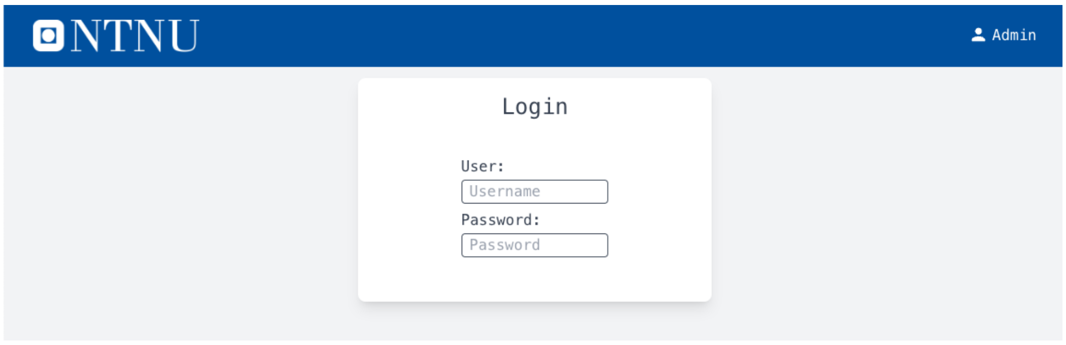


Figure 4.13: Admin login screen.

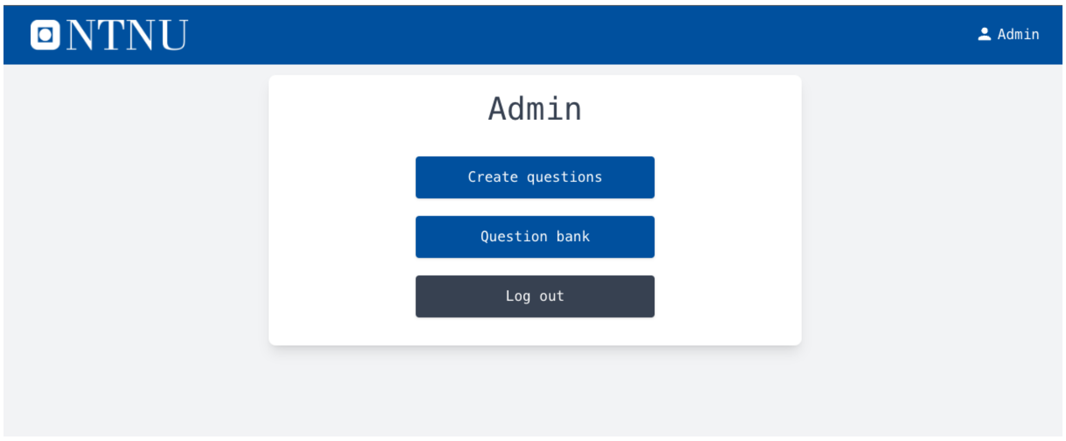
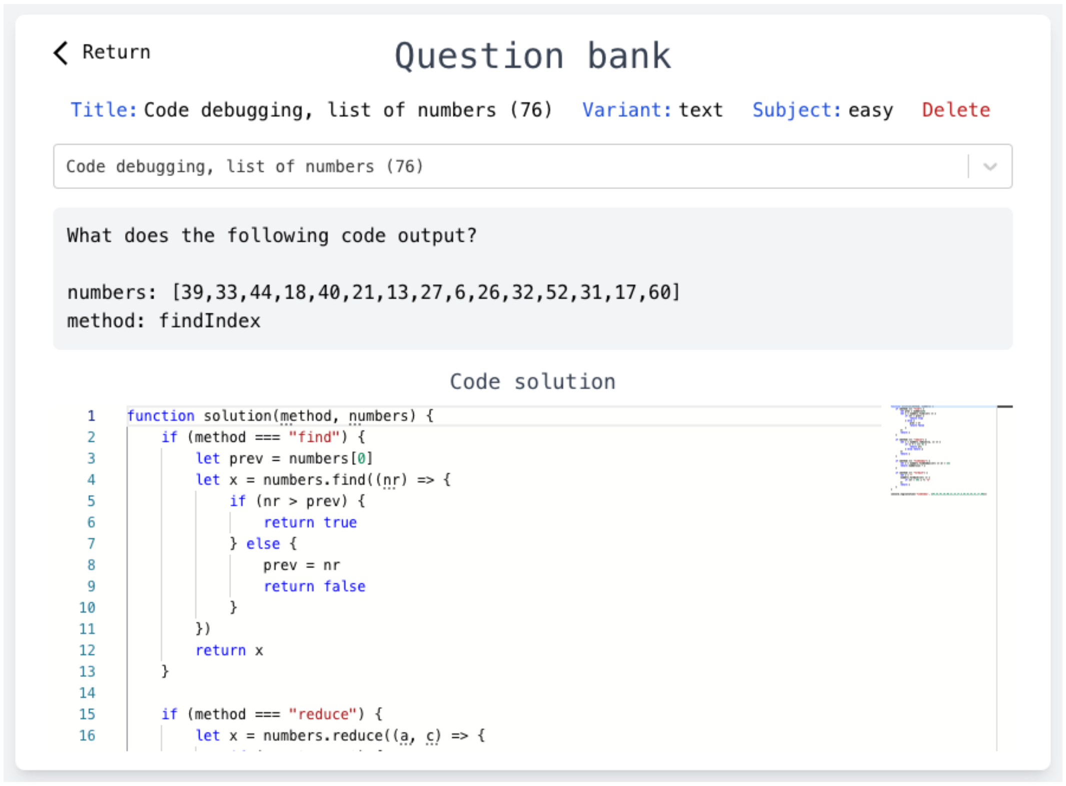


Figure 4.14: Admin menu user interface.

This menu contains the page for generating questions, as described in depth in the previous section, and for viewing the question bank. The question bank contains the information about all the question instances generated, including title, description, category, attached code and question solution. The question bank is depicted in Figure 4.15.



Return
Question bank
Title: Code debugging, list of numbers (76)
Variant: text
Subject: easy
Delete
Code debugging, list of numbers (76)

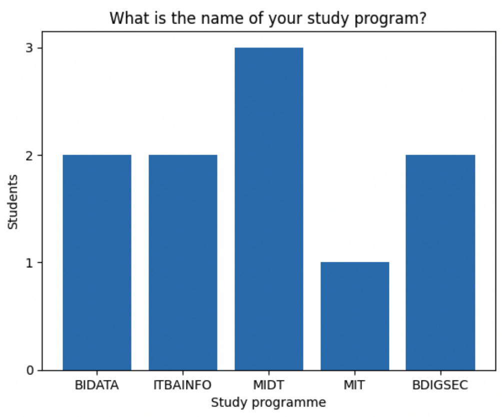
```
What does the following code output?

numbers: [39,33,44,18,40,21,13,27,6,26,32,52,31,17,60]
method: findIndex
```

```
function solution(method, numbers) {
    if (method === "find") {
        let prev = numbers[0]
        let x = numbers.find((nr) => {
            if (nr > prev) {
                return true
            } else {
                prev = nr
                return false
            }
        })
        return x
    }

    if (method === "reduce") {
        let x = numbers.reduce((a, c) => {
```

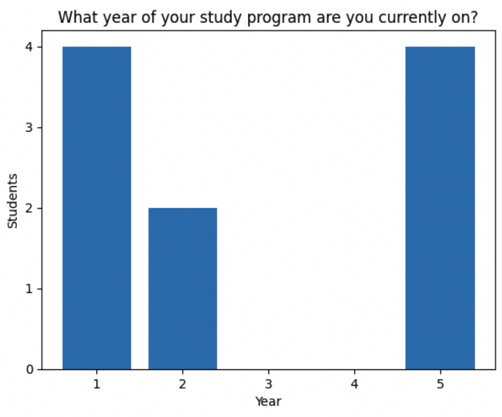
Figure 4.15: Question bank interface.

