INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Automatic Question Generation about Introductory Programming Code

Tiago Filipe Martinho Soares

Master in Computer Engineering

Supervisor:
André Leal Santos, Assistant Professor
Iscte - Instituto Universitário de Lisboa

Co-Supervisor:
Nuno Miguel de Figueiredo Garrido, Assistant Professor
Iscte - Instituto Universitário de Lisboa

September, 2021

TECHNOLOGY
AND ARCHITECTURE

Department of Information Science and Technology

Automatic Question Generation about Introductory Programming Code

Tiago Filipe Martinho Soares

Master in Computer Engineering

Supervisor:
André Leal Santos, Assistant Professor
Iscte - Instituto Universitário de Lisboa

Co-Supervisor:
Nuno Miguel de Figueiredo Garrido, Assistant Professor
Iscte - Instituto Universitário de Lisboa

September, 2021

For the women of my life, my mother, my grandmother and my life partner Cláudia.

For Him, who created all things.

Acknowledgements

I would like to thank my supervisors, Prof. André Santos and Prof. Nuno Garrido for their support on this project. I would like to thank Prof. André Santos especially, for his mentoring, guidance and all the time he has dedicated to help me improve this dissertation. Without him, this goal could not be achieved.

To my mother and my late grandmother, a special thank you for raising me and giving me everything you could, despite all the hardships.

To my brother, thank you for always helping me getting out of trouble and being my father figure when I was younger, and for always being so happy with my accomplishments.

To my soon-to-be wife, Cláudia, thank you for your continuous support and love, throughout all these years.

And finally, thank you God for inspiring me every day.

Resumo

Muitos alunos que aprendem a programar acabam por escrever código que não entendem. A maior parte dos sistemas de avaliação de código disponíveis avaliam a solução submetida funcionalmente e não o conhecimento da pessoa que o submeteu. Esta dissertação propõe um sistema que gera questões sobre o código submetido pelo aluno, analisa as suas respostas e devolve as respostas corretas. Desta forma, os alunos refletem sobre o código que escreveram e os professores das unidades curriculares de programação conseguem identificar melhor as suas dificuldades.

Conduzimos uma experiência com alunos de licenciatura e mestrado em Engenharia Informática e cursos relacionados de forma a perceber quais as suas dificuldades e testar a robustez do protótipo. Concluímos que a maior parte dos alunos, embora entendam detalhes simples do código que escrevem, não entendem o comportamento do programa na sua totalidade e o estado que este possui num determinado momento. São também sugeridas melhorias ao protótipo e à condução de futuras experiências.

Palavras-chave: Introdução à programação, ferramenta pedagógica, sistemas de tutoria inteligentes.

Abstract

Many students who learn to program end up writing code they do not understand. Most of the available code evaluation systems evaluate the submitted solution functionally and not the knowledge of the person who submitted it. This dissertation proposes a system that generates questions about the code submitted by the student, analyses their answers and returns the correct answers. In this way, students reflect about the code they have written and the teachers of the programming courses can better pinpoint their difficulties.

We carried out an experiment with undergraduate and master's students in Computer Science degrees in order to understand their difficulties and test the prototype's robustness. We concluded that most students, although understanding simple details of the code they write, do not understand the behaviour of the program entirely, especially with respect to program state. Improvements to the prototype and how to conduct future experiments are also suggested.

Keywords: Introductory programming, pedagogical tool, intelligent tutoring systems.

Contents

Code Listings

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1. Motivation

Students that aspire to become professional developers or software engineers are faced with multiple obstacles when learning the first steps and intricacies of programming. While students can write code that works, it does not mean that they understand it, at least entirely. However, any technique focused on artifact analysis assumes that students understand the code they use in their projects. As noted by Brennan [1], this is not necessarily true. This means that analysing code solutions submitted by a student for a given problem is not enough to measure his or her programming knowledge.

Salac and Franklin [2] propose that written assessments or interviews are necessary to find out whether students understand the concepts both included and not included in their code. While interviews can provide a more complete picture of student learning, they are very time-consuming, making them unrealistic for teachers who are already very time-constrained. Further, these kinds of in-person approaches do not scale well to large classes (e.g., hundreds of students) and online learning. This paper also concludes that the correlation between the presence of specific code constructs on a student's code solution and his or her performance on a question asking about those code constructs is very low. Essentially, artifact analysis only shows that a student built something, not that they understand it.

Kennedy and Kraemer [3] also noticed that the ability to produce a program that behaved correctly did not guarantee that the students fully grasped the underlying concepts. Some students actually submitted erroneous code that just happened to work for a given task, while expressing doubts about the correctness of their implementations.

1.2. Context

There are currently multiple systems oriented towards the goal of providing feedback on a student's code solution, mostly identifying errors [4]. The typical use case for code submission platforms is focused around evaluating the code solution functionally (i.e. if it works or not). As explained previously, this might not be enough to assess a student's understanding of the code. Having a system that could mimic a human mentor could be more effective. The way a teacher usually assesses a student's knowledge of a given subject is through a test, oral or written, and provides individual feedback targeting the specific issues that came up in the learning process. Usually though, programming courses have lots of students and there are not enough resources to provide a continuous follow-up process to assess if they understand what they are doing.

Questioning a student's written code solution in detail can only be done by a human, currently. Since this approach is not scalable the idea of creating a system that could mimic this behaviour (to a certain extent) arose, as set out in [5]. The authors propose an approach to automatically generate questions about student-written program code. The prototype of this dissertation is a realisation of many of the ideas exposed in this paper.

1.3. Research Questions

- How to automatically derive questions (and corresponding answers) about arbitrary introductory programming code?
- To what extent can students correctly answer questions about the properties of their own code?

1.4. Objectives

The main objective of this work consists in investigating existing feedback generation systems, including their methodologies, context and results, and implementing a prototype that can generate questions as described in the previous sections. These questions can be static or dynamic, meaning that some questions can be generated by just statically analysing the code and inferring its context (e.g., 'How many parameters does function [F] have?') while others can only be generated by running the code solution on an execution environment (e.g., 'How deep does the call stack grow from executing [F]?'). Having a functional prototype, another objective is to use it to gather students' answers to questions about their own code and then take relevant conclusions.

In short, having a system that analyses code solutions submitted by the students and provides questions about their own code could help to measure their understanding of the underlying concepts.

1.5. Research Method

The reference model used for the investigation method is the Design Science Research Process (DSRP) [6] Model shown in Figure 1.

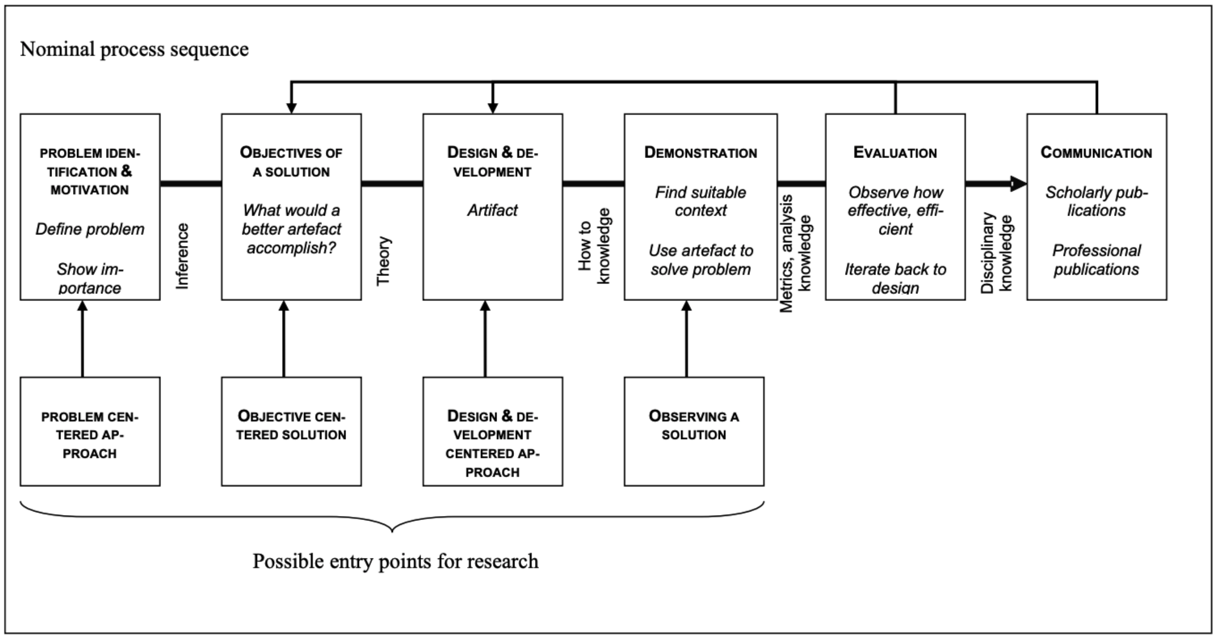

Nominal process sequence
PROBLEM IDENTIFICATION & MOTIVATION
Define problem
Show importance
Inference
OBJECTIVES OF A SOLUTION
What would a better artefact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, analysis knowledge
EVALUATION
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
COMMUNICATION
Scholarly publications
Professional publications
PROBLEM CENTERED APPROACH
OBJECTIVE CENTERED SOLUTION
DESIGN & DEVELOPMENT CENTERED APPROACH
OBSERVING A SOLUTION
Possible entry points for research

Figure 1. Design Science Research Process (DSRP) Model (Peffers, 2020)

In the context of this dissertation, the application of the DSRP model follows a problem-centred approach, starting with the "Problem identification and motivation" activity, since the idea for this research resulted from observation of an existing problem described in the motivation section of this dissertation. The main objective of the proposed solution consists in being a complementary self-assessment tool that can help students to better understand programming code concepts. The artifact is then created in the ‘Design & Development’ activity. This activity includes determining the artifact’s desired functionality and architecture as well as implementing it. In the “Demonstration” activity the artifact’s efficacy is demonstrated to solve the problem. This demonstration consists in experimentation of the artifact in a particular context (e.g., classrooms or online classes). The results are then measured in the “Evaluation” activity. These results are measured by comparing them to the objectives proposed initially in the “Objectives of a solution” activity. Depending on this measurement and time constraints one can iterate back to the “Design & Development” activity to further improve the effectiveness of the artifact or continue to the “Communication” activity, i.e. when the work is published and/or presented. This includes communication of the problem and its importance as well as the artifact and its utility.

1.6. Document Outline

Regarding the document structure, chapter 2 describes the research conducted in multiple subjects related to the topic of this dissertation. Chapter 3 defines important terms that serve as theoretical background that is necessary to understand the following chapters. In chapter 4, technical aspects regarding the prototype’s architecture and development are discussed, and how it accomplishes one of the main objectives proposed. Chapter 5 describes how the prototype was evaluated and how it was used to take relevant conclusions about students’ understanding of their own code, accomplishing another proposed objective. Finally, conclusions are drawn and steps for future work are described in chapter 6.

CHAPTER 2

Literature Review

This chapter shows the research conducted on multiple subjects related to this dissertation. It starts by analysing studies related to problems that students face when writing code and further establishes the relevance of this research. Then, a brief review on the subject of static code analysis is done since it is a prerequisite for generating questions based on students' code. Afterwards, existing general code feedback systems are reviewed since it allows for understanding of the mechanisms and strategies used to generate and present feedback to the users. Finally, question generation as code feedback in particular is also reviewed to understand how existing prototypes work and how the prototype presented in this dissertation differs from these.

