Imperial College London

BEng Individual Project - Final Report

jSCAPE - Java Self-assessment Center of Adaptive Programming Exercises

Author:
Alexis Chantreau

Supervisor:
Dr. Tristan Allwood

June 17, 2014

Abstract

Programming is generally acknowledged to be a difficult discipline to learn, requiring problem solving skills, attention to detail and the ability to think abstractly. Yet, there is a definite interest in this subject as demonstrated by the number of students enrolling in computer science courses.

In these courses, it is usually difficult for teachers and students to get an idea of how they are performing as they can only rely on a few assignments. This project aims to address this issue, by providing a solution which increases the amount of practise and feedback available to students and teachers.

The result is a complete product in the form of jSCAPE (Java Self-assessment Center of Adaptive Programming Exercises) with core features consisting of providing programming exercises, displaying performance statistics, adapting the difficulty of exercises and generating exercises automatically.

We used a difficulty category based algorithm, as well as concepts presented in Item Response Theory to adapt the difficulty of exercises to the student's ability.

Acknowledgements

I would like to thank Dr. Tristan Allwood for proposing this project as well as providing guidance and support throughout. I would also like to thank my personal tutor, Prof. Duncan Gillies, for overseeing my time at Imperial. Last, but not least, I would like to thank my family, especially my parents for supporting me and always being there for me.

Contents

List of Figures

List of Code Listings

Chapter 1

Introduction

1.1 Motivation

Programming is generally acknowledged to be a difficult discipline to learn, requiring problem solving skills, attention to detail and the ability to think abstractly. One could say that these skills are somewhat developed in high school during mathematics courses, but programming is still a "beast" of its own. In addition, different programming paradigms exist, such as functional programming or imperative programming. Knowing how to program in Java can still make learning Haskell a difficult process.

Yet, in this 21st century society, programming is definitely a useful skill to have, and there is an interest in the population to learn these skills. Indeed, this can be seen by the number of students enrolling in computer science courses at universities, or the increase in websites such as Codecademy[1], Coursera[2], Udacity[3], which provide online computer science/programming courses for free.

In many of these situations, it is difficult for teachers, lecturers or course leaders to provide enough support through exercises, assignments and to monitor student's progress in such a way which allows them to modify their teaching to help struggling students. Indeed, teachers and students can generally only rely on a few homework assignments to get an idea of how they are doing. Coming up with a solution to increase the amount of practise and feedback would be beneficial to both students and teachers.

Thus, there is a clear need to provide a platform for students to practise their programming skills and understanding of programming concepts, in a

context of self-assessment only. In such a system, requiring teachers to come up with all the exercises by themselves can be both time consuming and ineffective: some exercises may not be challenging enough for certain top students, or on the contrary too difficult for struggling students, which can be discouraging for them.

1.2 Objectives

Having identified the problems associated with teaching and learning programming, we were led to formulating objectives in order to make the project successful and useful to the parties involved.

The main objective of the project was to produce a web-based application to be used in self-assessing one's programming knowledge, whether it be in high school, at university or as part of an online course. For this application, four key features were identified:

- **Programming questions/exercises -** The web platform should allow students to practise their programming skills and understanding of programming concepts. There should be no limit to the number of questions a student can answer, so that if a student desires more practice, then he should be able to do that. Additionally, it should be possible for a specific set of people, such as teachers, lecturers or tutors, to add questions/exercises to the system.

- **Progress tracking -** Designated people, such as teachers, lecturers or tutors, should have access to detailed statistics about the students performances. This will provide them with useful information about difficulties particular students, or the entire class, may be facing. In addition, the system should give feedback to the students, in the form of simple statistics, allowing them to identify their weak areas and thus improve on them.

- **Adaptive difficulty -** The questions or exercises presented to the students should be suited to their ability. Not only will this stimulate the learning process, but it will also give a better indication of a student's understanding of the programming concepts being tested.

- **Automated generation -** There should be a mechanism to allow for some degree of automated or semi-automated generation of exercises. This will provide a large supply of "fresh" questions, so that students

don't end up answering the same questions and memorizing the answers to them.

While investigating existing solutions (chapter 3) we found out that some of these features were less common than others. The availability of programming exercises and progress tracking are very essential in such systems, therefore many of the related software we looked at implemented these features. On the other hand, relatively few tools integrated some form of adaptive difficulty. Finally, almost none of the tools featured automated generation of questions, opting instead to allow exercises to be added manually to the system, or downloaded from existing exercise banks.

1.3 Contributions

Within the context given above, this project makes the following contributions:

- **jSCAPE**: a web application for students, named Java Self-assessment Center of Adaptive Programming Exercises, with the following features:
  - the ability to view programming exercises and answer them while receiving feedback, a so called form of self-assessment.
  - graphs, tables and pie charts displaying statistical data on the student's performance.
  - three different exercise selection algorithms, that decide which is the best exercise to present to the student.
- **Admin tool**: a tool for teachers/lecturers/tutors for:
  - displaying student performance statistics.
  - displaying exercise statistics.
  - defining exercise categories.
  - adding exercises manually.
  - generating exercises automatically.

1.4 Report Structure

Chapter 2 will present the theoretical basis for this project and the concepts necessary to follow the implementation details of the system.

Chapter 3 will give an overview of related work, and will mention how these influenced the design of jSCAPE, in particular which features would be useful for such as system.

Chapter 4 will present the jSCAPE system in detail, showing all the features which are available.

Chapter 5 will cover the design and implementation details of the jSCAPE system, in particular how the difficulty of exercises is adapted to the student's ability and how exercises can be automatically generated.

Chapter 6 will contain the results of evaluating jSCAPE as a whole, and its different components.

Chapter 7 summarises the achievements of the project and discusses possible extensions that can be made to the system in the future.

Chapter 2

Background

This section gives an overview of the theoretical basis for this project. We start off with discussing Computer Based Tests (section 2.1) along with some of their advantages and disadvantages. This leads us to considering an improvement in the form of Computerized Adaptive Testing (section 2.2). Finally, we present probabilistic concepts (section 2.3) and how they can come together to implement an effective computer based assessment framework.

2.1 Computer Based Tests

The increase in the usage of computers in today's society has quite naturally made its way to education and assessment. As such, high schools, universities and companies now have the possibility of using computer based tests to assess students or employees in their knowledge of particular subjects.

Computer based tests (CBT) offer several advantages. Indeed, the test offers more possibilities in terms of question variety because a computer can display images, videos, graphs, etc... In a simple pen and paper test, videos or animations can't be displayed, and images or graphs are often of lesser quality. Students are becoming increasingly familiar with technology, especially computers and web browsers, therefore this doesn't present many issues in administering CBTs to students.

These types of test are advantageous for teachers because everything is stored on the computer, which means low paperwork, automatic grading and the possibility of easily generating statistics from this data.

CBTs are typically "fixed-item" tests where all the students answer the same

set of questions, usually provided by the person responsible for the assessment. This isn't ideal since students can be presented with questions that are too easy or too difficult for them to answer. Consequently, the results of the test won't give a very accurate representation of a student's ability, and for this reason, these types of tests aren't extremely useful. This problem lead to the research and the development of computerized adaptive testing (CAT).

2.2 Computerized Adaptive Testing

Computerized adaptive testing (CAT), also called *tailored testing*, is a form of computer-based testing which administers questions (referred to as *items* in the psychometrics literature) of the appropriate difficulty by adapting to the examinee's ability. For example, if an examinee answers an item correctly, then the next item presented will higher on the difficulty scale. On the other hand, if they answer incorrectly, they will be presented with an item lower on the difficulty scale.

From an architectural perspective, a computerized adaptive test (CAT) consists of five components [6]:

1. Calibrated item pool

An item pool is needed to store all the items available for inclusion in a test. This item pool must be calibrated with a psychometric model. During this phase, the item parameters are estimated according to the chosen model and scaled to fit with already existing items. Usually, the psychometric model employed in these systems is called Item Response Theory (IRT) (section 2.3.4). Calibration is a complex process, and to be done accurately it requires a considerable amount of data. Typically, it is performed by psychometricians, aided by expensive and sophisticated calibration software.