4.2 Experiment results

In this section, the results from the evaluation of the question generation tool will be presented. This includes the results from the four different parts of the experiment, the first being the data collected about the programming experience of the participants, the second being their results of the learning outcomes questionnaires, the third being the results from answering questions using the tool and lastly, the evaluation results using the Technology Acceptance Model.



(a) Study programme.



(b) Year of study.

Figure 4.16: Participant study programmes and year of study.

4.2.1 Participant experience data results

The part first part of the experiment consisted of collecting some data about the participant's previous programming experience, to get some idea of their current programming knowledge. The first two questions asked about the name of their study programme and what year of their studies they were on. The participants came from a range of different study programmes, with some being more focused on programming than others. The uniting factor of all the participants is that they had taken credits in a JavaScript programming course. Figure 4.16 shows how many students were from each study programme. BIDATA refers to a bachelor in Computer Science, ITBAINFO refers to a bachelor in Information Technology, MIDT refers to a master in Computer Science, MIT refers to a master in Informatics and BDIGSEC refers to a bachelor in Digital Infrastructure and Cyber Security. The next graph in the figure shows what year of their studies the participants, with the participants all being either first, second or fifth year students. This was expected, as the classes that the students were recruited from were either first, second or fifth year. Some of the fifth year students had answered that they were on their second year, as they were on the second year of their 2-year master degree, but this was changed to fifth year in the data to make the answers consistent.

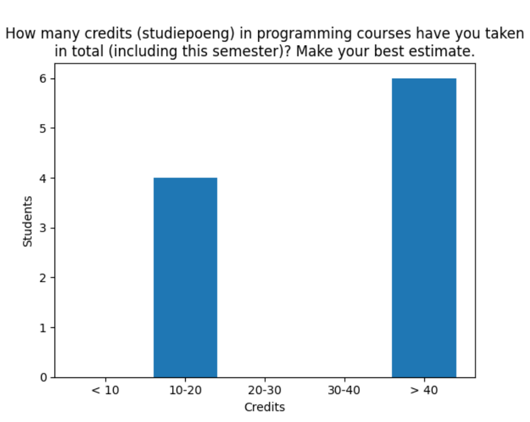
After asking about the participant's study programme and study year, they were asked a range of questions relating to their programming experience. They were asked to estimate how many programming credits they had taken in university, and how many JavaScript programming credits. They were asked to choose between five categories, under 10 credits, between 10 and 20 credits, between 20 and 30 credits, between 30 and 40 credits or over 40 credits. They were also asked to rate their overall programming abilities on a 7-point scale from poor to excellent, to rate their experience with JavaScript outside of their studies and their experience with functional array methods in JavaScript on a 7-point scale from no experience to very experienced.

The data showed that over half of the participants had over 40 programming credits while under half had between 10-20 credits, as seen in Figure 4.17. None of the other options were chosen, as seen in graph (a). The participants all also rated their own programming abilities as average or above average, as seen in graph (c). This means that all the participants were somewhat familiar with programming in general, with no participants being very inexperienced with programming. When it came to JavaScript programming credits, the majority had between 10-20 credits with some outliers, as seen in graph (b). When asked about experience with JavaScript outside of their studies in graph (d), the answers were quite varied, with most having some experience, some having no experience and some having a lot. When it came to experience with the array functional method almost half answered they were very experienced, with the rest having varying experience. All participants had at least some experience with functional array methods beforehand, seen in graph (e). Overall all the participants had some experience with JavaScript, with some very experienced participants.

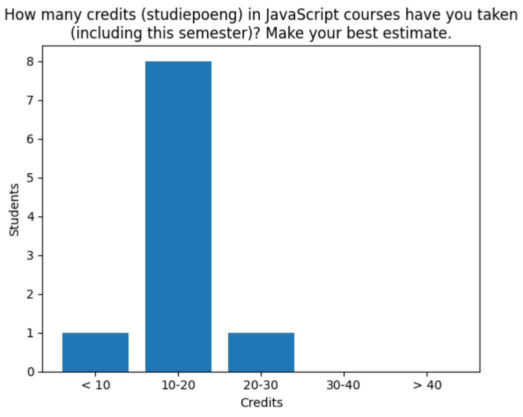
4.2.2 Learning outcomes results

The next part of the experiment focused on evaluating the learning outcomes of the participants relating to array functional methods. The participants were asked 12 multiple choice questions testing their understanding of what each functional array method does, when to use them and the correct syntax for using them. The number of correct answers for each participant is presented in Table 4.1. The table shows the number of correct answers on the learning outcomes questionnaire before utilizing the tool, the number of correct answers after using the tool, and the delta between these values. These values demonstrate how much improvement in learning outcomes each participant had from answering questions from the tool, as having used the tool was the only differentiating factor from the results before and after.

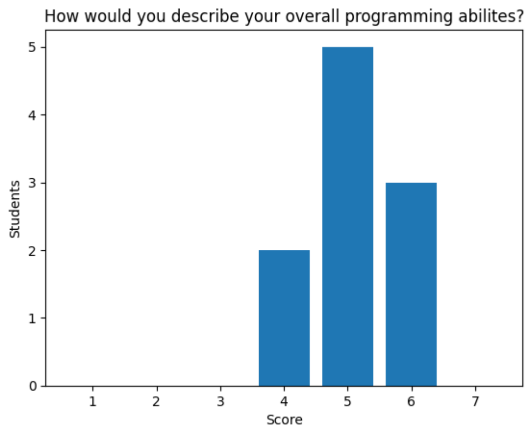
Measurements of the performance of the participants on the learning outcomes questionnaire are presented in Table 4.2. Before using the tool the participants had an average and median score both of 8 out of 12 questions. After using the tool the average score was 10.4 and the median score was 11, with an average improvement of 2.4 and a median improvement of 2. This translates to an average 30% and



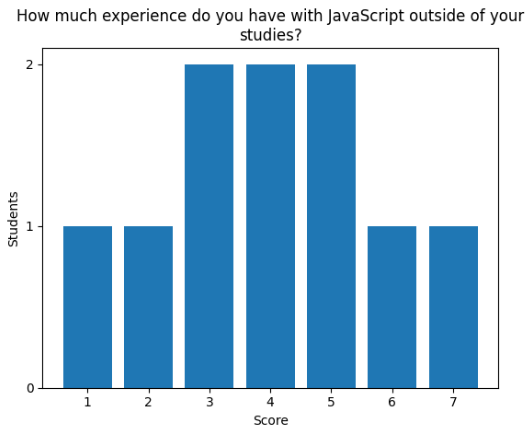
(a) Total programming credits.



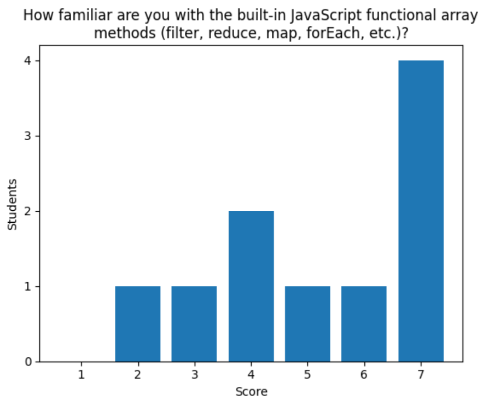
(b) JavaScript programming credits.



(c) Overall programming abilities.



(d) JavaScript experience outside studies.



(e) Experience with array functional methods.

Figure 4.17: Participant's previous programming experience.

ID	Correct before	Correct after	Δ
A	4	7	3
B	9	12	3
C	10	12	2
D	4	11	7
E	11	12	1
F	8	9	1
G	12	12	0
H	7	8	1
I	8	10	2
J	7	11	4

Table 4.1: Results before and after using the tool.

37.5% median performance improvement after using the tool. Only one participant answered all the questions correctly before using the tool, while four participants managed a full score afterward.

Measure	Value
Average score before	8
Median score before	8
Average score after	10.4
Median score after	11
Average improvement	2.4
Median improvement	2
Average percentage improvement	30%
Median percentage improvement	37.5%
Full score before	1
Full score after	4

Table 4.2: Measurements of learning outcomes questionnaire scores.