2.1. Problems students face when writing code

A literature review on students' misconceptions and difficulties [7] asserts that students exhibit misconceptions and other difficulties in syntactic knowledge, conceptual knowledge (how the code works) and strategic knowledge (when and how to use code to solve a problem). Many sources of these difficulties are related to the students' prior knowledge in fields like math and natural language while others are related to teachers' knowledge and instruction.

Derus [8] explored the views of students and the difficulties they experience while learning to program in fundamental programming courses. It is concluded that lack of visualisation of the program's state during code execution is one of the major contributing factors of students' difficulty in learning how to program along with lecturers not providing enough practical code examples.

A study conducted by a ITiCSE working group [9] established that many students do not know how to program at the conclusion of their introductory courses. The highlighted cause for the problem was that students lack the ability to problem-solve, meaning that they cannot decompose a problem into smaller problems, solve them and combine the results to reach a final solution. A later ITiCSE working group [10] spanning twelve institutions in seven countries established that many students lack the knowledge and skills that are prerequisites for problem solving. The skills are more related with the students ability to read code, as opposed to writing it as many students failed to systematically analyse a short piece of code.

This dissertation also aims to understand the difficulties faced by students when writing code in order to further contribute to this area of study.

2.2. Static code analysis

Statically analysing programming code is a prerequisite for generating questions based on a student's code solution. Gomes et al. [11] wrote an overview on static code analysis including existing static analysis tools for different programming languages. Essentially, there are 2 types of software analysis: dynamic and static analysis. Static analysis is the analysis of computer software without the execution of the programs built from that software. Dynamic analysis is performed by executing programs on a real or virtual processor. Some static analysis tools for Java noted in this paper include FindBugs, PMD, CheckStyle and others. These tools perform syntactic checks on program source code and report any breach of standards.

Striewe and Goedicke [12] reviewed different static analysis approaches for programming exercises and the respective tools used. The approaches covered in the paper are limited to the context of object-oriented programming in Java. These tools are categorized depending if they analyse source code, bytecode or both. Since bytecode analysis does not fulfill all requirements for learning scenarios and source code analysis seems to do so, the latter might be the preferred approach. Some approaches use abstract syntax trees (AST) while others use abstract syntax graphs (ASG). The main difference between these is that an ASG is an AST enriched with additional arcs. Essentially, it shows connections between method call nodes to respective method declarations, field access and field declaration, etc. It should be noted that it's only a difference of data formats since ASGs are generated from ASTs so any information available in the graph is also available in the tree, albeit implicitly. The paper also mentions how one can integrate an external tool into an existing system. Essentially, two different ways of integration exist: integrate the tool as a library and use its API (Application Programming Interface) or start the tool from the command line as a separate process. All the tools listed in the paper provide API integration which is usually the preferred method.

2.3. Generic code feedback

Substantial work has been done regarding the subject of providing automatic feedback to the user, according to [4]. However, this feedback is usually about the coding errors, if they exist. This approach might not be enough considering the student might just search for a solution he or she does not understand and submit it. According to this systematic review, there are three great types of difficulties that students find: syntax knowledge, conceptual knowledge and strategic knowledge. Syntax knowledge can be acquired with feedback based on compiler errors. Conceptual knowledge can be acquired with explanations on subject matter and examples illustrating concepts. The solution proposed in the context section of this dissertation can be used to address this type of difficulty and improve conceptual knowledge of the problem as the student will always have to reflect on his or her code solution, regardless if his or her submission was correct. Lastly, strategic knowledge can be acquired with compiler errors, knowledge about task constraints and knowledge about how to proceed. This review also concludes that, despite many techniques and studies have been done on this matter, it is still a challenge to provide significant and diversified feedback to the user that does not focus only on errors.

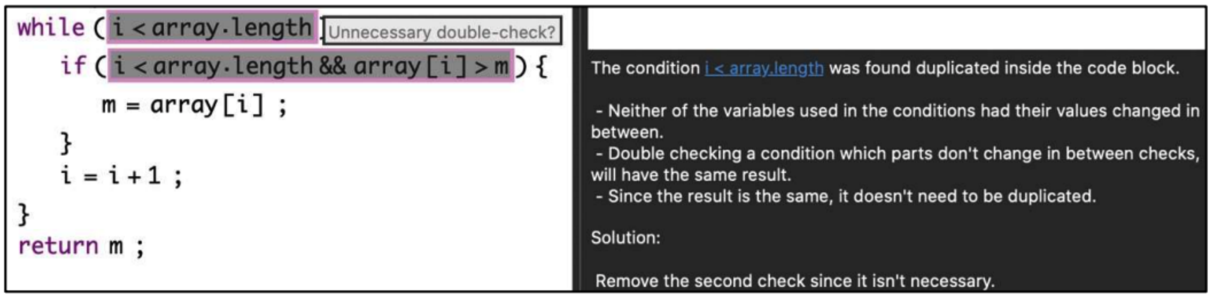
Rivers and Koedinger [13] used a method that relies on using solution spaces. A solution space is, in this context, ‘a graph representation of all the possible paths a student could take in order to get from the problem statement to a correct answer, where the nodes are candidate solutions and the edges are the actions used to move from one solution state to another’. After creating the solution space a method based on the Hint Factory [14] is used, i.e a logic tutor which uses prior data to give stuck students feedback on how to proceed. Essentially, the method described uses solution spaces to determine where a student is in his or her problem-solving process and then traverses that space to find the nearest correct solution to determine what feedback to provide to the student. Nevertheless, the authors note that despite having the solution spaces implemented and tested, the process of generating feedback is still in progress.

Parihar et al. [15] developed the GradeIT system. It provides automated grading, repairing and feedback. They noted that compiler messages are generally geared towards professional programmers and may refer to advanced concepts not covered in class despite forming the basic feedback for most programming assignments. Therefore, a simplified feedback for most frequent errors was created. A fixed set of rewrite rules is used to reformat the compiler error message into a simple message suitable for a novice programmer. Furthermore, two types of examples are provided: valid code fragments to explain to the student how to correct the error and invalid code fragments to show common variations of the error.

Marin et al. [16] proposed a semantic-aware technique to provide personalized feedback for java assignments in the context of Massive Open Online Courses (MOOCs). For a given assignment, instructors select relevant patterns they expect to find in student submissions from a provided knowledge base. They can also set different types of constraints among these patterns to correlate them and provide more fine-grained feedback. The student's code submission is transformed into an extended program dependence graph and subgraph pattern matching is performed over it to provide personalized feedback associated with the patterns and constraints indicated by the instructor.

Alfredo [17] developed a pedagogical tool that uses static analysis to identify issues with students' code, while providing an explanation on why a given bad practice present in the code can harm their learning. The tool uses the notion of a Control Flow Graph to implement detectors of bad practices. A detection example is show in Figure 2.

The type of feedback provided by the prototype of this dissertation has a different approach from the ones mentioned previously in the sense that it does not provide steps for a student to reach a solution. The code solution is submitted and then questions are generated as feedback so both students and professors can assess students' understanding of their own code.



```
while ( i < array.length ) Unnecessary double-check?
   if ( i < array.length && array[i] > m ) {
      m = array[i] ;
   }
   i = i + 1 ;
}
return m ;
```

The condition i < array.length was found duplicated inside the code block.

- Neither of the variables used in the conditions had their values changed in between.
- Double checking a condition which parts don't change in between checks, will have the same result.
- Since the result is the same, it doesn't need to be duplicated.

Solution:

Remove the second check since it isn't necessary.

Figure 2. Code bad practice detection (Alfredo, 2020)

2.4. Question generation as code feedback

There is earlier work on automatic generation of questions for programming code. For instance, Zavala and Mendoza [18] used semantic-based Automatic Item Generation (AIG) in order to create programming exercises that can be used for assignments or quizzes in introductory programming courses. The idea behind AIG is that one can use a test-item template with embedded variables and formulas to generate hundreds or even thousands of test-items, i.e variations of the same question. In this case, Linked Open Data [1] is a method used in this semantic-based AIG approach to generate programming exercises that reflect real-world concepts. An example of this approach, based on a diagram present in the article, is shown in Figure 3. It shows a test-item template serving as input to the algorithm that instantiates it and outputs multiple test-items.

Thomas et al. [19] used a stochastic tree-based generation algorithm and a subsequent simulation of execution to generate small programs accompanied by multiple-choice questions. The incorrect answer options serve as distractors that correspond to reasonable paths of execution of the program but are ultimately incorrect. This line of work is primarily motivated by the need to provide many fresh questions for students to practice on, and/or as a precaution for plagiarism. The approach presented in this dissertation, while sharing the same general objective of generating questions automatically, differs specifically in what types of questions are generated. The objective is to infer the context of the student's solution and generate questions adapted to that same context, not generate variants of the same exercise.

[1]https://www.w3.org/standards/semanticweb/data

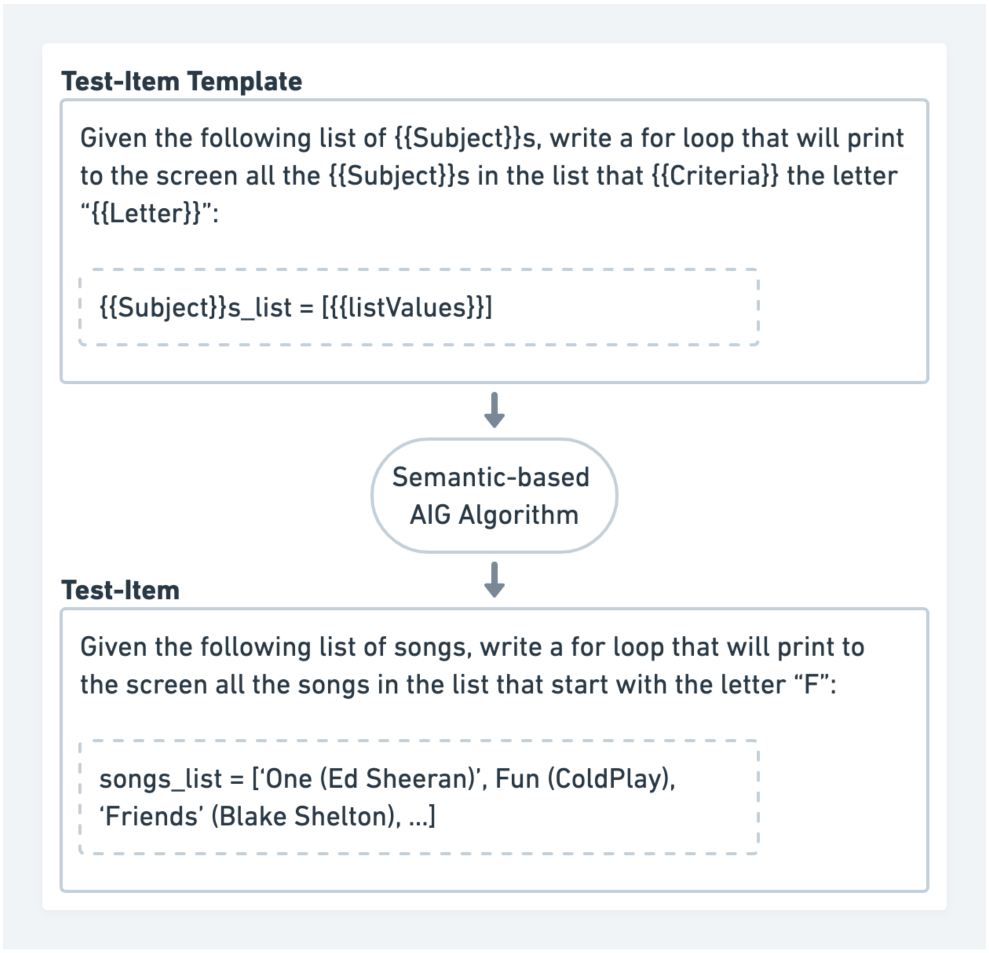

Test-Item Template
Given the following list of {{Subject}}s, write a for loop that will print to the screen all the {{Subject}}s in the list that {{Criteria}} the letter "{{Letter}}":
{{Subject}}s_list = [{{listValues}}]
Semantic-based AIG Algorithm
Test-Item
Given the following list of songs, write a for loop that will print to the screen all the songs in the list that start with the letter "F":
songs_list = ['One (Ed Sheeran)', Fun (ColdPlay), 'Friends' (Blake Shelton), ...]