2. Starting point

Initially, when zero items have been administered, no information is known about the examinees and so the CAT is unable to estimate their ability. As a result, the item selection algorithm will fail to choose the next item to be administered. If there is previous information available, for example an examinee's ability estimate in a closely related subject, then this can be input into the system to form the starting point configuration. Often this data isn't available or too costly to collect, so the CAT's initial ability estimate for the

examinee corresponds to the mean on the ability scale - hence the first item presented will be of average difficulty.

3. Item selection algorithm

The item selection algorithm chooses the next item to present to the examinee based on the ability estimate of the examinee up to that point. Several methods exist and largely depend on the psychometric model in use. One of the most commonly used methods is the *maximum information method* (section 2.3.4), which selects the item which maximizes the information function with respect to the estimated ability at that point.

4. Scoring algorithm

The scoring algorithm refers to the steps taken to update the examinee's ability estimate after an item has been answered. The two most commonly used methods are *maximum likelihood estimation* (section 2.3.4) and *Bayesian estimation* (section 2.3.4).

5. Termination criterion

The termination criterion specifies when the CAT should finish. For example the CAT can terminate when the change in the ability estimate after each iteration is below a certain threshold, or when time has run out, or when N items have been administered, etc... Obviously, the CAT shouldn't be terminated too early, so as to allow enough time to estimate the examinee's ability with acceptable accuracy.

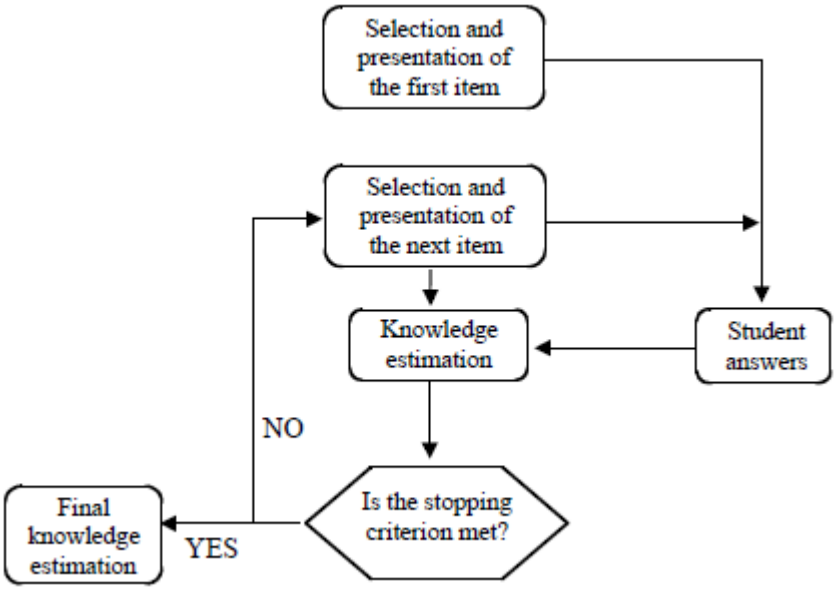

Figure 2.1: Flowchart of an adaptive test. Adapted from [4].

The flowchart in figure 2.1 corresponds to components 2-5, and illustrates the basics of the algorithm implemented in CAT. [5] gives a more detailed description of the procedure:

1. The pool of items that haven't been administered yet is searched to determine the best item to present to the examinee, according to the current estimation of his ability.

2. The chosen item is presented to the examinee, who then answers it correctly or incorrectly.

3. The ability estimate is updated, based upon this new piece of information and the previous ability estimate.

4. Steps 1–3 are repeated until a termination criterion is met.

5. The algorithm returns a final ability estimate for the examinee's performance along with a confidence level: a percentage value indicating how accurate the estimate is.

CATs offer several advantages over traditional CBTs. As a result CATs have been used in many areas[7], such as education, job hiring, counselling, clinical studies, etc... Since CATs administer items by adapting to the examinee's ability, the test-taking experience ends up being a more positive one. Indeed,

examinees won't have to deal with answering items which are too difficult or too easy compared to their ability level, a problem which appears in traditional CBTs.

In addition, by administering only those items which will yield additional information, CATs end up being more accurate in estimating an examinee's ability level. This contrasts with CBTs which usually provide the best precision for examinees of medium ability, whereas extreme scores end up being less accurate.

Lastly, CATs can come up with an ability estimate much quicker and with fewer administered items when compared to traditional CBTs. Indeed, an adaptive test can typically be shortened by 50% and still maintain a higher level of precision than a fixed version.[8]

Despite the advantages mentioned above, CATs have some limitations. A frequent complaint is that an examinee isn't allowed to go back and change his answer to a past item. This limitation exists to prevent the examinee from intentionally answering items incorrectly to make subsequent items easier, and then going back and selecting the correct answers to achieve a perfect score. For similar reasons, it isn't possible to skip items, the examinee must select an answer to move on to the next item.

The second issue has to do with the items themselves. First of all, there is the need for a large bank of items to cater to all ability levels. Developing an item pool of sufficient size can be very time consuming. David J. Weiss writes in [9] that item pools with 150-200 items are to be preferred, although 100 high quality items can sometimes be enough to achieve adequate estimations of ability levels.

Secondly, for the CAT to be of good quality the item pool needs to be calibrated accurately. This requires pre-administering the items to a sizeable sample and then simultaneously estimating all the item parameters for each item. The guidelines in [10] suggest that sample sizes may be as large as 1000 examinees. This phase is costly, time consuming and often times simply unfeasible.

Lastly, item exposure is a possible security concern. Sometimes particular items may be presented too often and become overused. This may result in examinees becoming familiar with them and sharing them to other examinees of the same ability level, thus corrupting the results of the test. This problem

can be solved to some extent by modifying the item selection algorithm to include some exposure control mechanism.

A brief overview of CATs was given in this section. All of these concepts will be explored in more detail in item response theory (section 2.3.4) and in the implementation of adaptive testing in jSCAPE (section 5.4).

2.3 Probabilistic Test Theory

In the previous section we listed some of the components necessary for the development of CATs. Many of these components, especially the item selection and scoring algorithms, rely heavily on probabilistic concepts. Therefore, in this section we go over a few topics in probability and how they can be implemented in a psychometric model to be used in computerized adaptive testing.

2.3.1 Probability Theory

Probability theory provides us with a means to model uncertainty in data and to infer information from observed data. This is especially useful when it comes to estimating latent variables, i.e. variables that are not directly observed, instead they are inferred from other observed variables. For instance, examinee ability is a latent variable, hence the reason for section 2.3.

The probability of an event E occurring is a numerical value between 0 and 1, indicating how probable it is that we will observe event E. It is denoted as $P(E)$ and $0 < P(E) < 1$. The value 1 indicates total certainty, whereas the value 0 indicates impossibility.

In addition, there is the concept of conditional probability where the probability of an event E_1 occurring, given that event E_2 has occurred, can be denoted as $P(E_1|E_2)$. This concept is also important in statistical inference because it allows us to update prior beliefs given additional observed data. This is explained in more detail in sections 2.3.2 and 2.3.3.

2.3.2 Likelihood and Maximum Likelihood Estimation

Although the terms probability and likelihood are used interchangeably in every day life, in statistics a distinction can be made.

For any stochastic process, let us denote the observed outcomes as x and the set of parameters as θ. When we say probability, we want to calculate $P(x|\theta)$, i.e. the probability of observing the outcomes x given specific values for the set of parameters θ.

However, sometimes we do not know the specific values for θ, instead, we have observed some outcomes x, and want to find out how likely a particular value of θ is given the observed outcomes x. We call this the likelihood or likelihood function, and it is denoted as $L(\theta|x)$. The likelihood of a set of parameter values, θ, given outcomes x, is equal to the probability of those observed outcomes given those parameter values, i.e. $L(\theta|x) = P(x|\theta)$.