In the previous questionnaire, the participants were asked about their experience level. While this might give an indicator of their knowledge of the topic, it is possible that the participants might overestimate or underestimate their capabilities. It is therefore relevant to look at the correlation between the number of correct answers and the participant's stated experience. Table 4.3 shows a table with the number of correct answers on the learning outcomes questionnaire, together with the information about the participant programming experience from the previous part of the experiment. The data regarding programming credits and JavaScript credits are translated into a 5-point scale, translating under 10 credits to 1 and translating over 40 credits to 5, with the rest of the options in between. The rest of the experience values are expressed as the 7-point score shown in Figure 4.17, (c), (d) and (e). The ID anonymously signifies the different participants in the study.

ID	Correct LO	Total credits	JS credits	Coding exp.	JS exp.	AFM exp.
A	4	5	1	5	4	2
B	9	2	2	4	3	5
C	10	5	2	6	5	7
D	4	5	2	5	6	7
E	11	2	2	6	7	7
F	8	5	2	5	4	4
G	12	5	3	5	5	7
H	7	2	2	5	1	6
I	8	2	2	4	2	4
J	7	5	2	6	3	3

Table 4.3: Correct learning outcomes answers and programming experience table.

The correlations between previous experience and correct answers on the learning outcomes questions are shown in Table 4.4. The correlations are calculated using the Pearson correlation coefficient, with the R-values being presented in the table. The number of correct answers before using the tool has a moderate correlation with the number of JavaScript credits the participant had achieved and the stated previous experience using array functional methods in JavaScript. Total number of programming credits, overall programming abilities and experience with JavaScript outside of their studies had a weak or no correlation with the learning outcomes before using the tool. The same can be seen with the performance after using the tool, with the number of JavaScript credits, experience with array functional methods, and also experience with JavaScript outside of their studies had a moderate correlation with the number of correct answers after using the tool. The amount of improvement in the learning outcomes from before to after had weak or no correlation with any of the previous programming experience measurements of the participants.

Correlation	Before	After	Δ
Total programming credits	-0.2421	-0.0468	0.2782
JavaScript credits	0.7071	0.6412	-0.3516
Overall programming experience	0.1694	0.213	-0.03
JavaScript experience	0.2282	0.4967	0.1513
Array functional methods experience	0.5114	0.6195	-0.112

Table 4.4: Correlation between learning outcomes answers and previous experience.

4.2.3 Answering generated questions results

After the participants had answered the first questionnaire testing their current knowledge of the learning outcomes regarding array functional methods in JavaScript, they were asked to answer questions generated by the question generation tool for a 35-minute session. During this session, the participants used the tool to practice questions relating to the topic of array functional methods. Meanwhile, data was collected from the participants based on their usage of the tool. Data was collected based on four different actions: each time the participant submitted and answered a question correctly, each time a participant submitted and answered a question wrong, each time the participant skipped a question, and each time a participant loaded a new question. There were no limits on submitting solutions for a question, so the participants could submit an answer wrong multiple times for a single question. There was neither any limit on how many times a participant might skip a question. A new question was viewed each time a participant answered a question correctly and went to the next question when the participant skipped a question or if the participant changed the question category from standard to intermediate. Table 4.5 shows how many times each participant performed each of these actions.

Table 4.6 shows a table with the R-values from the Pearson correlation coefficient between the usage data for each participant with the performance on the learning outcomes questionnaire before and after using the tool, and the improvement from before to after. There are no strong correlations between the usage metric data and the performance on the learning outcomes questionnaire. This seems to be the case because of the participants used multiple different strategies when using the tool. Some participants view and submit as many solutions as possible, with some focusing on a few questions and using longer times on each question. The only moderate correlation with learning outcomes performance is the number of questions

ID	LO before	LO after	Correct	Wrong	Skip	Views
A	4	7	9	8	4	14
B	9	12	13	8	0	15
C	10	12	14	9	15	35
D	4	11	3	4	1	5
E	11	12	3	7	3	9
F	8	9	1	16	1	5
G	12	12	15	4	3	21
H	7	8	4	1	0	5
I	8	10	7	10	6	16
J	7	11	10	6	3	16

Table 4.5: Usage data from using the tool.

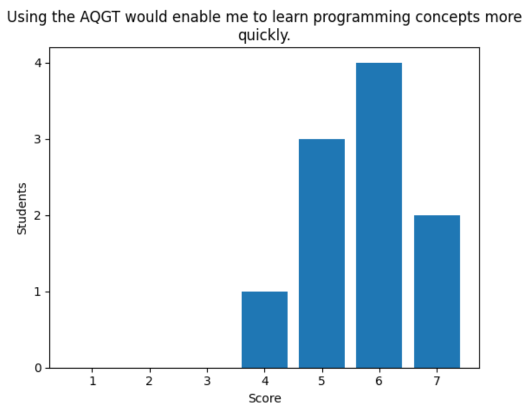
answered correctly and the number of questions viewed.

Correlation	Before	After	Δ
Correct	0.3949	0.4225	-0.1374
Wrong	0.0816	-0.077	-0.1786
Views	0.4469	0.4427	-0.1879
Skip	0.2635	0.254	-0.1173

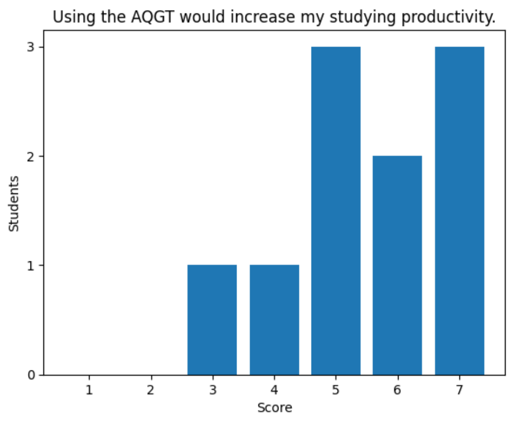
Table 4.6: Correlation between actions and learning outcomes performance.

4.2.4 Evaluation results

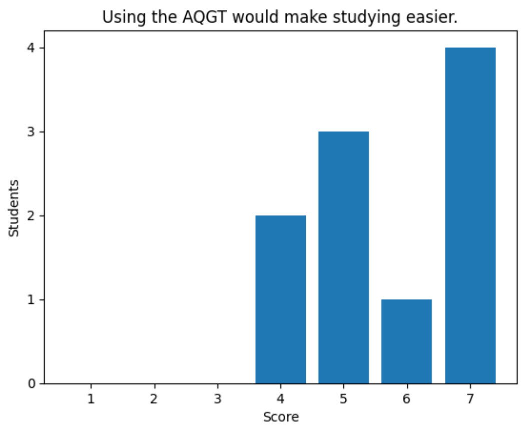
The last part of the experiment was an overall evaluation of the tool using the Technology Acceptance Model. This part consisted of a questionnaire with 10 questions based on the three categories in the model, the perceived usefulness of the tool, the perceived ease of use of the model and the user acceptance. At the end of the evaluation, there were also two open-ended questions asking about the learning experience of the tool and any suggestions for improvements. AQGT is used as an abbreviation in the questions, meaning Automatic Question Generation Tool.



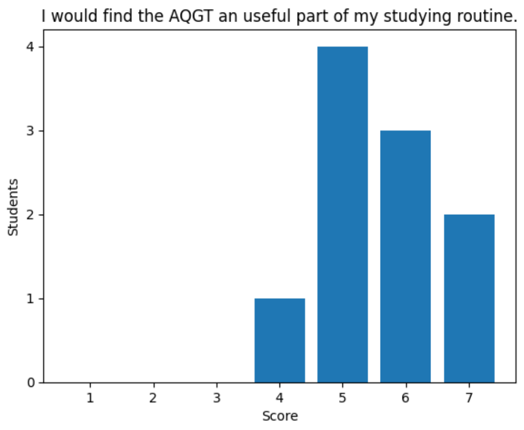
(a) Learning more quickly.



(b) Increase productivity.



(c) Make studying easier.



(d) Useful part of studying routine.

Figure 4.18: Perceived usefulness.