Figure 3. Semantic-based AIG in computer programming

As previously mentioned, this dissertation is a realization of many of the ideas exposed in [5]. This paper defines the concept of questions about learner's code (QLCs) as well as a process for generating them automatically. Figure 4 illustrates this process. Essentially, when a student submits code, it passes through a static analyser that extracts static facts (e.g., the number of variables) and through an execution engine that gathers dynamic facts (e.g., the call stack depth). Both these types of facts are used by the 'Question Engine' to determine which of the QLC templates present in the 'Template Database' are applicable to the submitted code. The authors also propose other inputs for the system like test cases and settings defined by a teacher. This dissertation builds on many of these concepts in order to implement the proposed prototype.

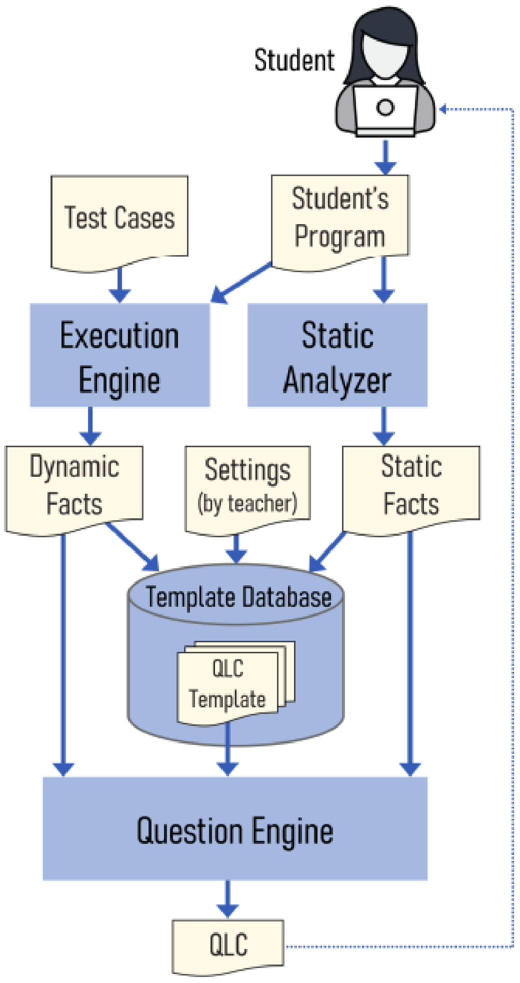


Figure 4. A process for generating QLCs (Lehtinen, 2021)

CHAPTER 3

Question Definition

This chapter starts by defining what question templates are and why they are important to generate relevant questions to the students. It then describes the existing question types and lists concrete question template examples segregated by these types.

3.1. Question Templates

In order to generate questions for students based on their submitted code, outlines for the questions must be prepared beforehand. These outlines are called 'Question Templates' and depending on the submitted code, instances of these templates, i.e. the questions themselves, will be generated. This dependency is expressed through a concept called 'Applicability' and each question template defines its own applicable rules. For example, a question concerning the number of parameters in a method should only be asked about a method that has at least one parameter defined in its method signature. The code snippet in Listing 1 shows a simple Java class composed of three methods and will be used as a running example for this chapter.

3.2. Static Questions

Static questions are questions that are generated by statically analysing the code. Static analysis is performed without actually executing the code and allows to retrieve relevant information from it, for example, the number of variables, parameters, method calls, etc. In order to generate static questions for a particular code construct, it must be possible to get elements (e.g., methods or functions) of a particular code file or program. In this case, a code analysis tool like *Paddle* [2] provides an API that allows to extract relevant information from the code. Each method contains variable declarations and assignments, method calls, etc. From the collected information it is then possible to

[2] `https://github.com/andre-santos-pt/paddle`

generate an appropriate question for a particular method or function and the respective answer. For instance, considering the code present in Listing 1, it is possible to verify that method *calculate* depends on two other methods and that it has three parameters. This type of information about the code can be posed as questions to the student after statically analysing it. Overall, these types of questions are more focused on introductory programming concepts like code semantics and are typically easier to answer compared to dynamic questions since they do not require the student to follow the code's execution path.

Table 1 contains descriptions as well as the applicability of static question templates.

```java
public class Calculator {

    public int calculate(int x, int y, char operation) {
        if (operation == '+') return sum(x, y);
        else if (operation == '-') return sub(x, y);
        return 0;
    }

    private int sum(int x, int y) {
        return x+y;
    }

    private int sub(int x, int y) {
        return x-y;
    }
}
```

Listing 1. Code submission running example

Table 1: Static Question Templates

Question Template	**Applicability?**	**Other observations**
Does ***{function}*** depend on other functions?	This is a 'yes or no' question that is asked about any function present on the submitted code. Since it can be asked about any function in the code it does not have any rule in order to be applied to a particular context.	The user is expected to answer with a boolean value (*true* or *false*) that indicates if the function depends on other functions or not.
How many functions does ***{function}*** depend on?	This question is asked if a given function depends on other functions, i.e. it is applicable to a given function if the answer to the question 'Does ***{function}*** depend on other functions?' is *true*.	The user is expected to answer with an integer value, indicating how many functions or methods the given function depends on.
How many loops does ***{function}*** have?	This question is asked if a given function has at least one loop construct (e.g., while/for) present in the code.	The user is expected to answer with an integer value, indicating how many loop constructs are present in a given function.
How many parameters does ***{function}*** have?	This question is asked if a given function has at least one parameter.	The user is expected to answer with an integer value, indicating how many parameters a given function has.
How many variables (not including parameters) does ***{function}*** have?	This question is asked if a given function has at least one variable.	The user is expected to answer with an integer value, indicating how many variables (not including parameters) a given function has.

Question Template	**Applicability?**	**Other observations**
Is ***{function}*** recursive?	This is a 'yes or no' question that is asked if a given function is recursive or has loop constructs present in the code. The reason for the latter is that it allows to distinguish between students that understand the difference between recursion and use of loop constructs.	The user is expected to answer with a boolean value (*true* or *false*) that indicates if the function is recursive or not.
What are the fixed value variables of ***{function}***?	This question is asked if a given function has at least one variable and it only has one assignment. It can be though of as a constant even though it does not need to be represented as such (i.e. it can be a simple variable declaration that does not have more than one assignment).	The user is expected to answer with a list of string values that represent the variables that have a fixed value.
What functions does ***{function}*** depend on?	This question is asked if a given function has at least one other function reference present in its body. This means it is applicable to a given function if the answer to the question 'Does ***{function}*** depend on other functions?' is the value *true*.	The user is expected to answer with a list of string values that represent the function or method calls present in a given function or method body.

Question Template	**Applicability?**	**Other observations**
What is the most determinant role of ***{variable}*** in ***{function}***?	This question is asked if a given function has a variable with a particular role. For example, a variable might be responsible for accumulating, incrementing or decrementing values, etc.	Since the names for these roles are subjective (i.e. do not have a strict, objective naming convention) a list of string values is provided to the user and he or she is expected to pick one of the values as the answer.
What are the parameters of ***{function}***?	This question is asked if a given function has at least one parameter. This means that this question is applicable if the answer to the question 'How many parameters does ***{function}*** have?' is greater than zero.	The user is expected to answer with a list of string values that represent the function's parameters.
What are the variables (not including parameters) of ***{function}***?	This question is asked if a given function has at least one variable (not including parameters). This means that this questions is applicable if the answer to the question 'How many variables (not including parameters) does ***{function}*** have?' is greater than zero.	The user is expected to answer with a list of string values that represent the function's variables.

Question Template	Applicability?	Other observations
Which variable will hold the return value of ***{function}***?	This question is asked if the function uses only one variable in its return statement. Since it is a static question and the code is not executed to generate this question, the question is only applicable if the return statements, should there be more than one, use the same variable.	The user is expected to answer with a string value that represents the variable that holds the return value.

3.3. Dynamic Questions

By contrast, **dynamic** questions are generated by analysing the code's behaviour. Dynamic analysis is performed by executing the code on a real or virtual processor. Since code is executed, this type of analysis allows the retrieval of information not present in the code itself including the return value of a given function, call stack depth of a program, variable assignments and method calls. Considering Listing 1, a student can be asked about the return value of method *calculate*. Since the value is not present in the code the student has to think about the code's execution path in order to provide an answer.

When generating dynamic questions, the way in which information is collected is different from generating static questions. With the latter, the values or responses for a given question are naturally present in the code while with the former these values are retrieved from the code execution. One has to keep in mind that for distinct executions of a given method the code path might be different depending on the code constructs and arguments generated to execute it. For example, in Listing 1, in the *calculate* method, if the operation argument is '+' or '-' the *sum* and *sub* methods will be called, respectively while any other operation argument will make the method return 0. In this case, the entire trace from a single method execution must be collected and persisted on a proper data structure, not only because of consistency but also performance reasons. From this data, appropriate questions can be posed to the student for a particular method, for example, the return value, number of variable assignments, call stack depth, etc.

Since the code is executed on a virtual processor it is also necessary to generate appropriate arguments in case one or more parameters are present in the method definitions.

Table 2 contains descriptions as well as the applicability of dynamic question templates, assuming the method definitions contain parameters. The alternate, parameterless, versions of the question templates are similar and differ only on the use of some words and the lack of arguments. They are not shown for the sake of brevity.

Table 2: Dynamic Question Templates

Question Template	**Applicability?**	**Other observations**
How deep does the call stack grow from the following invocation: ***{function(arguments)}***?	This question is asked if a given function has a call stack depth greater than 1. Considering the function invocation itself makes the call stack have a depth of 1 a method call in this function increases the call stack depth to 2. Other method calls in the original function do not increase the call stack depth. For example, in order for the call stack to have a depth of 3 the method that the original function calls must also have a method call inside its body.	This question allows to distinguish between students that understand how the call stack works and those that know how to follow the logic in a program that calls multiple functions but do not know how the call stack itself works internally. The user is expected to answer with an integer value that represents the function's call stack depth.

Question Template	**Applicability?**	**Other observations**
How many function calls are made with the following invocation: ***{function(arguments)}***?	This question is asked if a given function calls one or more functions. This question differs from its analogous static question mentioned previously in the sense that it asks about the calls effectively made in the code's execution, not the presence of the function definitions in its body. For example, if a function call is present inside a conditional if statement it will not get called if the expression evaluates to 'false'. In this case, it does not count as a function call but it counts for the number of functions the original function is dependent on.	This question requires the student to follow the function's logic path instead of simply counting the number of function references present in the code. The user is expected to answer with a integer value that represents the number of function calls made.