To highlight the distinction we illustrate with an example of how the two terms are used. If we consider a dice, a possible parameter is the fairness of the dice, while possible outcomes are which values are displayed after a roll. For instance, if a fair dice is rolled 5 times, what is the *probability* that a 6 will show up on every roll? If a dice is rolled 5 times and lands on 6 every roll, what is the *likelihood* that the dice is fair?

Maximum likelihood estimation refers to a method of statistical inference where one can find the set of parameter values (θ) which are most *likely* given the observed outcomes (x). As mentioned in the name of the method, this is done by finding the parameter values which maximize the likelihood function $L(\theta|x)$.

2.3.3 Bayesian inference

Bayesian inference is a method of statistical inference where Bayes' theorem is used to update the probability estimate for a hypothesis after observing additional events or outcomes.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

Let us consider a small example: There is a 60% chance of it snowing on Thursday. If it snows on Thursday, there is a 20% chance of it snowing on Friday. If it didn't snow on Thursday, there is a 70% chance it will snow on Friday.

Before observing any additional data, we can say that the probability of it snowing on Thursday is 60%. But, we are given additional information, namely that it snowed on Friday. How we can we update the probability that

it snowed on Thursday to reflect this observation?

We can do this by using Bayes' theorem shown in equation (2.1). Let A be the event "snowing on Thursday" and B be the event "snowing on Friday". Then we have:

$$P(A|B) = \frac{0.2 \times 0.6}{0.2 \times 0.6 + 0.7 \times 0.4} = 0.3$$

This example shows how additional observed data can affect one's prior beliefs. The observation that it snowed on Friday reduced the probability that it snowed on Thursday from 60% to 30%.

However, in the context of this project, we are interested in Bayesian inference of the probability distributions of parameters.

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}$$

The prior distribution, $P(\theta)$, is the distribution of the parameters θ before any data is observed. $P(x|\theta)$ is the distribution of the observed data conditioned on the parameters. The posterior distribution, $P(\theta|x)$, is the probability distribution of the parameters after considering the new information brought by the observed data.

2.3.4 Item Response Theory

We mentioned in section 2.2 that Item Response Theory (IRT) is usually the psychometric model of choice when developing a CAT, e.g. the Graduate Record Examination (GRE) and Graduate Management Admission Test (GMAT). CATs can still be implemented with Classical Test Theory but they offer less sophistication and less information to evaluate/improve the reliability of the test, making IRT the superior choice.

In psychometrics, an item is a generic term used to refer to a question or an exercise. For instance, in a mathematics exam, "What is the square root of 81?" is a possible item.

Item Response Theory gets its name from focusing on analyzing the items themselves as opposed to Classical Test Theory, which considers the test as a whole. Indeed, Classical Test Theory judges an examinee's ability simply on the number of correct answers obtained, totally disregarding which items were answered correctly or incorrectly, thereby making the assumption that

all items possess identical properties.

IRT hinges on the idea that it is possible to model, as a mathematical function, the probability of a certain response to an item given the item parameters and the examinee's ability level. IRT is based on a set of strong assumptions[11]:

1. A unidimensional trait denoted by θ;
2. Local independence of items;
3. The response of a person to an item can be modelled by a mathematical *item response function* (IRF).

The unidimensionality of the trait means that the items are supposed to measure one characteristic of the person, generally their ability, and that this trait should account for most of the variance in the test score. The trait level is denoted as the Greek letter theta (θ), and typically it has a mean of 0 and a standard deviation of 1. For example, in the context of this project, theta (θ) will represent an examinee's ability level in a particular programming subject (e.g. arrays, binary trees, etc...). Ability is a latent variable (section 2.3.1) because it isn't directly observable, therefore it must be inferred from the observable, concrete data such as examinee responses.

The local independence of items states that an examinee's responses to items are independent of one another. This property of conditional independence is crucial to estimating examinee trait levels as we will see later on in this section.

Several IRT models have been developed over the years to address the different types of tests that exist, e.g. multiple choice exams, agreement questionnaires (Likert scale), etc... These models all seek to achieve the same goal, modelling the examinee's ability on some ability scale, but they differ in the number of parameters associated with each item. We shall note that we only consider unidimensional IRT models which deal with dichotomously scored items, i.e. scored as correct or incorrect, in a multiple choice question for instance. We now take a look at these different models in more detail.

The one-parameter logistic model

Every IRT model is defined by two elements: a set of item parameters, and a mathematical item response function. This function plots the probability

that an examinee of a given ability will answer a particular item correctly. The one-parameter logistic (1PL) model is the simplest IRT model because in this model items are only characterized by one parameter: the difficulty parameter b_i, where the subscript i identifies item i. This has the effect of making the item response function quite simple.

$$P_i(\theta) = \frac{1}{1 + \mathrm{e}^{-(\theta - b_i)}} \tag{2.2}$$

Equation (2.2) shows the item response function for the 1PL model. θ is the examinee's ability level, $P_i(\theta)$ is the probability of answering item i at all ability levels, and as mentioned above, b_i is the item difficulty.

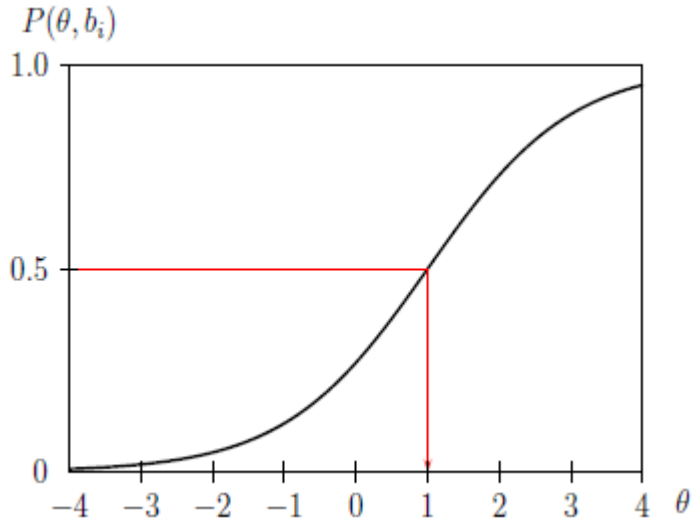

Figure 2.2: The item response function of the 1PL model. (Source:[13]).

Figure 2.2 shows the probability of answering an item correctly for all ability levels. In the 1PL model, the item difficulty is the point at which a correct response and an incorrect response are equally probable. In the figure this is shown in red, and in this case, the item has difficulty parameter $b_i = 1$. This highlights one of IRT's attractive properties, the fact that examinee ability and item difficulty are both measured on the same scale. The ability scale is a design choice, in this case it ranges from -4 to +4.

Ability (θ)	-4	-3	-2	-1	0	1	2	3	4
$P_1(\theta)$	0.007	0.018	0.047	0.119	0.269	0.5	0.731	0.881	0.953
$P_2(\theta)$	0.047	0.119	0.269	0.5	0.731	0.881	0.953	0.982	0.993

Table 2.1: Probabilities of a correct answer for two items in the 1PL model.

Table 2.1 shows the probability values for two 1PL items. $P_1(\theta)$ corresponds to the probability of answering item 1 (the item in figure 2.2) correctly. $P_2(\theta)$ corresponds to the probability of answering item 2 (with difficulty $b_2 = -1$) correctly. It can be seen that as the ability level of the examinee increases, so does the probability of him answering the item correctly. The difficulty parameter shifts the item response function to the left or to the right depending on its value.

Although the 1PL model provides a good basis for IRT it is rather simplistic and fairly limited. For instance, the model assumes that at low ability levels the examinee has close to no chance to answer the item correctly. However, in multiple choice questions there is always the option of guessing randomly. These existing limitations bring us to considering the next model.