Quantitative results

The first four questions in the evaluation were based on the perceived usefulness of the tool, as seen in Figure 4.18. These questions focus on how the participants evaluate the tool to be useful in achieving a desired outcome. For this tool, this meant whether it would be useful for learning programming concepts more quickly, increase studying productivity, make studying easier and whether it would be a useful part of their studying routine. Table 4.7 shows tool scored quite high on all these measures with an average score of 5.7 for learning more quickly, a score of 5.5 for increasing productivity, a score of 5.5 for making studying easier and a score of 5.6 for being a useful part of their studying routine.

The next three questions were based on the perceived ease of use of the tool. This means how easy it was to learn to use the tool, how easy it was to make the tool do what the participant wanted it to do, and to what degree the interface was clear

Perceived usefulness questions	Score
Using the AQGT would enable me to learn programming concepts more quickly.	5.7
Using the AQGT would increase my studying productivity.	5.5
Using the AQGT would make studying easier.	5.7
I would find the AQGT an useful part of my studying routine.	5.6

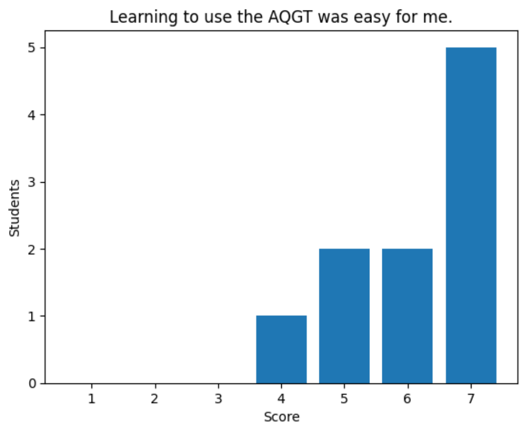
Table 4.7: Perceived usefulness average scores.

and understandable. When testing the tool, the participants received no tutorial or no information about how to use the tool, they were simply left with the user interface. The participants rated the ease of use very highly, with an average score of 6.1 for how easy the tool was to learn to use, a score of 5.9 for how easy it was to make the tool do what they wanted it to do and a score of 5.9 for how clear and understandable the interface was. These values are represented in Figure 4.19 and Table 4.8.

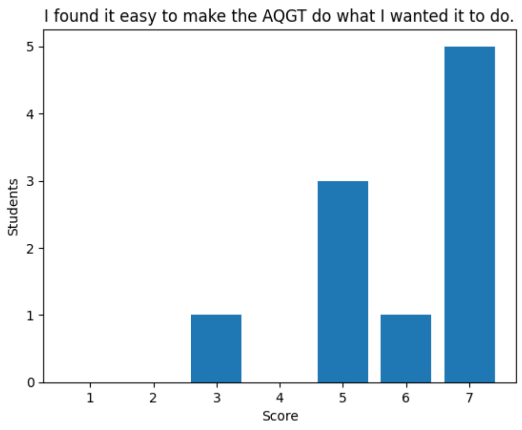
Perceived ease of use questions	Score
Learning to use the AQGT was easy for me.	6.1
I found it easy to make the AQGT do what I wanted it to do.	5.9
The user experience of the AQGT was clear and understandable.	5.9

Table 4.8: Perceived ease of use average scores.

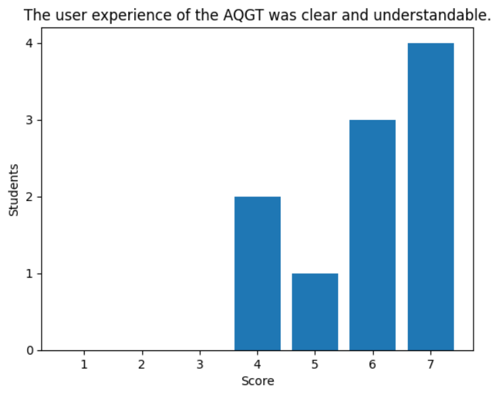
The last three questions in the evaluation focused on the user acceptance of the tool. These questions were based on how likely it is the participants actually would use the tool in a real-world setting, how likely it is they would recommend using it to others and how likely it is they would utilize it for a variety of different purposes. The evaluation for these questions is presented in Figure 4.20 and Table 4.9. These average scores were a bit lower than the questions for usefulness and ease of use, but this is to be accepted as the threshold for actually starting to use a new technology is higher than finding something useful or easy to use. The average score for how likely it is the participants would frequently use the tool is 4.9, the average score for likely it is they would recommend it to other students is 5.6 and the score for the likelihood for using the tool for a variety of different purposes is 5.9.



(a) Ease of learning to use.



(b) Ease of achieving desired action.

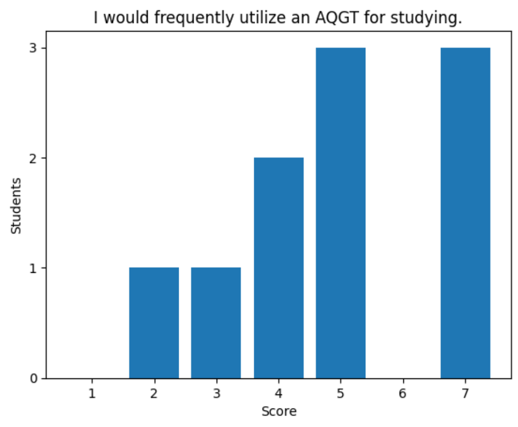


(c) Understandable user experience.

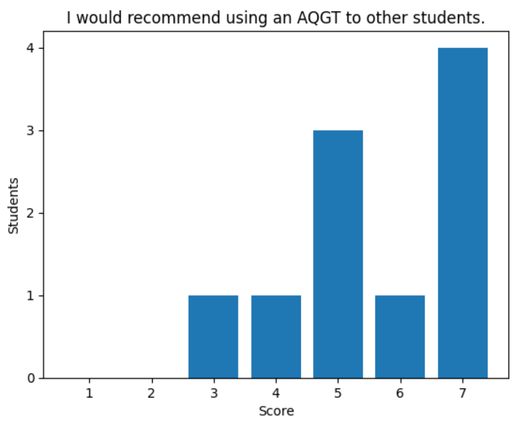
Figure 4.19: Perceived ease of use.

User acceptance questions	Score
I would frequently utilize an AQGT for studying.	4.9
I would recommend using an AQGT to other students.	5.6
I would use an AQGT for a varierity of purposes (preparing for an exam, learning a new concept, for repetion, etc.)	5.9

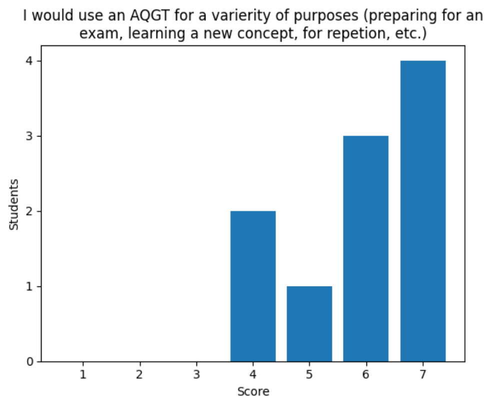
Table 4.9: User acceptance average scores.



(a) Would frequently use.



(b) Would recommend to others.



(c) Would use for a variety of purposes.

Figure 4.20: User acceptance.

Qualitative results

At the end of the evaluation, the participants answered two open-ended questions, ”How was the learning experience using the AQGT?” and ”Do you have any suggestions for improvements for the AQGT?”. For the first question, the overall feedback was very positive, with multiple participants remarking that they were able to learn very quickly using the tool and that it was surprisingly easy to use. There was also some constructive feedback, with some participants saying there could have been more variety in the questions, that the tool would be suitable for specific tasks but could not replace more complex assignments and that the tool would be better suited to evaluating current knowledge instead of being used as a learning tool.

The second question relating to suggestions for improvements also received a variety of answers. The most requested feature was the ability to see hints or view the solution to the question if the user is not able to solve it. This was due to some of the participants getting stuck on some of the questions, and without hints, there are no other way to proceed other than skipping the question. There were some questions where the participant needed to copy some text from the question into the code editor, which led to multiple participants suggesting having this code automatically embedded into the editor. A participant also requested to have a test case with the expected output.