Question Template	Applicability?	Other observations
How many times a value is assigned to ***{variable}*** from the following invocation: ***{function(arguments)}***?	This question is asked if the given function has at least one variable that is assigned a value more than once. Like the previous dynamic questions, this question requires the user to follow the function's logic path instead of simply counting the number of assignments that are visually perceived. The number of assignments can greatly vary depending on the existence of conditional statements and loop constructs.	The user is expected to answer with a integer value that represents the number of times a value is assigned to a given variable.
What is the return value from the following invocation: ***{function(arguments)}***?	This question is asked if a given function does not have a void return type.	The user is expected to answer with a string that represents the function's return value. The value itself might be a string or number.
What is the value sequence taken by ***{variable}*** with the following invocation: ***{function(arguments)}***?	This question is asked if there is at least one variable in the given function that has more than one assignment.	The user is expected to answer with a list of string values that represent the variable's ordered values, from the beginning to the end of the function's execution.

CHAPTER 4

Prototype Design and Implementation

This chapter describes the system that was built and used to generate the previously discussed questions. It starts by describing the system's architecture, namely how its components interact and the technologies used. Then, it describes how users can interact with it and how it behaves. Afterwards, the architecture of the question engine component is discussed specifically since it generates the actual questions and is this dissertation's main focus. It then proceeds with an overview of the database model and its relation with the question engine component. Finally, the chapter ends by mentioning a couple of technical limitations present in the prototype.

4.1. High-Level Architecture

The whole system was designed taking user experience and usability into account. Therefore, it is built, looks and feels like a web application for the end users, i.e. the students. The diagram in Figure 5 shows a high-level overview of the main components of the prototype.

The system follows a traditional client-server architecture and the interactions between these two components are made through REST [3] calls. The 'Question Generator UI' is a frontend application built with the TypeScript programming language and the Angular [4] framework. It is deployed and publicly available on the Firebase [5] platform for the user to interact with, including submitting the code and the answers to the generated questions.

[3] https://restfulapi.net/
[4] https://angular.io/
[5] https://firebase.google.com/

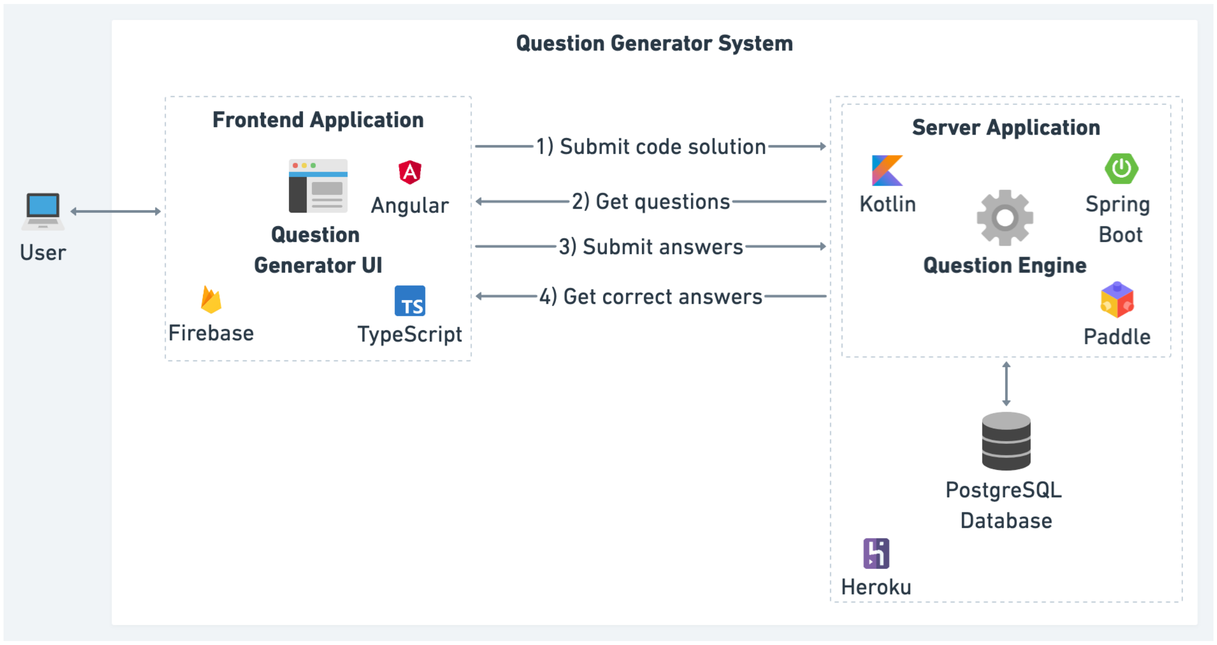


Figure 5. High-Level System Architecture

The business logic, including question generation and data processing occurs in the 'Question Engine' server application, a backend application built with the Kotlin programming language and the Spring Boot framework [6]. It also uses the 'Paddle' library for static code analysis and running the code in a virtual machine. The data generated from user interactions (code, questions, submitted and correct answers) is mapped to each user and persisted in a PostreSQL database. Both the server application and the database are deployed on the Heroku platform [7].

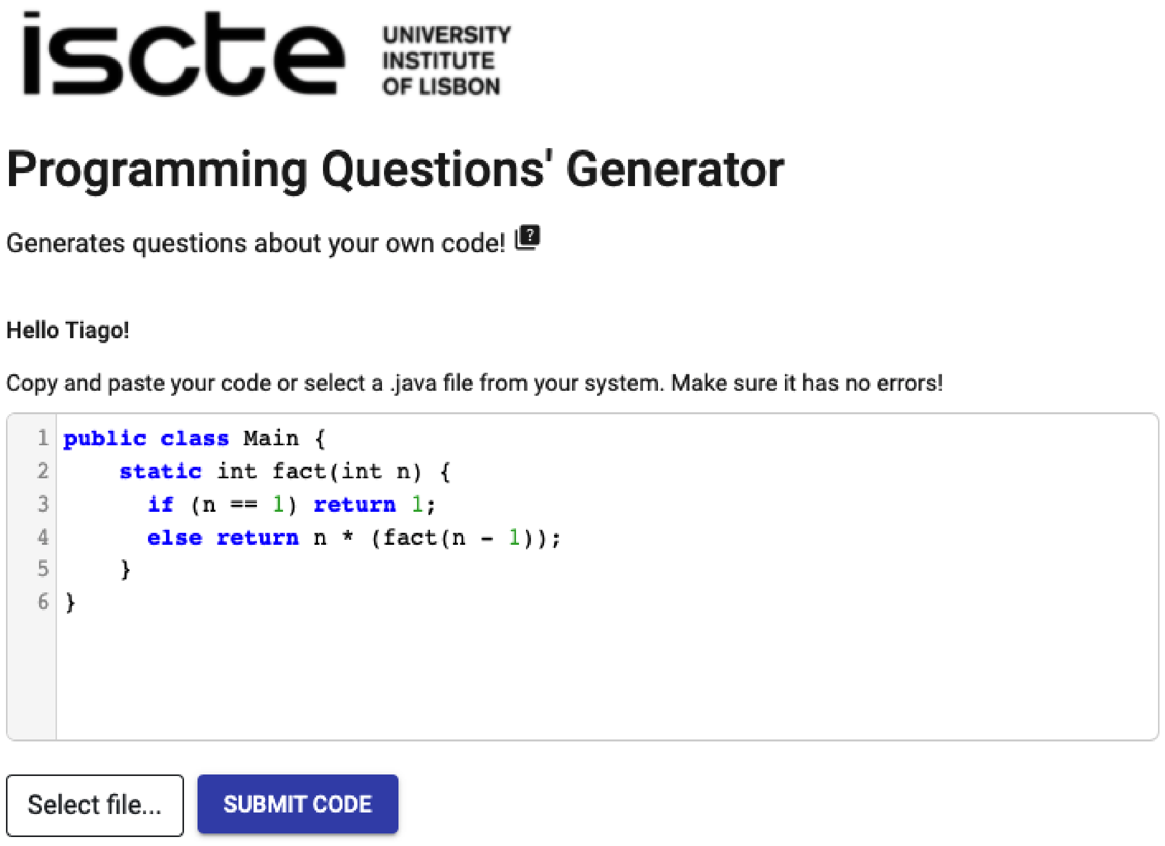
One of the main reasons for splitting the frontend and backend applications in this case is to allow separate deployments. At a given time in the prototype where further development on a particular application may not be needed, one can develop and deploy another application without affecting the former. This also translates into overall increased system availability as a user might be interacting with the 'Question Generator UI' (e.g., typing the code in the editor) while a quick deployment of the server application is being performed. It is also easier to understand where a given problem might be

[6] https://spring.io/projects/spring-boot
[7] https://www.heroku.com/

coming from any stage of development (i.e. building, testing or deploying the solution) if the components are isolated.

4.2. User Interaction Flow

In order to interact with the system, the user uses a browser to access the frontend application, which is the only application that is exposed directly to this user. The user first logs in with his or her google account and can then type his or her code (in Java) directly on the code editor, copy and paste or upload prepared code as shown in Figure 6.



Programming Questions' Generator

Generates questions about your own code!

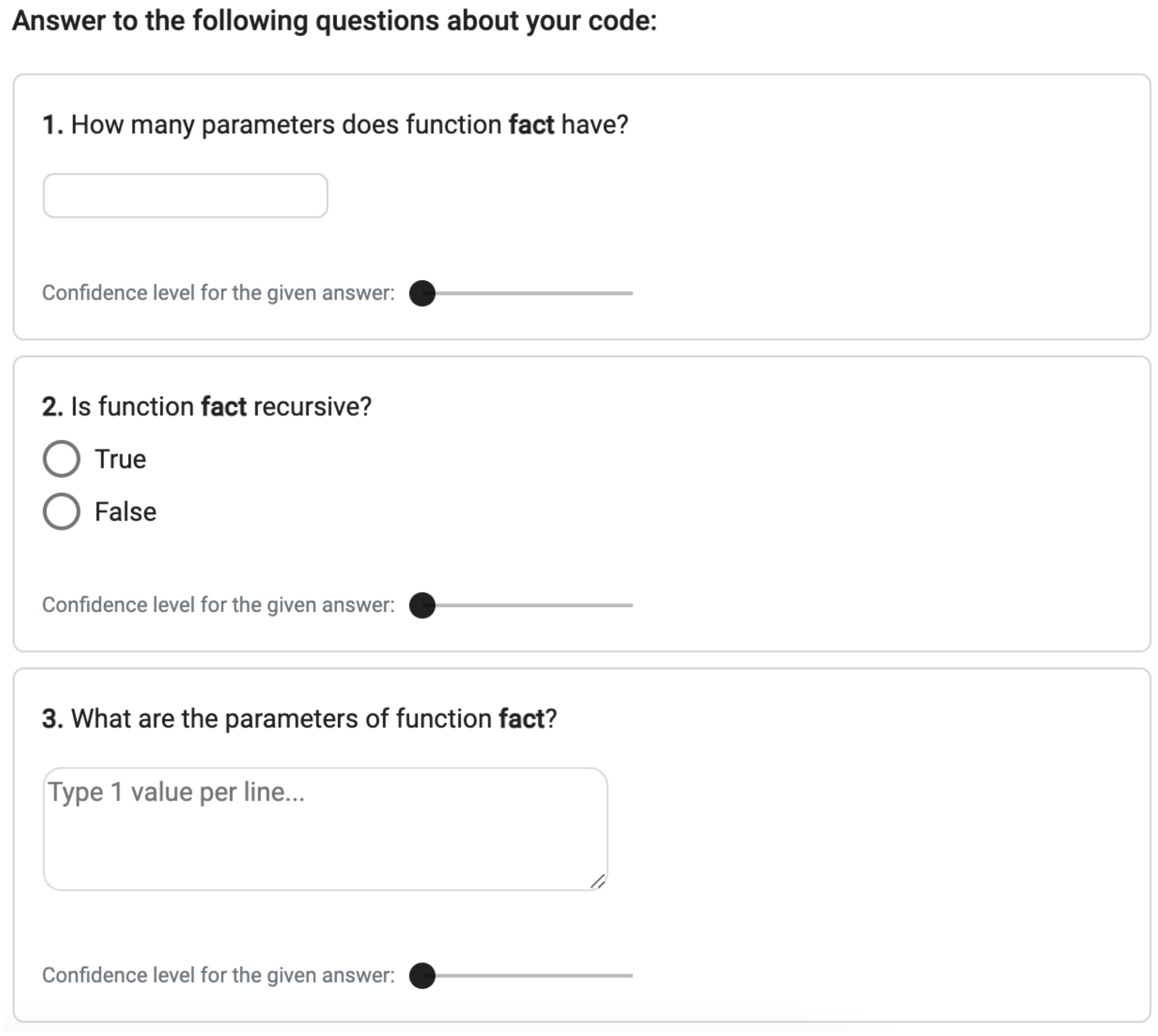
Hello Tiago!