The two-parameter logistic model

The two-parameter logistic (2PL) model builds upon the 1PL model by adding a new item parameter: the discrimination (a_i), where the subscript i identifies item i. Discrimination refers to an item's capacity to distinguish between examinees of different ability levels. Graphically, the discrimination parameter corresponds to the slope of the item response function, and typically it ranges from -2.8 to 2.8.

$$P_i(\theta) = \frac{1}{1 + \mathrm{e}^{-1.7a_i(\theta - b_i)}} \tag{2.3}$$

Equation (2.3) shows the item response function for the 2PL model. It is very similar to the equation of the 1PL model (Equation (2.2)), except for the inclusion of the discrimination parameter (a_i) and a constant of 1.7 to control the shape of the curve.

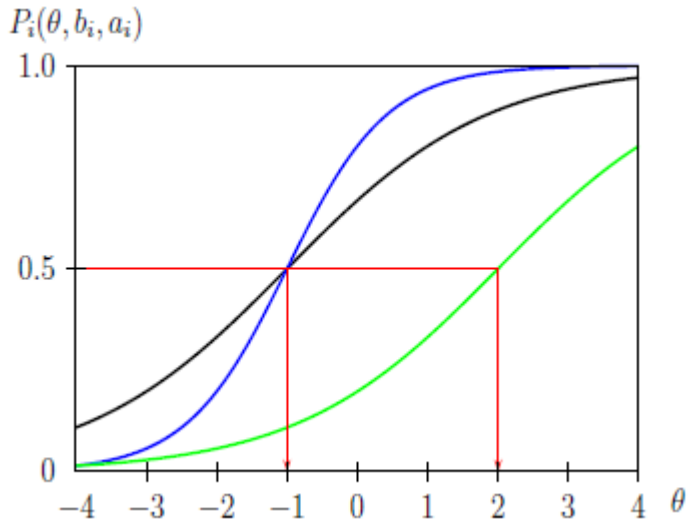

Figure 2.3: The item response functions of three items in the 2PL model. (Source:[13]).

Figure 2.3 shows the item response functions of three items in the 2PL model. As in the 1PL model, the item difficulty is still located where $P_i(\theta) = 0.5$, shown in the red lines. The item represented by the black curve and the item represented by the green curve both have the same discrimination, but different difficulty. As mentioned before, all this does is shift the item response function to the right or to the left, depending on which item you take as the reference point.

The black and blue curves illustrate the other scenario. The item represented by the black curve and the item represented by the blue curve have different discrimination, but identical difficulty. This has the effect of making the blue curve much steeper than the black one, thus affecting the probabilities over the ability scale. Indeed, the item represented by the blue curve discriminates quite well between respondents. Examinees of low ability have a much lower probability of answering the item correctly than examinees of higher ability. Compare this to the item represented by the black curve, where examinees of low ability still have a fair chance of getting the item correct.

The three-parameter logistic model

The three-parameter logistic (3PL) model takes into account the possibility of guessing a correct response to an item. This model is especially convenient for tests where multiple choice questions are present. In IRT, this parameter

is called the pseudo-guessing or chance parameter, and it is denoted as c_i, where the subscript i identifies item i. Graphically, it corresponds to a lower asymptote, i.e. $P_i(-\infty) = c_i$. The guessing parameter does not vary as a function of the examinee's ability. Indeed, whether they be of low ability or high ability, examinees both have the same probability of guessing the correct answer to an item.

$$P_i(\theta) = c_i + (1 - c_i)\frac{1}{1 + \mathrm{e}^{-1.7a_i(\theta - b_i)}} \tag{2.4}$$

Equation (2.4) shows the item response function for the three-parameter logistic model (3PL). The addition of c_i defines a lower asymptote at that value, whereas $1 - c_i$ acts as a weighting factor towards the 2PL model equation (Equation (2.3)). Typically the pseudo-chance parameter ranges from 0 to 0.35, greater values being considered unacceptable. Lastly, if $c_i = 0$ then we are simply left with a normal 2PL model.

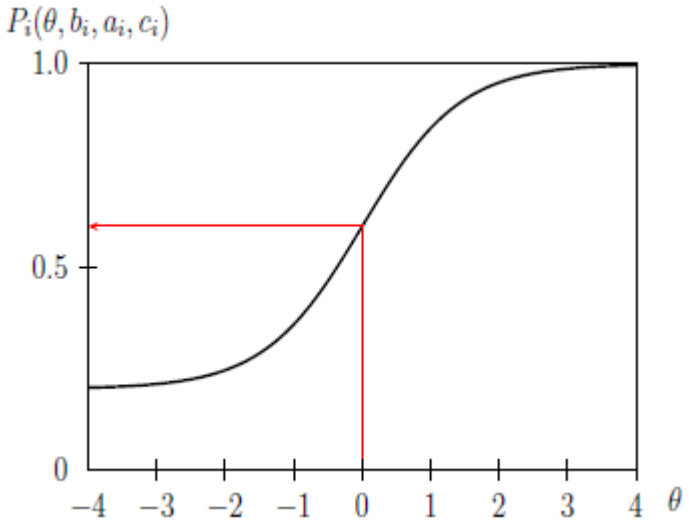

Figure 2.4: The item response function of the 3PL model. (Source:[13]).

Figure 2.4 shows the item response function of an item in the 3PL model. This item has parameters $a = 1.4$, $b = 0$ and $c = 0.2$. Now it can be seen that the guessing phenomenon is taken into account by this IRT model. Indeed, at the lowest possible ability ($\theta = -4$), the probability of a correct answer is $P(-4) = 0.200059$, which is very close to the guessing parameter of the item.

Something to note is that $P_i(b_i) \neq 0.5$, i.e. the probability of a correct response at the ability level equal to the item difficulty is no longer 0.5.

Instead, we have $P_i(b_i) = c + \frac{1-c}{2}$, and in figure 2.4 we have $P(0) = 0.6$. If we consider a multiple choice question, adapted to the examinee's ability level, then it makes sense that the probability of them getting it correct would be greater than 0.5, to account for the possibility of getting it right merely through guessing.

Item information

In section 2.2, when discussing item selection algorithms, we mentioned the maximum information method as a possible implementation. This method relies on the IRT concept of item information.

In psychometrics and statistics, the term *information* is defined as the reciprocal of the precision with which a parameter can be estimated[12]. Therefore, item information is the precision in the ability estimate that the item provides, at all ability levels. It is also an indication of the quality of the item in terms of how well that item can discriminate between several respondents.

The item information function for the 1PL model is calculated as

$$I(\theta) = P_i(\theta)Q_i(\theta),$$

where $Q_i(\theta) = 1 - P_i(\theta)$.

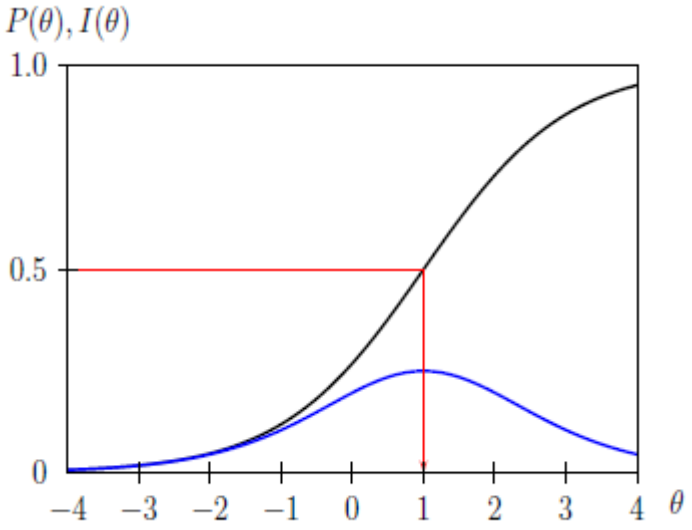

Figure 2.5: Item response function and item information function for the 1PL model. (Source:[13]).