Chapter 5

Discussion

This chapter contains a discussion focused on the methodology and results of both the question generation tool and the experiment. The first section covers the limitations, use-cases and why certain choices were made regarding the development of the tool. The second section contains a discussion of the results from the experiment, including covering the limitations of the experiment and the methodology.

5.1 Question generation tool discussion

This section is a discussion of the question generation functionality, automated assessment functionality, use-cases and the limitations of the question generation tool.

5.1.1 Question generation

An influence on the question generation capabilities for the tool was Rusak et al. [6] and their system for creating unique exams. A similar technique is used for generating the questions, but the functionality is greatly expanded on in multiple ways for this project. The method used by Rusak et al. is based on creating a question skeleton with a set of numeric variables. The possible numeric values are a set of numbers provided by the teacher, and are not randomly generated. The teacher then defines a formula for calculating the question solution based on the set of variables. The method produces a set of unique questions, but the questions need to be given to the students separately and there is no automatic assessment. Because of the limitation to only having numeric variables, the method is mainly limited to mathematical questions and is not that well suited for programming questions.

The first way the question generation tool differs from that of Rusak et al. is the variety of possible variables. While their tool supports integers, the tool developed for this project supports strings, integers, decimals, and lists. This greatly increases the range of variation possible. Especially the introduction of string variables makes the possibilities much greater, as it is possible to have multiple different paths in the question solution. Using combinations of strings, numbers, and lists also allows for greater variety, as opposed to only having numbers. Lists and strings are especially useful when creating coding questions, as the possible uses for strings and lists when coding are almost endless, including having many built-in functions.

Another difference is also that for Rusak et al. the teacher needs to manually define the possible numeric values, while for this project the teacher defines a valid interval and randomly picks a value from the interval. Using an interval allows for a much larger range of possible combinations, and requires less work from the teacher. A potential downside of using an interval is that it is a small possibility that some combination of unique variables might cause a compiling error and therefore an invalid solution. For example, if a set of randomly picked values causes the code to return an empty value or something that causes an error. This can be quite easily mitigated by checking if the generated questions produce a valid output solution, and removing the question if it does not.

The next novel contribution is the variation in question types. While Rusak et al. allows for one type of question skeleton format, the tool in this project allows for three types of question. This makes it possible to create more unique and specialized questions, with a special focus on programming questions. The question skeleton format from Rusak et al. is most similar to the coding question type for this project. This similarity comes from the fact that both require the user to define some variables for a question and then a function or formula that provides a solution based on the variables. This means that this question type also could be used to create math or statistics-type questions in addition to coding questions. The two other question types, the understanding code, and the produce output type questions are specifically created with coding in mind and are therefore not that transferable to question generation for other subjects.

5.1.2 Automated assessment

Automated assessment when answering questions is an important feature of the tool, bringing both benefits and some limitations. Because the main use for this question generation tool is as a self-study aid, automated assessment is almost a necessity. Without automated assessment the submissions would either have to be manually graded or the questions would have to be open-ended without a defined answer. Grading the questions manually would be very time-consuming and would eliminate one of the major benefits of the tool, which is to save resources and time when creating questions for a class. Having open-ended questions would not be a good solution either, as getting feedback on assessments is essential for the students to learn whether their way of understanding a topic is correct or wrong.

These benefits of automated assessment also come with some downsides. A downside is a limitation in the question variance. Because the system needs to be able to check the correctness of the submission against a solution, the solution can only be a single output. This means that the question needs to be designed in such a way that the solution can be calculated using a formula. It must always be possible to calculate the solution given a set of variables. This limits the variance of questions to those that can be evaluated by a single output solution. While this still allows for a large amount of variance, it makes it hard to create more open-ended questions. This automated assessment method is similar to that of Bruzual et al. [12]. Their system is based on compiling and running the submitted code, then checking if the submission produces the correct output using test cases. The assessment method for this project works similarly, by compiling and executing the submission cloud, then checking if the code output is equal to the solution.

5.1.3 Use cases

As mentioned previously, the main use case for the tool creating in this project is as a self-study learning tool to increase students learning outcomes, specifically aimed at programming courses. For this use case, the tool can be very helpful in making it possible for students to solve a much larger quantity of programming questions than they would normally have the opportunity to. Usually, a teacher will manually create a set of assignments or quizzes for a course, sometimes with a couple of voluntary exercises for the students to get some extra practice. The amount of assignments and exercises that can be created is limited by the time available for the teacher to create high-quality questions. When interviewing a university programming teacher for this project, feedback was received that students in his course were asking for more practice questions for certain topics. In this case, a question generation tool would be really helpful, as it would allow the teacher to generate a large set of questions for the students to practice without using too much time.

While the main use case for the question generation tool is a self-study aid, it would also be possible to use it for assessments. This was explored in the specialization project, but after some discussions with the university teachers, it was decided that a self-study aid was the more useful use case. This was for a couple of reasons. Firstly, a major reason for using a question generation tool for an exam or assessment is to mitigate cheating by giving students unique questions with unique solutions. This was a big problem during COVID-19 with the use of home exams, but now as most exams have transitioned back to on-site exams, this is less of an issue. Another challenge is creating fair questions, meaning that all the generated questions have a similar difficulty. This is very important for exams and assessments to make sure that all the students get a fair evaluation, and that some students get a harder exam than others. Despite these difficulties, the question generation tool could easily be utilized to generate exam questions as well as be used as a self-study tool. The teacher responsible would need to be extra precise when creating the questions to ensure fairness, and preferably use some evaluation metric, such as the method described by McCoubrie et al. [11] to ensure equal question difficulty. In addition to exam questions, the tool could also be used to generate graded assignments, with the same precautions in mind as when generating exam questions.

5.1.4 Limitations

A limitation of the system was briefly described above, which is the limitation in question variance due to the need of having a single output question solution. This makes the question generation tool best suited for questions where the solution can be calculated to a string, list, integer or some sort of code output. It is not well suited to open-ended questions and more complex applications, as these systems would be too complicated to evaluate accurately. Even with this limitation, it is possible to create some very complex questions. The only limitation on the complexity of the questions is that it must be possible to calculate the correct solution given the question variables.

Another limitation of the automated assessment system is that the evaluation is only based on the final output of the submission and not any of the code preceding the solution. This means that the system only evaluates the final value, but not the approach the student used to find the solution. This means that the student is able to get the correct answer even if using the wrong approach. With manual evaluations, this can be avoided by the teacher or teaching assistant reading the code and giving feedback to the student that the approach is wrong. This is much harder to achieve automatically, as the system would need to know what the correct approach is, then read and understand the submission code, then be able to differentiate a bad approach from a good approach. The current implementation, therefore, is only focused on evaluating the final solution, and the teacher can rather give hints in the question description on what approach the student should use. With this system, it is also possible for students to cheat to get a correct evaluation by simply outputting the correct solution. The system has some simple protection against this by checking if the code contains a print line containing only the solution string, but it is quite simple to bypass this if the student is creative. While this would a big problem if the intended use for the automated assessment system was to assess exams or graded assignments, it is not a problem when the intended use of the system is a self-study tool, and little time was therefore allotted to preventing this. The goal of using a self-study tool is to improve your own learning outcomes, and there is therefore little motivation for students to cheat.

There are also some limitations when it comes to getting feedback in the question generation tool. When the user compiles and submits their code, the user can get feedback in two different ways. The first way is if their code produces an error when compiling, the error message will be displayed in the user interface so the student can fix it. If the submitted code does not produce an error, it will display the code output. Secondly, if the students submit their code the tool will evaluate whether the output of the code is correct or wrong. The limitation of this system is that if the student submits their code and the answer is wrong, it is not possible to get feedback on what part of the code is wrong. This can make it hard to proceed with the question, as the student can struggle to find out what is wrong. With manual evaluation of assignments, the student can get written feedback from the teacher or teaching assistant on what is wrong with the solution and what they need to work on, which is not possible with the automated assessment system. This is due to the same reason that the system can not evaluate the solution code approach, it requires that the computer has a complex understanding of the submission and the question and is able to articulate a feedback message. A potential remedy to this, which was suggested by multiple students in the evaluation feedback in the experiment, is to be able to view the code solution or to get predefined hints if the student answers wrong multiple times. This will make it possible to find out what was wrong with their solution.