Copy and paste your code or select a .java file from your system. Make sure it has no errors!

```
public class Main {
    static int fact(int n) {
      if (n == 1) return 1;
      else return n * (fact(n - 1));
    }
}
```

Figure 6. Code submission

After submitting the code, the user is presented with relevant questions. A subset of generated questions can be seen in Figure 7.


Answer to the following questions about your code:

1. How many parameters does function **fact** have?

Confidence level for the given answer:

2. Is function **fact** recursive?

True

False

Confidence level for the given answer:

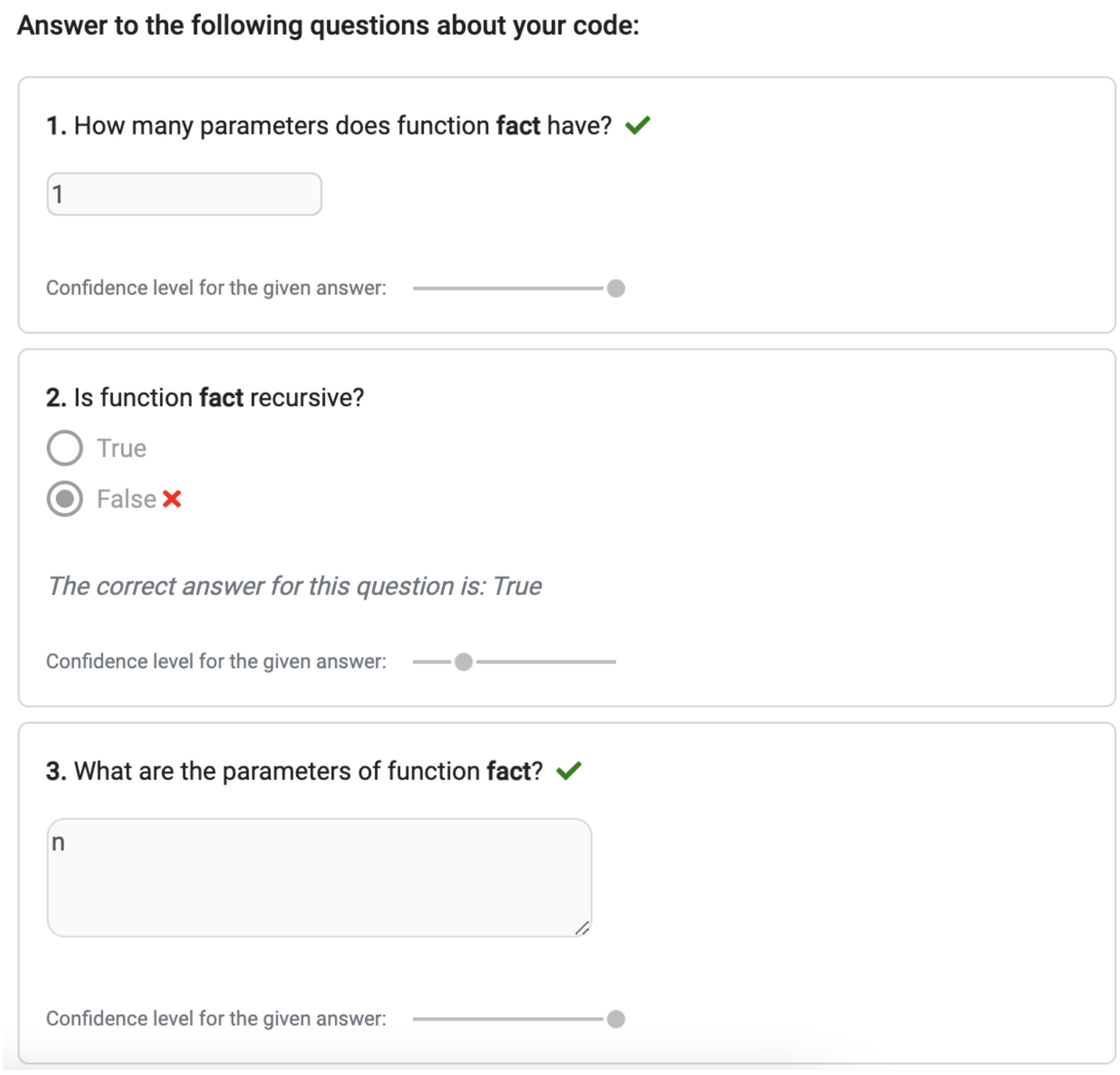
3. What are the parameters of function **fact**?

Type 1 value per line...

Confidence level for the given answer:

Figure 7. Generated questions

For each question type the user has the appropriate input in order to answer, i.e. number, text, selection, etc. For each answer, the user also selects and submits a confidence level indicating how confident he or she is that the answer to a given question is correct, with a value of 1 indicating the lowest confidence level and a value of 5 indicating the highest confidence level. After the user submits the answers and corresponding confidence levels, the system processes and persists this data and returns the correct answers to the questions as can be seen in Figure 8. This way, the user can see which answers are correct or incorrect and try to improve his or her level of understanding about the code. Afterwards, the user can make a new code submission to restart the process.


Answer to the following questions about your code:

1. How many parameters does function **fact** have? ✓

1

Confidence level for the given answer:

2. Is function **fact** recursive?

True [✓ False] ✗

The correct answer for this question is: True

Confidence level for the given answer:

3. What are the parameters of function **fact**? ✓

n

Confidence level for the given answer:

Figure 8. System's feedback on student's answers

4.3. Question Engine Architecture

The 'Question Engine' component is responsible for receiving, processing and persisting user data as well as generating appropriate questions and respective answers for the code the user submits. The service's main components are presented in Figure 9.

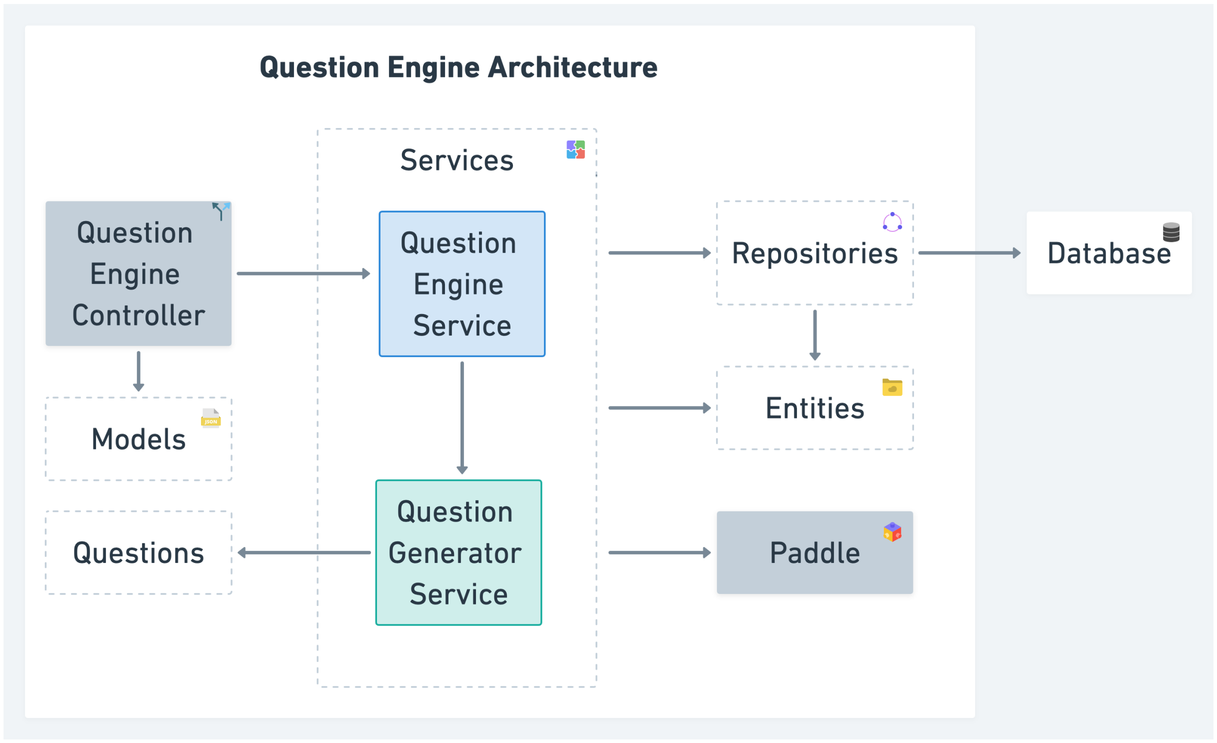

Question Engine Architecture
Services
Question Engine Controller
Question Engine Service
Repositories
Database
Entities
Models
Questions
Question Generator Service
Paddle

Figure 9. Question Engine Architecture

The *QuestionEngineController* is responsible for exposing the necessary endpoints for a client to use, in this case, the 'Question Generator UI' application. These endpoints correspond to the two main actions that the user of the system will take, namely, submitting code and answers to the generated questions. The *QuestionEngineService* is injected into the controller through dependency injection, which allows it to use the public methods exposed by the service without manually instantiating it. This service is responsible for the whole systems' orchestration which includes making the necessary calls to other services, repositories or other components depending on the action that is being taken, i.e. code submission and answer submission. When code is submitted to the system, it is first formatted and validated and in case there is a syntactic error, an exception occurs alerting the user to this fact. In case the code is valid, the system stores it in the database and associates it with a particular user. It then calls the *QuestionGeneratorService* to generate appropriate questions about the code, i.e. instances of question templates.

Each question template is represented as a class that ultimately implements behaviour from the question interface represented in Listing 2.

```
interface Question<T> {
    fun question(target: T): String
    fun applicableTo(target: T): Boolean
    fun answer(target: T): Any
    fun proficiencyLevel(): ProficiencyLevel
}
```

Listing 2. Question Interface

The *T* represents a generic type that serves as an input parameter to define the question in the *question* function. The correct answer for the question is determined in the *answer* function and depending on the question template, *T* can represent a procedure, a variable or even a complex object containing data about the whole procedure. The *proficiencyLevel* function defines the question's proficiency level, from 'C' to 'A', where 'C' indicates the lowest level of proficiency and 'A' indicates the highest level of proficiency. Each user is also assigned a level of proficiency based on his or her correct/total submitted answers ratio.

The question interface hierarchy is represented in Figure 10. As previously mentioned, questions are conceptually split into two types, static and dynamic. In the case of static questions, only static analysis is performed. Otherwise, the code is also run in a virtual machine. For both these use cases the 'Paddle' library is used.