In figure 2.5, two curves are plotted. The black curve is the item response function of an item, and the blue curve is the associated information function for that item. In the 1PL model, the maximum value of item information occurs where the probability of a correct answer and the probability of an incorrect answer are both equal to 0.5. This point is indicated by the red line. If we recall earlier, this point also represents the difficulty of the item, i.e. item parameter b_i. Thus, the item provides the most information for those examinees whose ability is equal to the difficulty of the item, 1 in this case. This means that administering this item will give more precise ability estimates for examinees whose true ability is at that particular level. Presenting this item to examinees not at that particular level, examinees of ability -2 for instance, will not yield very much precision to subsequent ability estimates.

The item information function for the 2PL model is calculated as

$$I(\theta) = a_i^2 P_i(\theta) Q_i(\theta),$$

where $Q_i(\theta) = 1 - P_i(\theta)$. In the 2PL and 3PL models, the discrimination parameter, a_i, plays a significant role in the function, as it appears squared in the function.

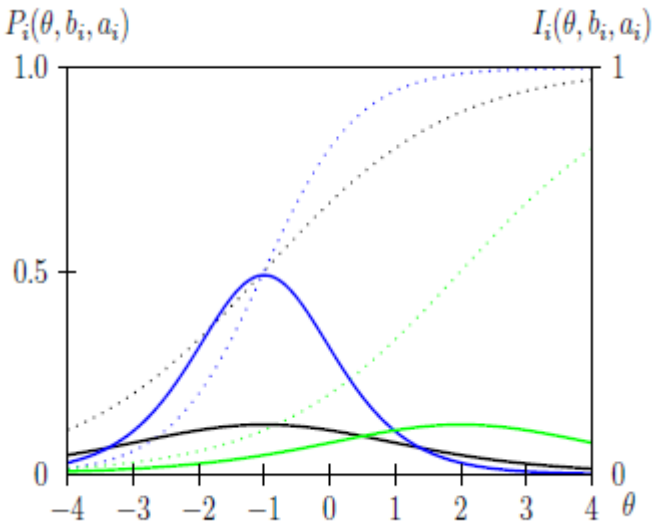

Figure 2.6: Item response functions and item information functions for three items in the 2PL model. (Source:[13]).

Figure 2.6 shows three 2PL item response functions, in dotted lines, matched in color with the corresponding item information functions in solid lines. Like

in the 1PL model, items still reach their maximum information at the point where ability is equal to the difficulty of the item. However, the amount of information provided at that ability level can now vary depending on how large the discrimination parameter is.

The items represented by the black and blue curves both have the same difficulty parameter. But, the item represented by the blue curve has a higher discrimination parameter, and therefore this affects the item information function. Higher discrimination values lead to more item information, whereas lower discrimination values make the item less informative.

The items represented by the black and green curves both have the same discrimination parameter, so the maximum information value will be the same for these items. However, these items do not share the same difficulty parameter. All this does is shift the curve along the ability axis, thus changing the location of maximum information for that item.

The item information function for the 3PL model is calculated as

$$I(\theta) = a_i^2 \cdot \frac{(P_i(\theta) - c_i)^2}{(1 - c_i)^2} \cdot \frac{Q_i(\theta)}{P_i(\theta)},$$

where $Q_i(\theta) = 1 - P_i(\theta)$.

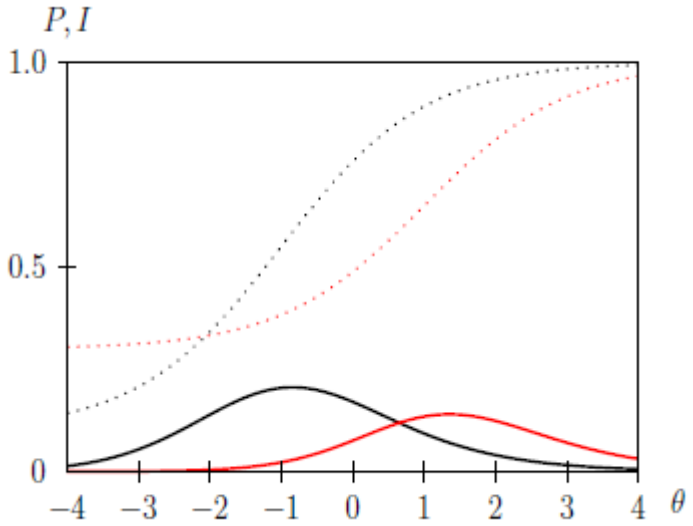

Figure 2.7: Item response functions and item information functions for two items in the 3PL model. (Source:[13]).

Figure 2.7 shows two 3PL item response functions, in dotted lines, matched in color with the corresponding item information functions in solid lines. In the 3PL model, the maximum of item information functions is no longer at the point where ability is equal to the difficulty of the item. Here, the guessing or pseudo-chance parameter (c_i) plays a role in the shape of the item information function. Higher guessing parameters lead to less item information, whereas lower guessing parameters lead to more item information. For instance, the item represented by the black curve has $c_i = 0.1$, and the item represented by the red curve has $c_i = 0.3$, which explains why the item represented by the black curve yields more information.

Estimating the ability

In section 2.2, when discussing scoring algorithms, we saw that the two most common methods for ability estimation were *maximum likelihood estimation* and *Bayesian estimation*.

In the *maximum likelihood estimation* method, we need to define a likelihood function in terms of the ability level (θ) we are trying to estimate:

$$L(\theta|\mathbf{u}) = L(\theta|u_1, ..., u_n) = \prod_{i=1}^{n} P_i(\theta)^{u_i}(1 - P_i(\theta))^{(1-u_i)},$$

where $\mathbf{u} = (u_1, ..., u_n)$ is called the response vector for an examinee, that is $u_i = 1$ if the examinee answers the i$^{\text{th}}$ item correctly, and $u_i = 0$ if the examinee answers the i$^{\text{th}}$ item incorrectly. $P_i(\theta)$ corresponds to the probability of answering the i$^{\text{th}}$ item correctly when the ability level is θ, and thus $1 - P_i(\theta)$ gives the probability of answering the i$^{\text{th}}$ item incorrectly when the ability level is θ.

Now that the likelihood function is defined, we can apply maximum likelihood estimation to find the ability level which is most likely given the examinee's response vector. Let us illustrate with a very simplistic example where ability levels are discrete values between -1 and $+1$. We would iterate through these ability levels and compute the following values, for example:

$$L(\theta = -1|\mathbf{u}) = 0.001$$
$$L(\theta = +0|\mathbf{u}) = 0.017$$
$$L(\theta = +1|\mathbf{u}) = 0.058$$

The likelihood function is maximized when $\theta = +1$, and so this is the maximum likelihood estimate for θ. In a CAT, and based on the examinee's

response vector $\mathbf{u}$, the system would assign $+1$ as the ability level for that examinee.

In the *Bayesian estimation* method, the CAT maintains a *knowledge distribution* $P(\theta)$, which represents the probability that the examinee's ability is θ. In the absence of any additional information, the CAT considers the examinee's initial knowledge distribution to be a uniform distribution.

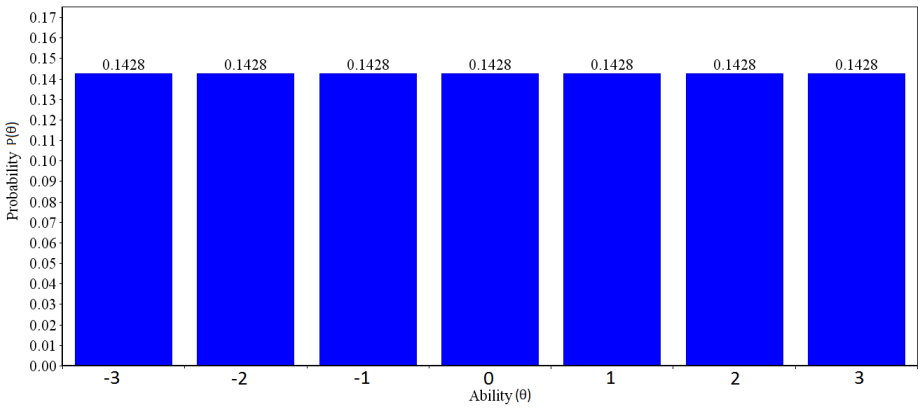

Figure 2.8: Initial examinee knowledge distribution.