5.2 Experiment discussion

This section contains a discussion of the experiment, including the methodology used, the design of the experiment, how the experiment being held digitally had an effect, a discussion of the evaluation results and finally the limitations of the experiment.

5.2.1 Methodology

Participants

As shown in the results section, the participants in the experiment came from a range of different study programmes. This included Computer Science, Informatics, Cyber Security and Information Technology. The participants were also from different years of study, some first-year students, some second-year students and some fifth-year students. This mix of study programmes and years of study made it difficult to design some parts of the experiment, as the participants had such diverse backgrounds. While all the study programmes are technology related, there is still a big difference in the amount of programming in each program. There was also no way of knowing how much, if any, experience the participants had using array functional methods beforehand.

This made it quite difficult to design the questions evaluating the learning outcomes. The questions needed to be easy enough for a participant with no prior experience to be able to learn to learn the concept in the 35-minute session using the tool, and at the same time not be so easy that an experienced participant might get all the questions correct before even using the tool. The learning outcomes questionnaire was therefore designed with 10 standard questions and 2 hard questions. The 10 standard questions related to the use of, the syntax of, and understanding of the basic array functional methods. They were designed such that a programmer with some previous experience with the topic would get a decent amount of questions correctly, but also such that a programmer with no previous experience could learn most of them in a quite short time. The last 2 questions were focused on some more niche functional methods that are not commonly used, and some more complex combinations of multiple functional methods. These questions were designed such that even a programmer with quite a lot of experience with the topic would struggle with answering them correctly in the pretest.

Digital experiment

Originally the plan was to carry out the experiment at a physical location on the university campus, but this was later changed to arranging it digitally over Google Meet. When recruiting the participants they were informed that the plan was to arrange the experiment physically. As participants started to volunteer, feedback was received from some students who wished to partake in the experiment, but were unable to attend physically due to not being in the city at the given time. The plan was to exclude these participants from the experiment and only keep the participants who could attend physically, but after a while the participants who could only attend digitally started making up a substantial portion of the participant group. To ensure that the experiment would have enough participants it was decided to also include digital participants. This led there to be two options: arranging a hybrid experiment with some digital and some physical participants, or a fully digital experiment. Due to the digital portion of the participant group being substantial and to ensure continuity across the experiment, the fully digital version was preferred over the hybrid version.

There were no real problems converting the experiment to being digital, as all the parts of the experiment were performed on a computer anyway, including answering the questionnaires and testing the question generation tool via the website. While performing the experiment was no problem logistically, there would have been some benefits to performing it at a physical location. Firstly, doing the experiment physically would allow for greater control that all the participants are actually doing what they are supposed to do. It is possible to see the screens of the participants and make sure that they are not checking other websites during the experiment or using other applications, and that they are fully utilizing the allotted time. This is not possible when performing the experiment digitally, as it is not possible to see what the participants are doing on their screens. This requires trusting that the participants are fully focused on the experiment and not doing other things simultaneously.

This problem was mitigated in a couple of ways when performing the experiment digitally. All the participants were required to have their cameras turned on during the experiment. This made it possible to make sure that all the participants were present at the computer during the entirety of the experiment. While this does not guarantee that they are not using the computer for other purposes than partaking in the experiment, it does guarantee they are present. Another way to control that the participant was doing the experiment was by collecting usage data when the participants were using the tool. This made it possible to see that each participant actually viewed questions, submitted and compiled solutions using the tool.

Another potential problem with arranging the experiment digitally is the potential for cheating on the learning outcomes questions. The participants were told to not use any external sources when answering the learning outcomes evaluation, but there was no way of validating this. If the experiment was performed at a physical location it would be easier to monitor that the participants did not use external sources. While it is possible to cheat, there would be no real incentives to as there was nothing to gain by answering more questions correctly.

Experiment design

As described previously, the experiment was designed with four main parts, each part having a unique goal. The experiment was meant to answer how effective a question generation tool is as a self-study aid to improve learning outcomes for JavaScript, and if the programming experience level of the participants affects this. The goal of the first part of the experiment was therefore to collect some data to determine the participant experience level. It was decided to collect data about their overall programming experience, their JavaScript experience, and their array functional methods experience. This was split into different parts because having a lot of programming experience does not necessarily mean being experienced with JavaScript and array functional methods. It is therefore possible for a participant to be a very experienced programmer, but still perform poorly on the learning outcomes evaluation. The programming experience was measured using credits taken at university and self-evaluated experience using a 7-point Likert scale.

The goal of the next part of the experiment was to have some sort of measurement of learning outcomes before and after using the question generation tool. Because the main goal of the question generation tool is to improve learning outcomes there needed to be some way of measuring an improvement. A multiple-choice questionnaire is a standard method of measuring learning outcomes and was a natural choice. The advantage of using a pretest-postest design experiment to measure improvements in learning outcomes is that there is a direct link between the usage of the tool and the improvement. The learning outcomes of each participant are measured right before testing the question generation tool and then right after. This means using the tool is the only influence on the participant between the two evaluations and can therefore be directly linked to the potential improvement. An observational study where the participants would have used the tool over a longer timespan and measured learning outcomes is not as direct, as there are many other factors that could influence the learning outcomes, which is a limitation of the study by Tsai et al. [10].

The third part of the experiment was for the participants to actually answer questions generated by the question generation tool. The participants had a 35-minute session to use the tool to answer questions. The amount of time for using the tool was limited by not wanting the experiment to run too long, so it would be relevant for another experiment to see if using the tool for a longer session would lead to even better learning outcomes. The goal of this part was for the participants to use the tool to increase their learning outcomes regarding array functional methods, in addition to testing how the tool would be used in practice. The participants were given no information about how to use the tool or how it worked, so the testing was also a test of the user interface and whether it would be clear and understandable to someone who had never used it before.

The goal of the last part of the experiment was to get an overall evaluation of the tool from the participants, in addition to the results from the learning outcomes improvement. This evaluation is done using questions based on the Technology Acceptance Mode, looking at perceived usefulness, perceived ease of use and user acceptance. This data is highly relevant because it says something about if the participants actually found it useful and whether they would actually use it. This is crucial because there is no point in creating a tool that improves learning outcomes if nobody wants to use it, or knows how to use it. The end of the questionnaire also allowed the participants to give a longer text feedback of their experience of using the tool or if they had some suggestions for improvements. This was added at the end in case there were some aspects that were not covered in the previous sections, or there was something the participants wanted to add. This was very useful as many of the participants wrote long and detailed feedback on the experience and suggestions for improvements, and provided valuable feedback that would not have been captured by the evaluation questions.

5.2.2 Results

Participant experience data discussion

Figure 4.16 shows the distribution of participants across study programmes and years of study. There is quite a bit of variation within both categories with there being students from five different study programmes and across three different years of study. This variation in students was desired as it would allow to compare the results between less and more experienced programmers. Because of the topic of the experiment and the focus of the question generation tool mainly being JavaScript, it was required that all the participants had some JavaScript experience from university. The first and second-year students were therefore recruited from two different JavaScript courses and the fifth-year students were Computer Science or Informatics students who all had obligatory JavaScript courses as part of their study plan.

Figure 4.17 contains the stated previous experience of the participants. The results from graph (a) and (b) shows the number of programming credits, both in total and for specifically JavaScript courses. This data was quite expected, with the first-year students having 10-20 credits in programming and the second and fifth year having over 40 credits. In hindsight, there could have been more options above 40 credits, as such a large portion of participants fit into this category. There were not any surprises with the number of JavaScript credits either, with most having 10-20 credits, translating to around two courses. Graph (c) shows how the participants would rate their own programming abilities, showing all the participants rating themselves as average or above average. This means that there were no programming beginners among the participants. Optimally it would have been better to have even more variation between participants, with more inexperienced programmers. Experience with JavaScript outside of university is displayed in graph (d) and is a normal distribution. Most participants had some experience, with some having no experience and some being very experienced.