The *StaticQuestion* interface extends the *Question* interface indicating that the *T* generic type should be any element present in a given code submission like a procedure or a variable. In fact, the *StaticQuestion* interface is extended to represent this exact use case. When generating questions about a given procedure, classes implementing the *ProcedureQuestion* interface will be instantiated. When generating questions about a given variable declaration, classes implementing the *VariableQuestion* interface will be instantiated.

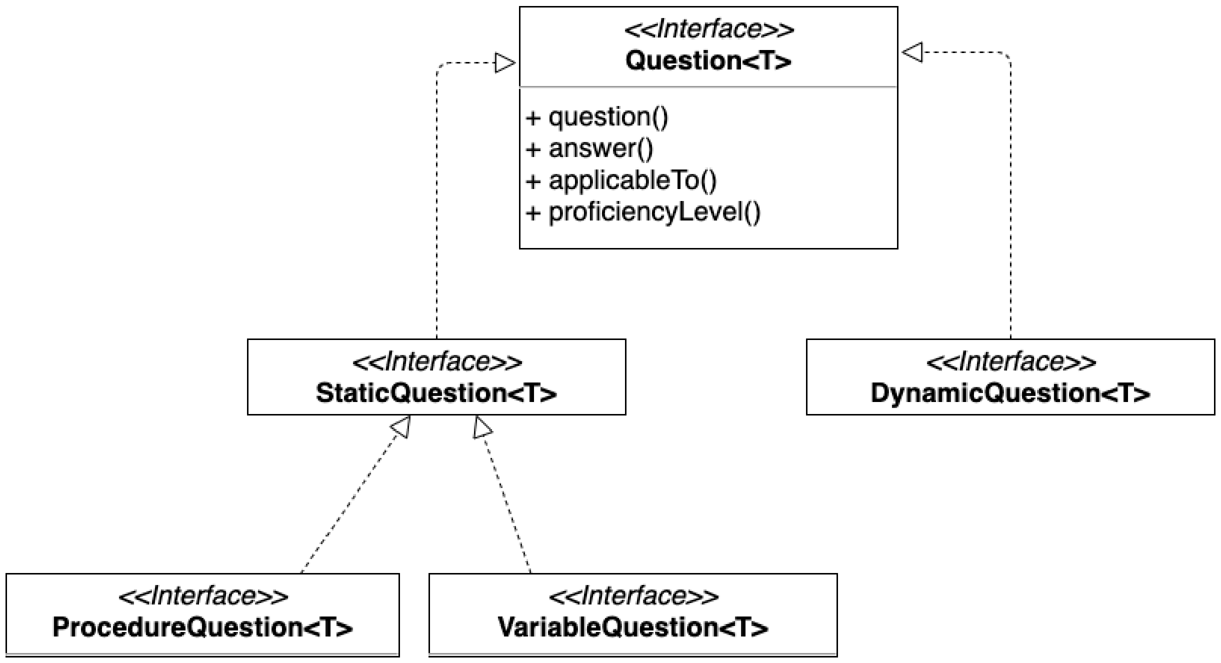


Figure 10. Question Interface Hierarchy

The *DynamicQuestion* interface extends the *Question* interface indicating that the *T* generic type should be a special object class. This class holds runtime related information about a given procedure, including a collection of code constructs named 'Procedure Call Facts'. A 'Procedure Call Fact' can be though of as any action or relevant value that has been taken or preserved from running the referred procedure in the virtual machine. For instance, the call stack depth, the number of methods called, the return value and the number of variable assignments are all procedure facts gathered from a single procedure execution. This data, along with the generated arguments, is then used to generate relevant questions and answers for a code submission.

A simplified diagram of the question generating process is shown in Figure 11. The way these questions are generated is by iterating each question template instance and calling the *applicableTo* function using the *T* argument to filter out the non-relevant questions for a given function in a code submission. For example, considering a function 'F' with no parameters, when passed as an argument for the *applicableTo* function of the question template that asks about the number of parameters of a given function, this last function will return 'false'. This happens because the rules defined in this question template specify that the question is applicable if the code submission contains at least one parameter. In this case, it is filtered out of the set of applicable questions to be asked about a given function in a code submission. This type of validation occurs for every question template.

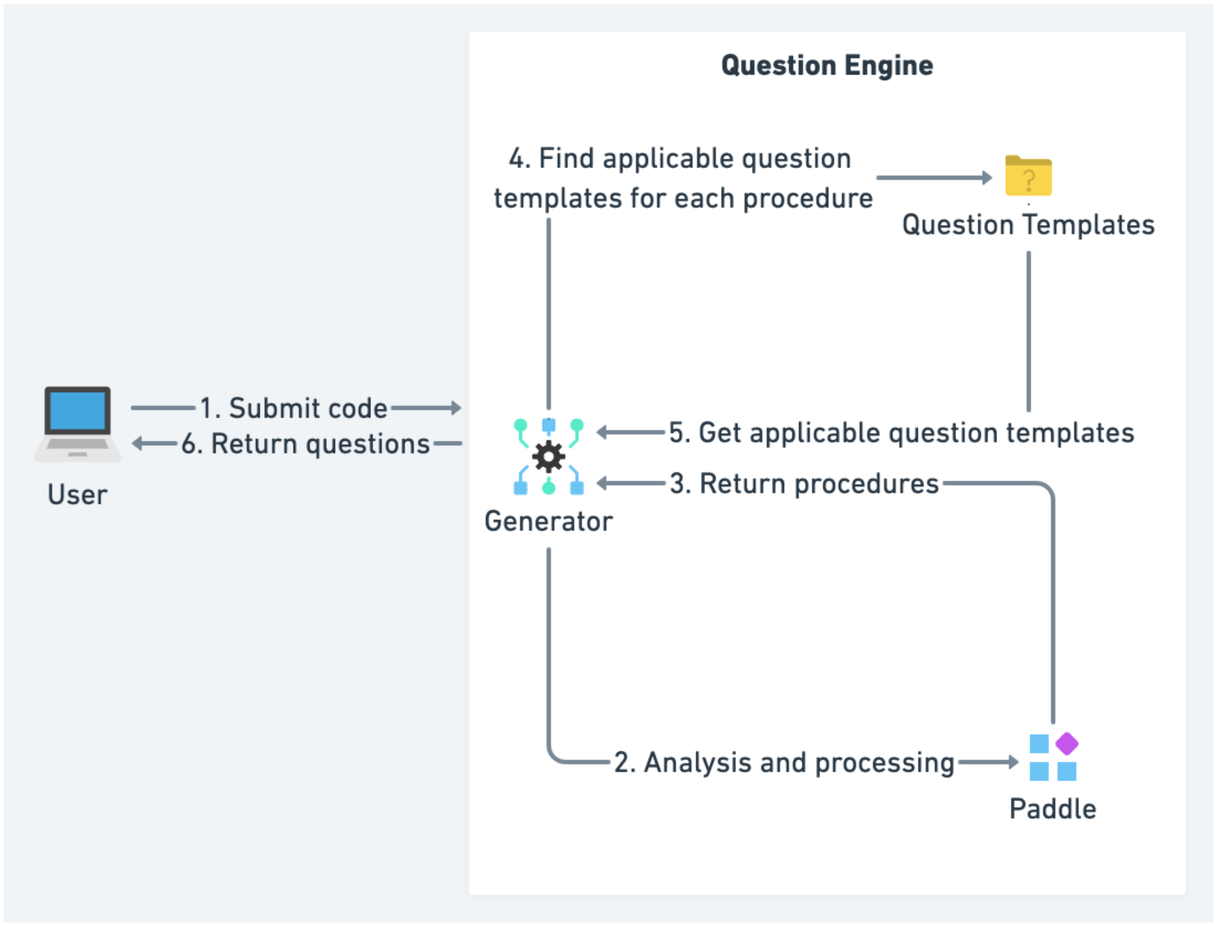

Question Engine
4. Find applicable question templates for each procedure
Question Templates
User
1. Submit code
6. Return questions
5. Get applicable question templates
3. Return procedures
Generator
2. Analysis and processing
Paddle

Figure 11. Question Engine Algorithm

Once these questions are generated, they are persisted in the database along with the respective correct answers, code submission and other relevant information including user information through the use of entities and repositories. In this context, an entity can be defined as a unit of data that captures the properties of a row of a table in the database. Listing 3 shows a representation of the *Question* entity class. The *@Entity* annotation specifies that the *Question* class is an entity and the *@Table* annotation specifies the table name and the schema that it belongs to. Each field represents a particular column in the the table and they can also be annotated depending on their role or relationship to other tables. More details on the database schema are given later in the database section of this chapter.

```
@Entity
@Table(name = "question", schema = "question_engine")
data class Question(@Id @GeneratedValue(strategy = GenerationType.IDENTITY)
                    var id: Long? = null,
                    @ManyToOne @JoinColumn
                    var questionTemplate: QuestionTemplate,
                    @ManyToOne @JoinColumn
                    var codeSubmission: CodeSubmission,
                    @ManyToOne @JoinColumn
                    var language: Language,
                    @OneToOne
                    var answerSubmission: AnswerSubmission?,
                    var question: String,
                    var correctAnswer: String,
                    var function: String) : AuditableEntity()
{}
```

Listing 3. Question Entity

A repository is a mechanism for encapsulating storage, retrieval, and search behaviour, which emulates a collection of objects [20]. Each repository is associated with a particular entity and, like services are used in controllers, they are used in services through dependency injection. Figure 4 shows a representation of the *QuestionRepository* interface, that is associated with the *Question* entity. This interface is annotated with the *@Repository* annotation to indicate that it is a repository. It extends the *CrudRepository* interface from the Spring framework so CRUD (Create, Read, Update, Delete) methods can be executed on the database table.

```
@Repository
interface QuestionRepository : CrudRepository<Question, Long> {}
```

Listing 4. Question Repository

After persisting the data, the questions are then returned back to the user through a model or DTO (Data Transfer Object) class. As the name suggests, a DTO is an object that is used to encapsulate data, and send it from one subsystem of an application to another [8].

When the user's answers to the questions are submitted to the system, it finds the saved questions and correct answers and associates them with the user's submitted answers. The user proficiency level is then updated based on the new correct/total submitted answers ratio and the correct answers are returned back to the client application, also through a DTO class.

4.4. Database Model

From the beginning of the conception of this prototype one of the core ideas was to somehow persist all the user interactions and respective generated information. Even though a functional version of this system could be built without it, having a database that records all these transactions is very beneficial. It allows splitting the whole problem into smaller steps (submitting code and submitting answers), gives the student a sense of progress and accomplishment between sessions through proficiency level updates and, probably most importantly, it allows for relevant conclusions regarding student's programming understanding to be drawn from the persisted data.

The previously discussed prototype implementation is tightly coupled with the database model displayed in Figure 12 (some details omitted for brevity).