For example, figure 2.8 shows the initial examinee knowledge distribution in a CAT having discrete ability levels ranging from -3 to 3. Continuous knowledge distributions are certainly possible as they allow for, in theory, infinitely more ability levels. However, they are more complex and require the use of integration to compute probabilities, thus for the purposes of illustration, we will only consider discrete knowledge distributions.

The *Bayesian estimation* method relies on Bayesian inference to derive the posterior knowledge distribution according to Bayes' rule:

$$P(\theta|\mathbf{u}) = \frac{P(\mathbf{u}|\theta)P(\theta)}{P(\mathbf{u})},$$

where, again, $\mathbf{u}$ is the examinee's response vector and θ corresponds to the possible ability levels.

The posterior distribution is the result of updating the prior knowledge distribution, $P(\theta)$, with observed data, $P(\mathbf{u}|\theta)$, i.e. the examinee's response pattern.

We can express this using the likelihood function:

$$P(\theta|\mathbf{u}) \propto L(\theta|\mathbf{u}) \cdot P(\theta)$$

After computing the posterior knowledge distribution, the CAT can estimate the ability estimation for the examinee. This is equal to the mode of the knowledge distribution, i.e. the ability associated with the highest probability value.

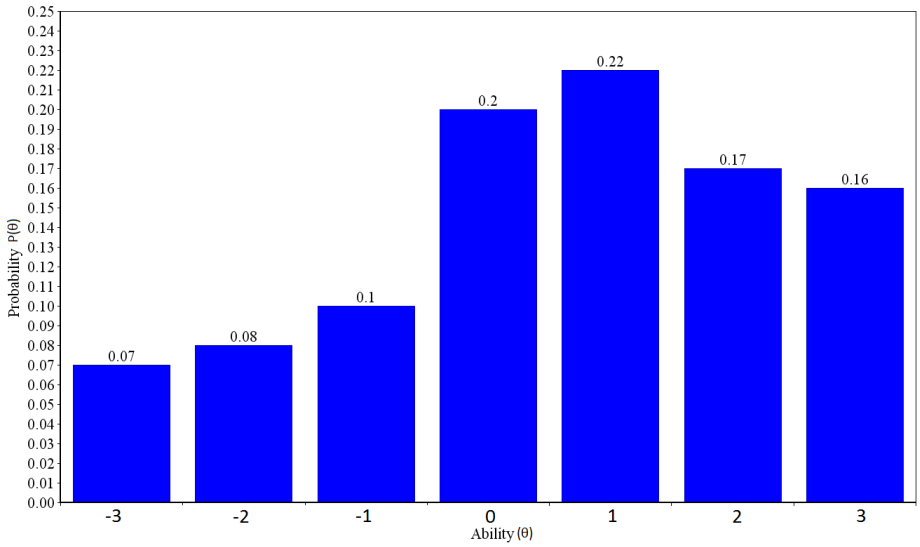

Figure 2.9: Examinee knowledge distribution after answering an item correctly.

For example, figure 2.9 shows the knowledge distribution of an examinee after being shown an item of medium difficulty and answering it correctly. It can be seen that lower ability levels have become less likely, whereas higher ability levels are now more probable.

The ability level associated with the highest probability is $\theta = 1$, thus this becomes the estimated ability. Over time, after the examinee answers more items, the estimation will become associated with even higher probability values. These probabilities indicate how confident the CAT is in that particular ability estimation.

2.4 Summary

In this section, the relevant background literature and theory for this project was discussed.

Firstly, we discussed types of computer based assessment and how they differ from regular assessment techniques. We then looked at an improvement over traditional computer tests in the form of computerized adaptive testing (CAT), where the difficulty of the questions in the test is adapted to fit the student's ability.

We provided an overview of the probability concepts necessary to understand Item Response Theory, and explained in quite some detail the techniques behind Item Response Theory. These will become the basis of one of the adaptive difficulty algorithms in jSCAPE. In the context of Item Response Theory, we have seen three models designed to calculate the probabilities of answering questions correctly, and knowledge estimation techniques in those particular models.

In the next chapter we take a look at related work which will help us in developing jSCAPE. We focus on how these works approach the design of some of the components of such a system, i.e. providing exercises, exercise generation, collecting statistical data, adapting the difficulty, etc...

Chapter 3

Related Work

With the rise of computer usage in today's society, they have quite naturally made their way over to education. As such, web-based, computer-based and adaptive assessment systems have been investigated and researched quite thoroughly over the years. The result of this research is numerous tools and software which provide students the opportunity to practise their programming skills, whether it be by writing code or answering programming questions, in an environment which tracks their progress. Almost all of these tools provide the basic features, that is the programming exercises themselves, and a component to view statistics or student results. Some of them distinguish themselves by adding useful and not so common features such as adaptive difficulty. From all the solutions we looked at, very few provided automate exercise generation, and for those who did, their approach was only limited to extremely simple exercises.

We have looked at quite a few solutions such as ELP (Environment for Learning to Program) and CourseMaster, which have the very basic features. We do not go into detail about how these solutions operate since the details are quite straightfowrad and not very interesting. Instead we focus on software which has approached the adaptive difficulty problem, and look into how they did so.

In this section we look at the related software which has influenced the design and implementation of our proposed solution, jSCAPE.

3.1 Programming Adaptive Testing

PAT[14] is a web-based adaptive testing system, developed in ActionScript/Flash, for assessing students' programming knowledge in Greek high schools.

The assessment is carried out by presenting students with 30 questions from various chapters of the introductory programming course. Some of the questions supported by the PAT system are true/false questions, multiple choice questions, gap filling in a piece of code, questions involving diagrams and questions where one has to determine the behaviour of a piece of code.

In PAT, questions are classified into different difficulty categories. Category A is for easy questions, category B is for intermediate questions and category C is for difficult questions. PAT seeks to adapt the difficulty of the questions to the student's ability, by choosing questions of the appropriate difficulty category.

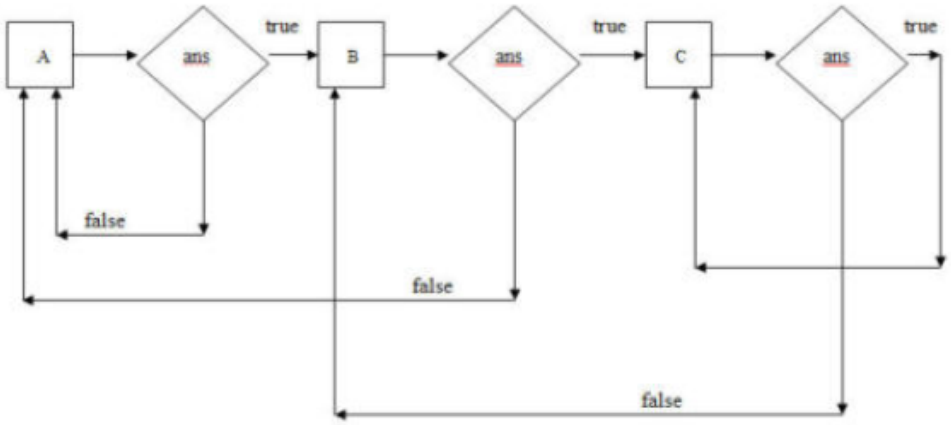

Figure 3.1: Adaptive sequence of questions in PAT. (Source:[14])

Figure 3.1 shows the possible adaptive sequences. This algorithm is quite simplistic, every correct answer leads to a promotion to the next level of difficulty until no further promotions are possible. Likewise, every incorrect answer leads to a demotion to a lower level of difficulty until no further demotions are possible.