The last graph (e) was the most surprising, showing the experience level of the participants regarding array functional methods in JavaScript. It was not expected that such a large portion of the participants would describe themselves as very experienced in using array functional method. Most of the questions evaluating the learning outcomes were designed for programmers with some, but not a lot of experience with array functional methods. If this was known before designing the learning outcomes questionnaire, the questions would have been designed to be a bit harder. While this is the experience level stated by the participants, this does not necessarily translate into actual abilities. It is possible the participants either overestimate or underestimate their own abilities.

Learning outcomes discussion

Table 4.1 shows the scores on the learning outcomes questionnaire before and after using the tool. Overall there was a significant improvement from the pretest to the posttest results, with an average score of 8 before using the tool and an average score of 10.4 after using the tool, meaning an average 30% improvement. It was unexpected that the participants would score so highly on the test before using the tool, as the participants had more experience with array functional methods than was expected when creating the learning outcomes questions. The high average score of the participants before using the tool meant that there was less possible room for improvement, as there were only 12 points available in total. If the test questions were designed to be harder, there might have been an even larger improvement from before to after due to the participants getting fewer questions correct in the pretest.

A point to consider when evaluating the results is that the participants had already seen the questions once when answering the posttest. This means that the participants had the possibility to quickly answer the questions they had already solved from the pretest. The participants therefore had a time advantage when answering the posttest questionnaire, even though the allotted time was the same. For this reason, it would have been useful to have a control group that was not exposed to the question generation tool. Then it would be possible to see if some of the improvement was due to it being the second time answering the test, or if the improvement was only due to exposure to the tool.

Table 4.4 shows the correlations between pretest scores, posttest scores and the delta between them with the various measures of previous programming experience including the total amount of programming credits, the total amount of JavaScript credits, overall programming abilities, experience with JavaScript outside of studies and experience with array functional methods. The score on the pretest has a moderate correlation with the number of JavaScript credits of the participants and the stated experience with array functional methods, with a weak or no correlation with the other attributes. It makes sense that only JavaScript credits and array functional methods experience correlates with the score on the pretest, as having overall programming experience does not mean that the participant knows a lot about array functional methods. While there is a correlation, it is only a moderate correlation. This shows that the stated experience of the participants is not that accurate in predicting the actual performance on the test. The results in the posttest also has a moderate correlation with JavaScript credits and with array functional method experience. This also makes sense for the posttest, as having more previous experience with JavaScript and array functional methods would result in a higher score in the posttest as well.

When looking at the delta improvements, there are no moderate or significant correlations between the delta and the previous experience of the participants. This means that both the experienced and the inexperienced programmers had a similar improvement in performance after using the tool. It might be expected that a programmer with less experience would have larger room for improvements, and therefore be correlated with a higher delta, but this is not the case. It could also be argued that a more experienced programmer is able to learn new concepts more quickly, and would thereby have a larger delta, but this also not the case.

Answering generated questions discussion

The exposure to the question generation tool consisted of the participants freely answering questions using the tool for 35 minutes. Ideally, the participants would have had more time to test the tool, but the time was limited by the total length of the experiment. The participants were given no information about how to use the tool, which meant the user interface was the only element for the participants to learn how to use the tool. The participants were told to not use the internet or any external sources during the testing period, with the exception of website containing documentation for the array functional methods [17]. This was to ensure that all the participants had access to the same information to solve the questions and that the participants would not find a solution online instead of writing it themselves. The execution of the testing went pretty well, with only a few issues arising. At the start of the experiment, some of the participants had an issue with loading the questions, but this was quickly solved by opening the tool in another browser. There was also a problem with using TypeScript instead of JavaScript on some of the questions, but this was also resolved. The participants were all able to understand how to use the tool without instructions.

Table 4.6 shows the correlations between the pretest scores, the posttest scores and the delta between with the number of correct answers, wrong answers, question views and questions skips. This shows that there is a weak to moderate correlation between the number of correct submissions and the number of question views with participant score before and after. This makes sense, as the participants who scored highly on the learning outcomes would be likely to have more knowledge of array functional methods, and thereby answer more questions correctly when using the tool. The figure also shows that there is no correlation between the number of correct or wrong questions, skips or views, with the delta improvement from the pretest to posttest. This means that answering more questions correctly or wrongly did not correlate with a larger or lesser improvement in performance on the posttest. Table 4.5 shows that the participants used different strategies when using the tool. Some participants answered as many questions as possible and some used their time to focus on a few questions. There were also some participants who skipped and viewed a large number of questions, while others skipped very few questions. In addition to this, there were also a a participant who struggled and managed to answer only one question.

Evaluation discussion

The participants evaluated the perceived usefulness and the perceived ease of use of the tool very highly overall. The questions were created based on standard questions used in Technology Acceptance Model, specifically based on some of the questions used in Lewis et al. [18] to evaluate perceived usefulness and perceived ease of use. The participants rated the tool very highly with regards to being helpful for learning programming concepts, increasing studying productivity and making studying easier. The scores for the perceived ease of use were also very high, with the participants finding the tool very easy to learn to use, easy to make the tool to what it was supposed to do and rated the experience of using the tool as clear and understandable. There was one participant who struggled with some parts of the user interface, as seen in Figure 4.19 (b), but overall the perceived ease of use was very high. In the Technology Acceptance Model perceived usefulness and perceived ease of use were found to be highly correlated with user acceptance. Due to the question generation tool scoring high on these attributes, it would be expected that tool also would score highly on user acceptance.

The user acceptance was also measured in the evaluation, as seen in Figure 4.20. As expected, the tool also scored highly on the questions relating to user acceptance. While the scores were high, they were not quite as high as the scores for perceived usefulness and perceived ease of use. This was especially the case when asking whether the participants would frequently utilize the question generation tool for studying, with an average score of 4.9. It is not surprising that the user acceptance scores would be a bit lower than the perceived usefulness and perceived ease of use scores, as the threshold for actually starting to utilize a new tool frequently is quite high. The score could also have been influenced by the wording of the question, especially by the term "frequently". Participants could want to use the tool, but not necessarily all the time, and therefore give a lower score.

The open-ended questions resulted in mostly very positive feedback, describing the learning experience as very effective and easy to use. One of the participants remarked that the tool would be better suited to evaluations rather than as a learning tool. This was because the participant argued that it was only possible to answer the questions in the tool if the participant already had the knowledge to answer the question. While this is true to some extent, as the participant needs to acquire the knowledge necessary to answer the question from an external source. The question generation tool does not have documentation or other learning tools built in, so the user needs to acquire the relevant information to answer the question from another source. But this is also true when solving university assignments or any other type of questions, the student needs to read about the topic in a course book or an online source before being able to solve the question. The question generation tool would therefore still be useful as a learning tool when used in combination with external information sources.

The most requested functionality from the evaluation was the ability to see hints if the user is stuck on a question, or being able to view the solution if the user is not able to solve it. This would be a good addition to the tool, as it is possible for users to get stuck on a question. The user only receives feedback on whether the solution is correct, wrong, or if there is a compile error. If the submission is wrong, it can be difficult to know what to change to get the correct answer. This was an issue for one of the participants during the experiment, where the participant was stuck on a question for a long time and could not figure out how to proceed, which was commented by the participant in the evaluation. Another useful suggestion was the possibility of having test cases with expected outputs. This would achieve some of the same functionality as having hints, as the user would be able to see what all the test cases for the given question. These test cases can thereby give some information about what the user needs to do to get the correct answer.

5.2.3 Research questions

The first research question was "How effective is question generation in improving learning outcomes for JavaScript programming courses?". This was explored in this project through the development of a question generation tool and evaluation of the tool through an experiment. The experiment consisted of using the question generation tool to generate JavaScript programming questions on the topic of array functional methods, then having participants answer a test measuring their learning outcomes before and after being exposed to the tool. The exposure to the tool resulted in an average 30% improvement in learning outcomes after a 35-minute session using the tool.