The question templates are represented in the *question_template* table. Each question is associated with one question template and, as previously mentioned, a question template is associated with a proficiency level. It holds data regarding the return type

[8]https://martinfowler.com/eaaCatalog/dataTransferObject.html

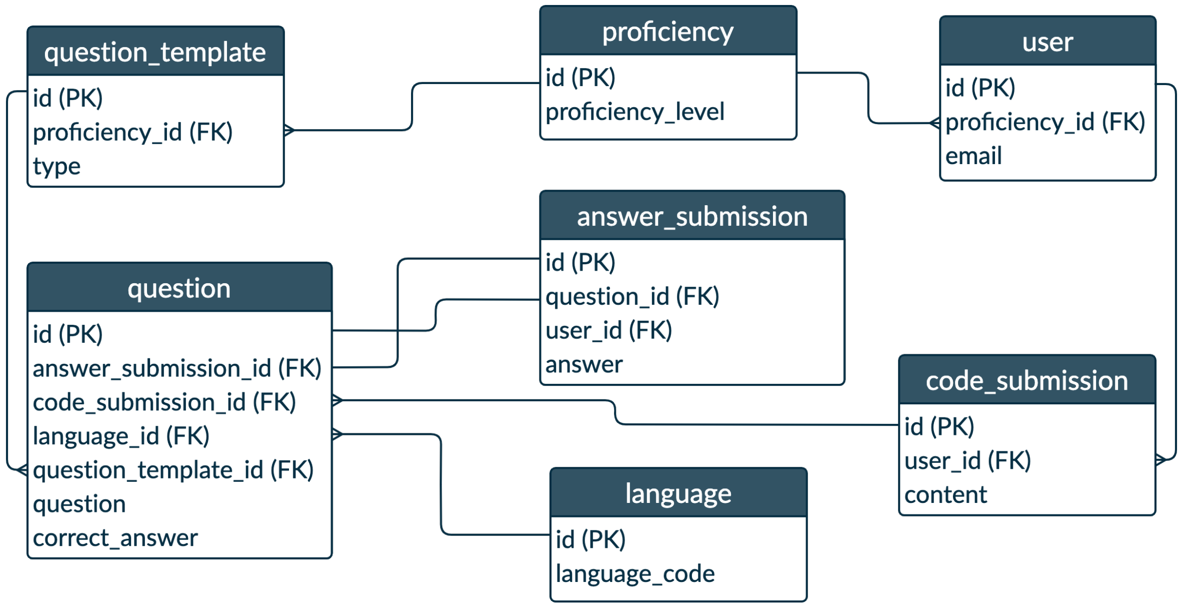


Figure 12. Database Model

allowing the client application to determine which type of input to display to the user, the question template type itself, i.e. static or dynamic, and the class name that models this behaviour. Since question templates are implemented in the code and it is hard to tell what question template is being processed or read from its id, the class name is used as a unique constraint in the table. When there is a code submission, the system will insert question templates of the generated questions in this table if they are not already present.

A question is represented in the *question* table and it is mapped to a given answer. It holds some relevant data to be used in the system's processing, including the correct answer for the question, which function the question asks about and the question itself. Each question, unlike question templates, is associated with one code submission meaning the system will always insert questions in *question* table for each submission. It is also associated with the *language* table. This allows the system to support multiple languages. A user can potentially generate questions in english or his or her native language. Naturally, translations for each language need to be added for the system to support this behaviour completely.

The *user* table holds user related data including their name, email and proficiency. This proficiency is represented in the *proficiency* table. It holds proficiency levels that can be assigned to both users and question templates and it is one of the links between these two entities. As discussed previously, the current prototype implements the proficiency levels ‘C’, ‘B’ and ‘A’, from the lowest to the highest proficiency level respectively.

The *code_submission* table holds the code submitted by a user and the *answer_submission* table holds the answer submitted by a user for a particular question.

4.5. Limitations

Although the prototype fulfills the proposed objectives it has some technical limitations that should be noted. For instance, there are a couple of questions that are presented as multiple-choice even though the system was not designed to generate incorrect answers as distractors for the student. In the case of the multiple-choice questions that exist, the values are hard coded and this is not a scalable solution. A concrete example of this is the question that asks about a variable role in a function. As previously mentioned, a list of choices is provided to the user but only one is correct. The incorrect choices are actually coded in the frontend application and are not generated by the question engine. This is by design since the system itself is mainly concerned with asking relevant questions, not providing relevant answers for these questions. In fact, the questions are generated after analysing and extracting certain values (the correct answers) from the code, not the other way around. Nonetheless, this design decision makes it harder to implement these types of questions which can be more visually appealing and intuitive for certain use cases. Therefore, generating more of these types of questions and finding a strategy to do so in a more technical appropriate way can be desirable.

CHAPTER 5

Evaluation

This chapter describes the experiment that was conducted using the developed prototype described in the previous section. It starts by describing the context and the participants of the experiment. Then, it describes how these participants were exposed to the system and how they used it. Finally, it lays down the results from the experiment as well as relevant observations and interpretations.

5.1. Context

In order to understand to what extent could students correctly answer questions about the properties of their code, the developed prototype was put to test in a few programming classes at the ISCTE-IUL university in the second semester of the 2020/2021 school year. These test runs also allowed to improve the prototype's stability and usability based on students' feedback and user experience. Initially, the system had a few bugs when submitting code and persisting data which resulted in some relevant data not being saved. Furthermore, some of the data that was persisted was not very useful on its own, i.e. some students submitted code and the system generated questions for it but they did not submit answers to the questions, possibly due to different reasons like system failure or just unwillingness to do so. Nonetheless, the data that was persisted can be used to draw some conclusions and insights.

5.2. Participants

For the purposes of this dissertation, there were 14 students that submitted valid code and answered to generated questions about it. These students all studied for either a bachelor's or master's degree with a very significant computer science background. Half of these students were in the first year of study while the other half was spread through the second and third years of the bachelor's degree and also the first year of the master's degree.

5.3. Method

The students were exposed to the tool in two ways. First, some students that were taking the 'Introduction to Programming' class used the tool with some guidance of the course coordinator (professor Nuno Garrido), essentially on how to access and use the tool. In this case, each student used a computer in the classroom to access the tool online and then followed the flow described in the previous chapter. The code submitted by each student represented a solution to a particular problem given previously by the course teacher. However, the prototype was still at a very incomplete stage so there were some unexpected errors and only a few code and answer submissions.

Later, the prototype was refined and stabilised and there was a more proper planning for the conduct of future experiments. We then asked other students, through university email, to participate in the experiment, out of class hours. We also asked them to bring code solutions to particular problems in advance so the time used in the sessions was exclusively for code and answer submissions. A couple of sessions were done involving more or less four or five students at a given time. There was a bit more adherence through this method. We provided guidance to these students, through the Zoom Cloud Meetings [9] videoconferencing software, on how to use the tool and on how to get past any errors that might occur. First we showed a demo of the prototype, by accessing it online, submitting a code sample and the answers to the generated questions. Afterwards, students did the same process with the code they prepared beforehand while we addressed any problems that were encountered with the tool's usability.

5.4. Results

In total, 185 questions and answers were generated and submitted, respectively. Of these questions, 144 are static and 41 are dynamic. Figure 13 shows the percentage of correct and incorrect answers to static and dynamic questions, respectively.

[9] `https://explore.zoom.us/en/products/meetings/`

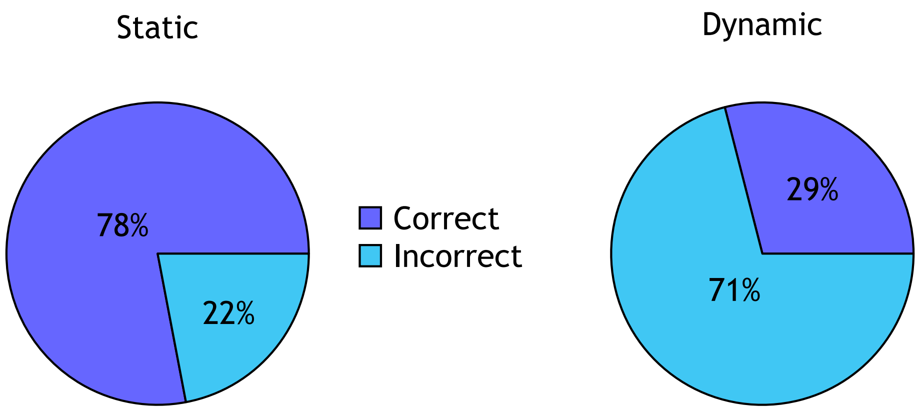


Figure 13. Correct and incorrect answers to questions by type

Approximately 78% of static questions were answered correctly while only about 29% of dynamic questions were answered correctly. Overall, students answered questions correctly about 68% of the time.

Table 3 shows the correct answers ratio by question template. As it can be observed, there is a considerably higher correct answer ratio present in static questions compared to dynamic questions. The only exception is the *WhatIsTheReturnValue* question template. This is because this template has the lowest amount of generated questions, with a total count of 2, which means there was one correct answer and one incorrect answer. This is mainly due to the nature of the code that was submitted in the classes since it consisted in having random arrays and matrices as return values. This means that this type of question was not generated or was generated and the result was discarded as the student had no way of knowing the result.

Figure 14 shows the relation between the students' performance and their degree and year of study. Ideally, senior students should have the highest percentage of correct answers. However, while the worst performances belong to first year students, the best performances do not belong to the most senior students necessarily. It can be observed that a few students from the first and second years had a better performance than all other students, including students studying for a master's degree.

Table 3. Total and percentage of correct answers by question template

Question Template	Total answers	Percentage of correct answers
HowManyVariableAssignments	13	15%
HowDeepCallStack	8	25%
WhatVariableValues	13	38%
HowManyMethodCalls	5	40%
WhatFixedValueVariables	14	43%
WhatVariables	14	43%
WhatIsTheReturnValue	2	50%
HowManyVariables	14	64%
WhatParameters	14	64%
WhichVariableHoldsReturn	14	86%
WhatIsTheVariableRole	9	89%
HowManyLoops	14	93%
IsRecursive	14	93%
CallsOtherFunctions	13	100%
HowManyFunctions	5	100%
HowManyParams	14	100%
WhatFunctions	5	100%

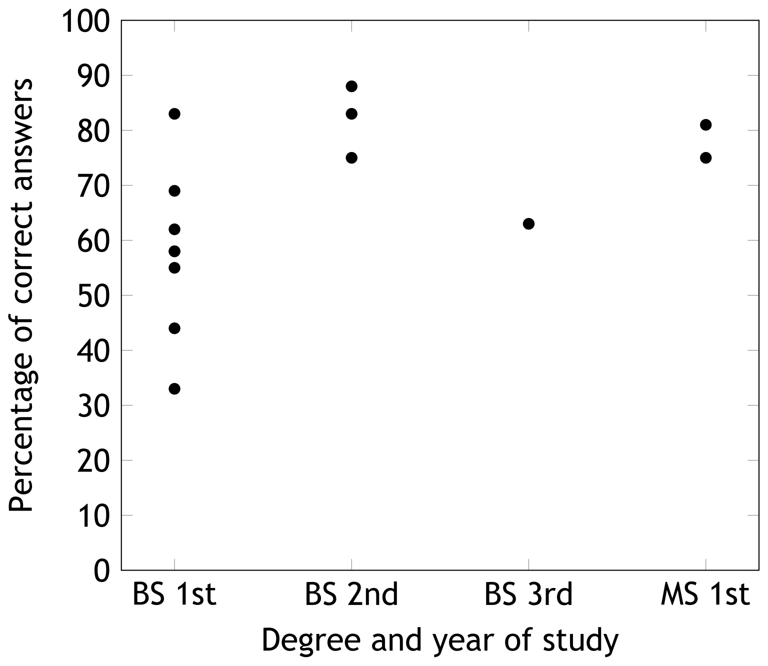


Figure 14. Percentage of correct answers by degree and year of study

5.5. Discussion

The discrepancy in Figure 13 is to be expected since dynamic questions require the student to have a better understanding of how their code works instead of just pointing out names of elements present in their code, like in static questions. This makes dynamic questions naturally harder to answer. The difficulty discrepancy between static and dynamic questions is even more apparent when drilling through the data and inspecting answers to each question.