At the end of the test, the system shows the student's total number of correct answers out of 30 and how well the student performed on each chapter. Also, PAT classifies the student into one of three programming skill levels based

on their final score and a weighted score, where the difficulty of the questions answered correctly is considered. These results are available to both students and teachers so that they can be used to improve performance later on in the school year.

We feel that the adaptive algorithm increases the difficulty of questions too quickly and doesn't take into account guessing or possible slip ups from students. This limitation can be circumvented by, for instance, requiring a number of correct answers at the current difficulty level before progressing to the next one.

PAT only provides assessment at specific times throughout the school year and no opportunity for students to practice and self-assess in their own time. In addition, it isn't possible for teachers to upload their own questions to the system. The question bank remains static and contains 443 questions. Lastly, the authors of PAT admit that the statistical data available to students and teachers is fairly limited, and that improvements should be pursued in future work.

3.2 SIETTE

SIETTE[15] (System of Intelligent Evaluation Using Tests for Tele-education) is a web based environment for generating and constructing adaptive tests. It has been used with great success in courses from the computer science engineering school, the telecommunication engineering school and the philosophy faculty, all at the University of Malaga, Spain.

SIETTE is a vast system, and at the time of publication (2005) it contained information about 15000 student test sessions, and the knowledge base contained 84 subjects, 1852 topics, 3820 question and 220 tests.

SIETTE is designed to be used by both students and teachers. Teachers use the system to create tests, define the subject topics, the questions and their parameters. Students use the system to take the tests specified by the teachers. The tests can be used for self-assessment, where the correction is shown immediately after the student has answered. Hints and more extensive feedback are also available in this mode. Or, the tests can be used as exams, where the score counts towards the student's final grade. In this mode, hints and feedback aren't provided. It is important to note that tests have a fixed number of questions, thus, new tests must be created every time a student

runs out of practice.

As mentioned earlier, SIETTE constructs adaptive tests, therefore, when a student answers a question, his ability is re-estimated and the next question is selected accordingly. Implementation of this is done by using item response theory in the computerized adaptive testing framework.

SIETTE uses the three-parameter logistic (3PL) model and measures the student's knowledge in terms of a discrete random variable θ, which ranges from 0 to $K-1$, where K is the number of discrete knowledge levels. When it comes to creating items, the guessing parameter c is determined automatically, whereas the other two parameters must be entered by the teacher. The discrimination parameter (a) must be a number between 0.5 and 1.5. The difficulty parameter must be a natural number between 0 and $K-1$. A few years after the SIETTE paper was published, the authors added an item calibration tool because teacher estimates of item parameters are never very accurate.

To estimate students' knowledge level, SIETTE uses the Bayesian estimation method (section 2.3.4) with a knowledge distribution per student, per topic. Also, SIETTE provides teachers with the option of choosing different item selection procedures for tests, such as random selection, difficulty-based selection and bayesian selection.

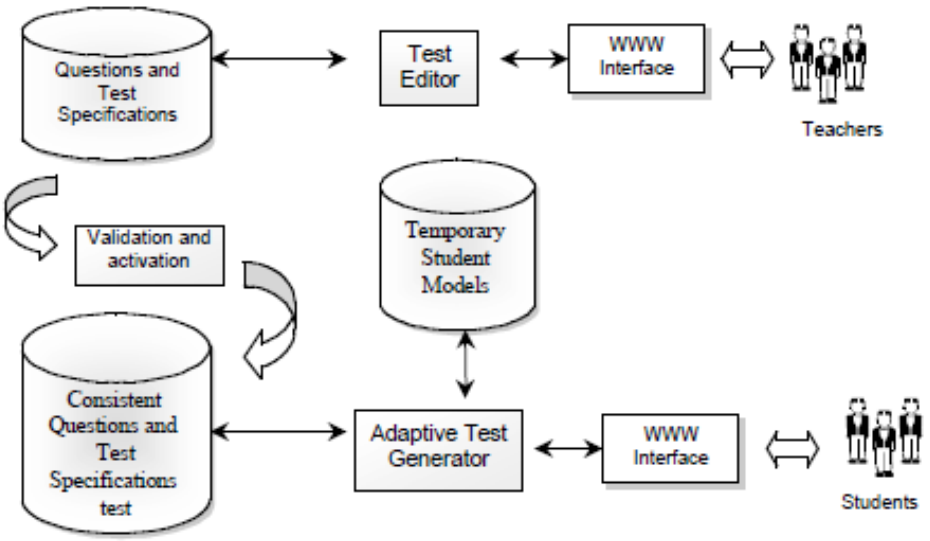

Figure 3.2: SIETTE Architecture. (Source:[4])

Figure 3.2 gives on overview of the SIETTE architecture. The system is comprised of several components[16]:

- The *knowledge base*: where tests, topics and questions are stored. Some supported question types are true/false, multiple choice, multiple response and fill-in-the-gap. More interactive questions also exist, implemented as Java applets, where one has to color parts of a map, or move around objects so that they appear chronologically, for example.
- The *student model repository*: a collection of student models, where information about students' knowledge level estimation, which questions they answered, etc... is stored.
- The *student workspace*: a web interface where students take tests.
- The *test editor*: a tool where teachers can define tests, topics, questions.
- The *result analyzer*: a tool which presents graphical data about students' performance, knowledge estimation levels, etc...
- The *item calibration tool*: a module used to calibrate items by determining the item parameters (difficulty, discrimination and pseudo-chance).

SIETTE is a large and complex system, containing many useful features relevant to the area of web-education, and going through all of these wouldn't be possible. Since SIETTE has been used to such success in university courses we decide to inspire ourselves from this system, especially for the implementation of the adaptive difficulty component. Details about the algorithms used in SIETTE, and hence jSCAPE, can be found in section 5.4, with Java code to illustrate. However, SIETTE does not provide mechanisms to automatically generate exercises, and so we shall define our own algorithms for this.

3.3 Summary

We have looked at some of the relevant work in the field of computer based education and assessment. We saw that PAT provided a simple algorithm for increasing or decreasing the difficulty of exercises, and so an improved version of this algorithm will feature in jSCAPE. We also saw that SIETTE provided many of the features we set out to replicate in jSCAPE, therefore particular parts of our implementation will be inspired by SIETTE.

SIETTE is the most complete system we have come across while doing research for this project. Moreover, examining some existing solutions which haven't been detailed in this section, has given us insight into the most common features available in these types of systems.

In the next chapter we present the system developed as part of this project: jSCAPE.

Chapter 4

The jSCAPE System

The jSCAPE system is designed for two distinct groups of users: students and teachers/lecturers. This separation of roles lead to the development of the main application for students, and an administrator tool for teachers/lecturers.

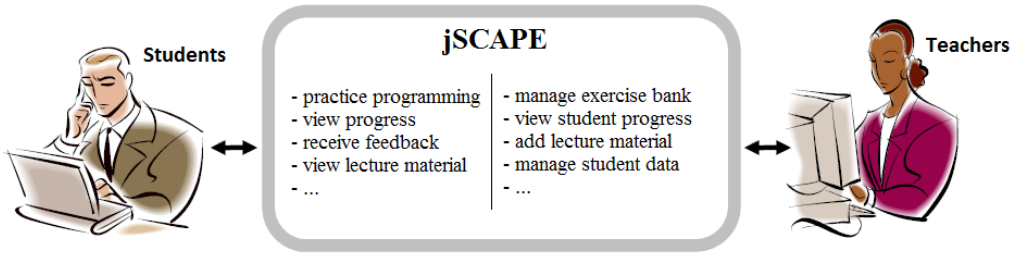

Figure 4.1: Use case diagram of the jSCAPE system.

Figure 4.1 shows some of the main capabilities of the jSCAPE system. Students can practice their understanding of programming concepts by answering exercises provided by the lecturers, and receive feedback while doing so. In addition, students can track their progress by viewing various graphs and charts of their performance on particular exercise categories. Finally, students can access lecture notes and website links provided by the teacher.