The second research question was "Is question generation more effective in improving learning outcomes for beginner programmers than intermediate?". In the experiment participants were asked to provide information about their previous programming credits, including the total amount of programming credits taken at university, JavaScript course credits, overall programming abilities, experience with JavaScript outside of their studies and their experience with array functional methods. It was found that the amount of JavaScript credits and the experience with array functional methods correlated moderately with learning outcomes before and after using the tool, but there was no correlations between the participant experience and the size of the improvement on the learning outcomes tests. This means that for this experiment there was no difference found in effectiveness in increasing learning outcomes for beginner programmers than intermediate.

These findings are based on a very limited sample size of 10 participants, with only using the tool for a short session of 35 minutes. It would therefore be relevant to repeat the experiment with a larger sample size and longer time usage and see if the findings are the same.

5.2.4 Limitations

There are some limitations to this experiment. There was no control group when performing the experiment, so all the participants performed the same tasks. This means there is no alternative to compare the question generation tool against. For further work the experiment could be repeated, but with one group using the question generation tool for a given time period and the other group reading documentation or a textbook for the same time period, then compare the learning outcome improvements. This would give a more accurate representation of the utility of the tool, as it would be contrasted with other comparable studying methods.

Another limitation is the sample size with only 10 participants. Preferably the question generation tool should have been tested on an entire class of students to replicate how the tool would be used in practice. The time the participants had to test the was also a limitation, as they had only 35 minutes to test the tool. This was limited by not wanting to make the experiment take too much time, which could dissuade students from wanting to participate. Using the tool for only 35 minutes is quite a short time to learn a new topic and is not that representative of how the tool would be used in practice.

A limitation mentioned previously is the fact the experiment was conducted digitally instead of at a physical location. This made it hard to monitor whether all the participants did what they were supposed to do, used all the allotted time, or used external sources to cheat on the questionnaires. This also made it more challenging to ask for assistance or to clarify something during the experiment.

Chapter 6

Conclusion and further work

6.1 Conclusion

Automatic question generation has many potential uses, one of which is being used as a learning tool. A question generation tool can be used to automatically generate a large number of questions that students can use to learn a topic. This can save teachers a lot of time and resources by reducing the time needed for creating and grading questions, and it ensures that students can have access to a continuous supply of new questions. The goal of this project was to develop a new question generation tool specifically made for JavaScript programming courses and to evaluate its effectiveness as a learning tool.

The research questions for this project were "How effective is question generation in improving learning outcomes for JavaScript programming courses?" and "Is question generation more effective in improving learning outcomes for beginner programmers than intermediate?". The findings were based on a limited sample size of 10 university students with varying programming experience. The results from the experiment was that using the question generation tool for a 35-minute session led to an average 30% improvement in learning outcomes for a JavaScript programming topic. When looking at the correlations between the previous programming experience of the participants and their improvement in learning outcomes, there was no correlation found between the size of the improvement and their experience level. This means the tool had the same effectiveness for both the beginner and intermediate programmer participants.

6.2 Further work

The work done for this project can be expanded on by both expanding and improving the functionality of the question generation tool and by doing further experiments evaluating the tool with different methodologies. While the question generation tool developed for this project is a final prototype and could be used in practice, there is still much room for improvement. A limitation of the automated assessment functionality is that it only looks at the final output, and not at the approach used in the code. This is an aspect of the tool that could be improved by developing a test case suite, or by implementing some code analysis functionality. There are also possibilities for expanding the question generation variance using machine learning, for example by combining the current methods with semantic-based question generation, such as Yao et al. [9]. Functionality suggestions were also received during the evaluation. Being able to get hints if the user is stuck on a question was suggested by multiple participants, in addition to being able to view the solution. Another suggestion was building documentation into the question answering so that the user can view documentation relevant to the question without needing to find it on an external site. These functionality suggestions would be highly useful for the user experience of the tool if implemented.

As for further work evaluating the effect on improving learning outcomes there are many ways this could be improved. The first suggestion would be to perform the same, or a similar, experiment with a control group. Performing the experiment again with a control group not exposed to the tool would make it possible to distinguish if the improvement in learning outcomes were due to exposure to the tool or if some of the effect was due to it being the second time answering the questionnaire. It would also be interesting to have one group of students use the question generation tool to practice a subject, and another group read a textbook about the subject and see if one group have better learning outcomes. It would also be relevant to repeat the experiment with some changes to the methodology, such as a larger sample size, more time allotted to testing the tool, participants from non-technical study programmes and arranging the experiment physically instead of digitally. In addition to these controlled experiments it would be interesting to test the tool in practice by making it available as a study tool in a programming course at university and evaluating if it can improve student performance in a real world setting.

References

[1] K. A.-M. Sarpong, J. K. Arthur and P. Y. O. Amoako, 'Causes of failure of students in computer programming courses: The teacher-learner perspective', *International Journal of Computer Applications*, vol. 77, no. 12, 2013.

[2] S. H. Kang, 'Spaced repetition promotes efficient and effective learning: Policy implications for instruction', *Policy Insights from the Behavioral and Brain Sciences*, vol. 3, no. 1, pp. 12–19, 2016.

[3] R. F. Bruner, 'Repetition is the first principle of all learning', *Available at SSRN 224340*, 2001.

[4] G. Kurdi, J. Leo, B. Parsia, U. Sattler and S. Al-Emari, 'A systematic review of automatic question generation for educational purposes', *International Journal of Artificial Intelligence in Education*, vol. 30, no. 1, pp. 121–204, 2020.

[5] D. J. Danielsen and S. S. Fallmyr. 'Rekordmange blir tatt for fusking – savner informasjon om konsekvensene'. (2022), [Online]. Available: `https://www.nrk.no/nordland/rekordmange-studenter-blir-tatt-for-juksing-pa-eksamen-1.15886579` (visited on 6th Dec. 2022).

[6] G. Rusak and L. Yan, 'Unique exams: Designing assessments for integrity and fairness', in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 1170–1176.

[7] G. Sindre, 'Kan fusk på hjemmeeksamen forhindres?', *Nordic Journal of STEM Education*, vol. 5, no. 1, 2021.

[8] D. D. Dixson and F. C. Worrell, 'Formative and summative assessment in the classroom', *Theory into practice*, vol. 55, no. 2, pp. 153–159, 2016.

[9] X. Yao, G. Bouma and Y. Zhang, 'Semantics-based question generation and implementation', *Dialogue & Discourse*, vol. 3, no. 2, pp. 11–42, 2012.

[10] D. C. Tsai, A. Y. Huang, O. H. Lu and S. J. Yang, ‘Automatic question generation for repeated testing to improve student learning outcome’, in *2021 International Conference on Advanced Learning Technologies (ICALT)*, IEEE, 2021, pp. 339–341.

[11] P. McCoubrie, ‘Improving the fairness of multiple-choice questions: A literature review’, *Medical teacher*, vol. 26, no. 8, pp. 709–712, 2004.

[12] D. Bruzual, M. L. Montoya Freire and M. Di Francesco, ‘Automated assessment of android exercises with cloud-native technologies’, in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 40–46.

[13] A. R. Hevner, S. T. March, J. Park and S. Ram, ‘Design science in information systems research’, *MIS quarterly*, pp. 75–105, 2004.

[14] H. Z. Došilović and I. Mekterović, ‘Robust and scalable online code execution system’, in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, IEEE, 2020, pp. 1627–1632.

[15] D. M. Dimitrov and P. D. Rumrill Jr, ‘Pretest-posttest designs and measurement of change’, *Work*, vol. 20, no. 2, pp. 159–165, 2003.

[16] F. D. Davis, ‘Perceived usefulness, perceived ease of use, and user acceptance of information technology’, *MIS quarterly*, pp. 319–340, 1989.

[17] *Javascript array documentation*, `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array`, Accessed: 2023-06-11.

[18] J. R. Lewis, ‘Comparison of four tam item formats: Effect of response option labels and order.’, *Journal of Usability Studies*, vol. 14, no. 4, 2019.