When considering Table 3, among static questions, there is a group of related questions that had a considerable lower percentage of correct answers. This group consists in the following question templates: *HowManyVariables*, *WhatVariables* and *WhatFixed-ValueVariables*. The first question of the group asks the student the number of variables that exist in a given function. The problem with this question is that some students did not consider certain variable declarations in their code, especially iterator variables declared in *for loop* constructs. Unlike the previously discussed dynamic question, this question was asked to each of the 14 students (typically variables exist in any program) which means that 5 students answered incorrectly.

The *WhatVariables* question template asks the student what variables are present in the code. It only differs from the the previous question in the sense that the user not only has to count the number of variables but also identify their names which makes this question somewhat harder and therefore resulting in a lower percentage of correct answers. This question was also posed to all students and 9 students answered incorrectly. Compared to the previous question, the additional 3 students that got this question wrong named the parameters of the function instead of variables present in loop constructs, even though the question mentions that parameters should not be included in the answer (although they are indeed variables). This shows that these students answered correctly to the *HowManyVariables* question by accident and that the fact that the number of parameters was the same as the number of variables declared in loop constructs, was a lucky coincidence. It also shows that some students do not read or understand questions completely since the information about not including parameters in the answer was ignored.

Regarding the *WhatFixedValueVariables* question template, it was also asked to all students and 9 of them answered incorrectly. There seems to be a few different reasons for this. The first is that some students did not consider parameters that did not change their value as fixed value variables. Another reason is related to assigning values to array indices. Some students assumed that by assigning values to array indices the array itself is being mutated and do not consider the array variable a fixed value variable in this case. Lastly, some students simply did not know what the question meant and replied with '0' or 'I do not know'.

Regarding the dynamic questions, most students that answered the *HowManyMethodCalls* question seemed to only count the number of functions called directly by the function that the question refers to. They did not take into consideration calls that were made in other functions, caused indirectly by the first call. This might be considered an interpretation problem and not necessarily a lack of understanding on the code's flow execution. This question was posed and answered 5 times, with only 2 students submitting a correct answer.

The *WhatVariableValues* and *HowManyVariableAssignemnts* question templates have similar reasons for having a low percentage of correct answers mainly due to the fact that some the generated arguments to the question had high values and these were used as upper value limits of iterator variables in nested loop constructs. As such, inside these loop constructs some variables were assigned tens of times which probably made the respective questions tedious to answer. In the case of the *HowManyVariableAssignments* question some students also did not consider the line where a variable is both declared and assigned as an assignment. Therefore many students missed the correct answer just by one count. Only 2 students out of 13 answered the *HowManyVariableAssignments* question correctly and 5 out of 13 answered the *WhatVariableValues* question correctly.

Finally, the *HowDeepCallStack* question template was asked 8 times and answered correctly only 2 times. It is not obvious what the incorrect answers submitted by the students mean as the values are, most of the time, very different from the expected ones. The data indicates an overall lack of familiarisation with the concept of a call stack. One student answered one value below the correct call stack depth, possibly indicating that he or she did not assume that the function that starts the call stack counts to its depth.

Overall, taking the question templates with the highest correct answer percentage into account it can be observed that students do not have problems identifying function calls in their code and how many parameters they have. However, while all students could identify the number of a parameters of a given function some did not understand what is the definition of a parameter. Some students ended up answering to the *What-Parameters* question with both the type and the name of the parameter and others answered with the parameter's initial value, i.e. the argument.

Regarding Figure 14 it is hard to take meaningful conclusions from this since the student sample size is small but it is safe to say that senior students do not necessarily know how to program better than junior students.

CHAPTER 6

Conclusions and Future Work

We demonstrated that the idea of generating questions automatically based on student submitted code is achievable. Nonetheless, some problems and limitations were found. The system has a low count of dynamic questions and more of these could be added, for instance, questions that relate to loop iterations. In addition, this prototype can only ask questions about introductory programming code that is not using complex objects or non-standard language methods, written in the Java programming language. This is due to the fact that this prototype relies entirely on the 'Paddle' framework for static analysis and dynamic execution and only focuses on creating an engine that generates questions. In this way, the prototype itself can be improved in order to handle these cases.

Nonetheless, researchers from Aalto University (Finland), including the main author of (Lehtinen, 2021), are currently testing a prototype in an algorithms course that is based on the question engine developed in this work. Their course uses Python as the programming language, but given that the question engine is built on 'Paddle', they could use our question engine by developing a Python translator to obtain 'Paddle' models. The prototype as a whole will also be used in the 'Introduction to Programming' classes at the ISCTE-IUL university as a small part of the course's evaluation process.

Another research question was to understand to what extent could students correctly answer questions about the properties of their own code. While poorer performances seem to come from first year students, the best performances do not necessarily come from the most senior students as some students from the first and second years scored higher than all others. Also, while most students can grasp simple details about their own code (i.e., what function calls are present inside another function, how many parameters a function has, etc.) most students cannot understand its behaviour, meaning that they fail to keep up with the program's state (i.e., variable values and assignments, call stack depth, etc.), which indicates a lack of conceptual knowledge in the subject. Moreover, a designed experiment with homogeneous groups and larger sample sizes, for each given year of study, would provide more reliable results. We believe that the difficulty in recruiting volunteers for the study was partially justified by the pandemic context of the past academic year.

References

[1] K. Brennan and M. Resnick, ‘New frameworks for studying and assessing the development of computational thinking’, in *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, vol. 1, 2012, p. 25.

[2] J. Salac and D. Franklin, ‘If they build it, will they understand it? exploring the relationship between student code and performance’, in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2020, Trondheim, Norway, June 15-19, 2020*, M. N. Giannakos, G. Sindre, A. Luxton-Reilly and M. Divitini, Eds., ACM, 2020, pp. 473–479. doi: 10.1145/3341525.3387379. [Online]. Available: https://doi.org/10.1145/3341525.3387379.

[3] C. Kennedy and E. T. Kraemer, ‘Qualitative observations of student reasoning: Coding in the wild’, in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, Aberdeen, Scotland, UK, July 15-17, 2019*, B. Scharlau, R. McDermott, A. Pears and M. Sabin, Eds., ACM, 2019, pp. 224–230. doi: 10.1145/3304221.3319751. [Online]. Available: https://doi.org/10.1145/3304221.3319751.

[4] H. Keuning, J. Jeuring and B. Heeren, ‘A systematic literature review of automated feedback generation for programming exercises’, *ACM Trans. Comput. Educ.*, vol. 19, no. 1, 3:1–3:43, 2019. doi: 10.1145/3231711. [Online]. Available: https://doi.org/10.1145/3231711.

[5] T. Lehtinen, A. L. Santos and J. Sorva, ‘Let’s ask students about their programs, automatically’, in *29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*, IEEE, 2021, pp. 467–475. doi:

10.1109/ICPC52881.2021.00054. [Online]. Available: https://doi.org/10.1109/ICPC52881.2021.00054.

[6] K. Peffers, T. Tuunanen, C. E. Gengler, M. Rossi, W. Hui, V. Virtanen and J. Bragge, ‘Design science research process: A model for producing and presenting information systems research’, *CoRR*, vol. abs/2006.02763, 2020. arXiv: 2006.02763. [Online]. Available: https://arxiv.org/abs/2006.02763.

[7] Y. Qian and J. Lehman, ‘Students’ misconceptions and other difficulties in introductory programming: A literature review’, *ACM Trans. Comput. Educ.*, vol. 18, no. 1, 1:1–1:24, 2017. doi: 10.1145/3077618. [Online]. Available: https://doi.org/10.1145/3077618.

[8] S. Derus and A. M. Ali, ‘Difficulties in learning programming: Views of students’, in *1st International Conference on Current Issues in Education (ICCIE 2012)*, 2012, pp. 74–79.

[9] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting and T. Wilusz, ‘A multi-national, multi-institutional study of assessment of programming skills of first-year cs students’, in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR ’01, Canterbury, UK: Association for Computing Machinery, 2001, pp. 125–180, isbn: 9781450373593. doi: 10.1145/572133.572137. [Online]. Available: https://doi.org/10.1145/572133.572137.

[10] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon and L. Thomas, ‘A multinational study of reading and tracing skills in novice programmers’, *ACM SIGCSE Bull.*, vol. 36, no. 4, pp. 119–150, 2004. doi: 10.1145/1041624.1041673. [Online]. Available: https://doi.org/10.1145/1041624.1041673.

[11] I. Gomes, P. Morgado, T. Gomes and R. Moreira, ‘An overview on the static code analysis approach in software development’, *Faculdade de Engenharia da Universidade do Porto, Portugal*, 2009.

[12] M. Striewe and M. Goedicke, ‘A review of static analysis approaches for programming exercises’, in *Computer Assisted Assessment. Research into E-Assessment -*

International Conference, CAA 2014, Zeist, The Netherlands, June 30 - July 1, 2014. Proceedings, M. Kalz and E. Ras, Eds., ser. Communications in Computer and Information Science, vol. 439, Springer, 2014, pp. 100–113. doi: 10.1007/978-3-319-08657-6_10. [Online]. Available: https://doi.org/10.1007/978-3-319-08657-6%5C_10.

[13] K. Rivers and K. R. Koedinger, 'Automatic generation of programming feedback; A data-driven approach', in *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013, Memphis, USA, July 9-13, 2013*, E. Walker and C.-K. Looi, Eds., ser. CEUR Workshop Proceedings, vol. 1009, CEUR-WS.org, 2013. [Online]. Available: http://ceur-ws.org/Vol-1009/0906.pdf.

[14] J. Stamper, T. Barnes, L. Lehmann and M. Croy, 'The hint factory: Automatic generation of contextualized help for existing computer aided instruction', in *Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young Researchers Track*, 2008, pp. 71–78.

[15] S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare and A. Bhattacharya, 'Automatic grading and feedback using program repair for introductory programming courses', in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2017, Bologna, Italy, July 3-5, 2017*, R. Davoli, M. Goldweber, G. Rößling and I. Polycarpou, Eds., ACM, 2017, pp. 92–97. doi: 10.1145/3059009.3059026. [Online]. Available: https://doi.org/10.1145/3059009.3059026.

[16] V. J. Marin, T. Pereira, S. Sridharan and C. R. Rivero, 'Automated personalized feedback in introductory java programming moocs', in *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, IEEE Computer Society, 2017, pp. 1259–1270. doi: 10.1109/ICDE.2017.169. [Online]. Available: https://doi.org/10.1109/ICDE.2017.169.

[17] F. Alfredo, 'Avaliador pedagógico da qualidade de código', M.S. thesis, Iscte - Instituto Universitário de Lisboa, 2020. [Online]. Available: http://hdl.handle.net/10071/22052.

[18] L. Zavala and B. Mendoza, ‘On the use of semantic-based AIG to automatically generate programming exercises’, pp. 14–19, 21st Feb. 2018. doi: 10.1145/3159450.3159608. [Online]. Available: https://dl.acm.org/doi/10.1145/3159450.3159608 (visited on 07/12/2020).

[19] A. Thomas, T. Stopera, P. Frank-Bolton and R. Simha, ‘Stochastic tree-based generation of program-tracing practice questions’, in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE 2019, Minneapolis, MN, USA, February 27 - March 02, 2019*, E. K. Hawthorne, M. A. Pérez-Quiñones, S. Heckman and J. Zhang, Eds., ACM, 2019, pp. 91–97. doi: 10.1145/3287324.3287492. [Online]. Available: https://doi.org/10.1145/3287324.3287492.

[20] *Domain-driven design - tackling complexity in the heart of software*. Addison-Wesley, 2004, isbn: 978-0-321-12521-7.