Teachers can manage the exercise bank, whether it be adding exercises manually or automatically generating new ones based on templates. They can keep track of their students' progress and thereby identify any difficulties particular students are having. Finally, teachers are responsible for adding lecture material, website links and creating student profiles to store in the database.

In the rest of this chapter we take a closer look at the current available features of jSCAPE.

At the time of writing this report, we would like to note that the screen shots of the application do not represent the final version of jSCAPE, in particular, the graphics and logos haven't been finalized.

4.1 Student view

4.1.1 Login screen

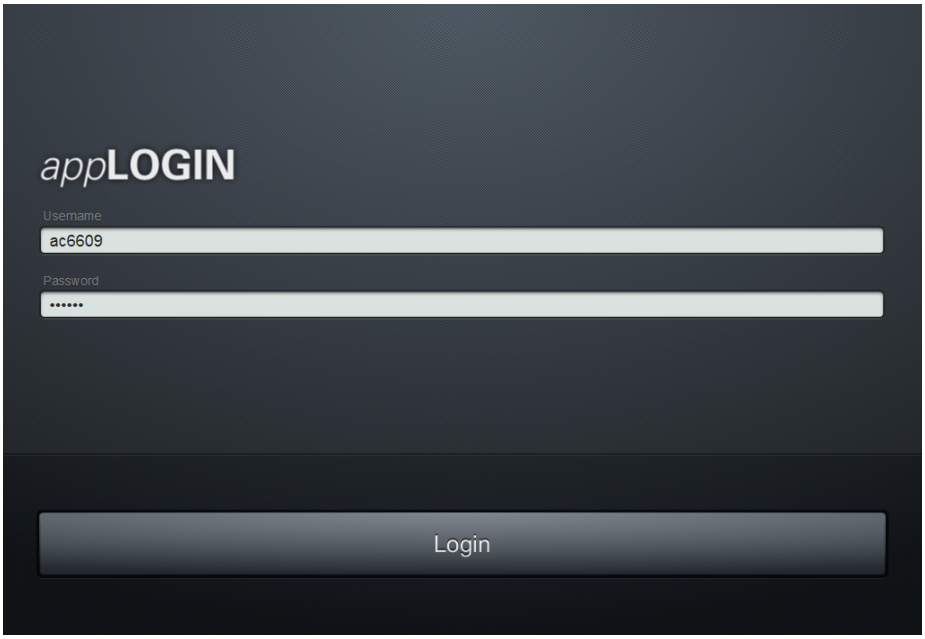

Figure 4.2: The jSCAPE login screen.

For a student to use jSCAPE, they need to be in possession of login credentials, usually acquired by asking the appropriate teacher or lecturer. A connection to the jSCAPE system will be rejected if the entered login details are incorrect. Otherwise, the student can proceed to jSCAPE and access its features.

4.1.2 Practising programming

One of the main features of jSCAPE is the ability to practise programming through answering exercises. Selecting the Practice tab brings the student

to a window where exercise categories, defined by the teacher, are displayed. These categories exist to separate exercises so that students can focus on practising one particular concept at a time.

Figure 4.3 gives an overview of the Practice tab in jSCAPE. In this case, seven exercise categories have been defined by the teacher: Arrays, Loops, Syntax, Conditionals, Binary Trees, Strings and Objects. Order is irrelevant, students simply choose what type of exercise they want to practice.

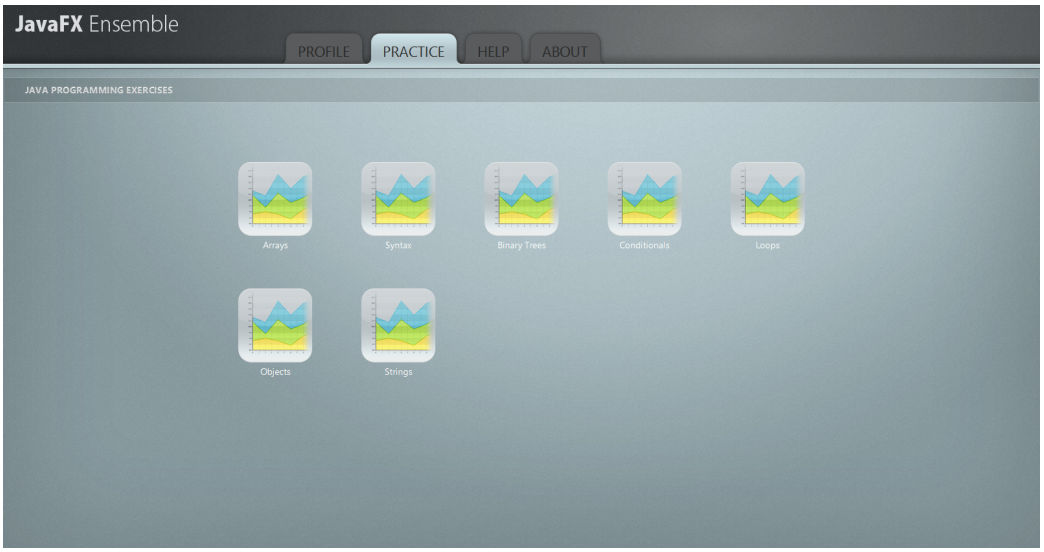

Figure 4.3: An overview of the Practice tab in jSCAPE.

Clicking one of the exercise categories will bring the student to a new window, with an exercise and some helpful information about the chosen exercise category. Figures 4.4 and 4.5 give examples of what this exercise view can look like.

In the case of the binary tree exercise (figure 4.4), exercise data, in the form of a binary tree, is displayed on the left of the window. On the right side of the window is the exercise itself, in this case it asks what should be printed if the binary tree is traversed using the in-order algorithm, and gives four options to choose from. It can be seen that the student has attempted to answer the exercise and has selected the wrong answer. This is indicated to the student in the form of "wrong answer" in red. A correct answer would display "correct answer" in green. Moreover the solution to the exercise is provided immediately after the student has answered it.

Example exercises

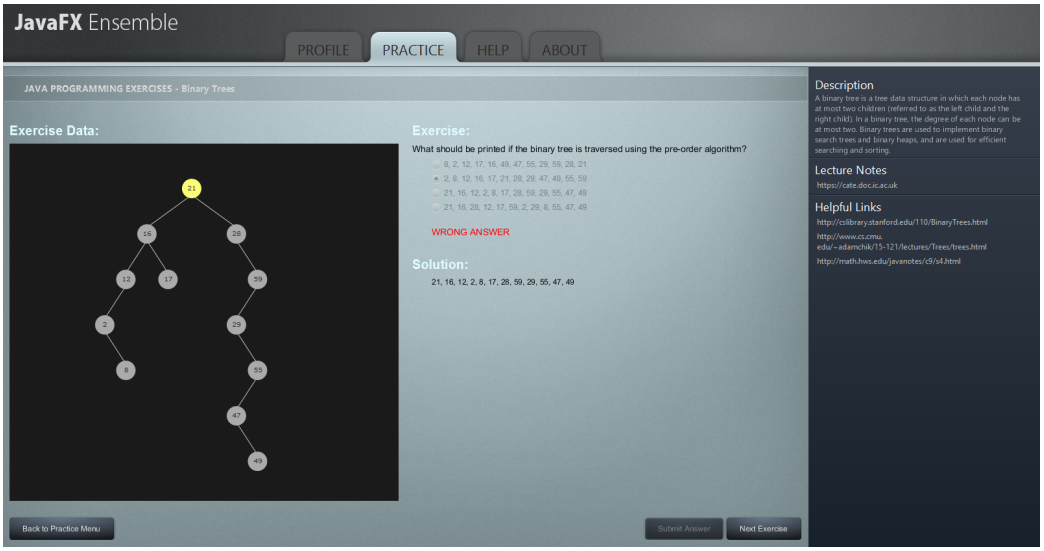

Figure 4.4: The Practice tab view showing an exercise on binary trees.

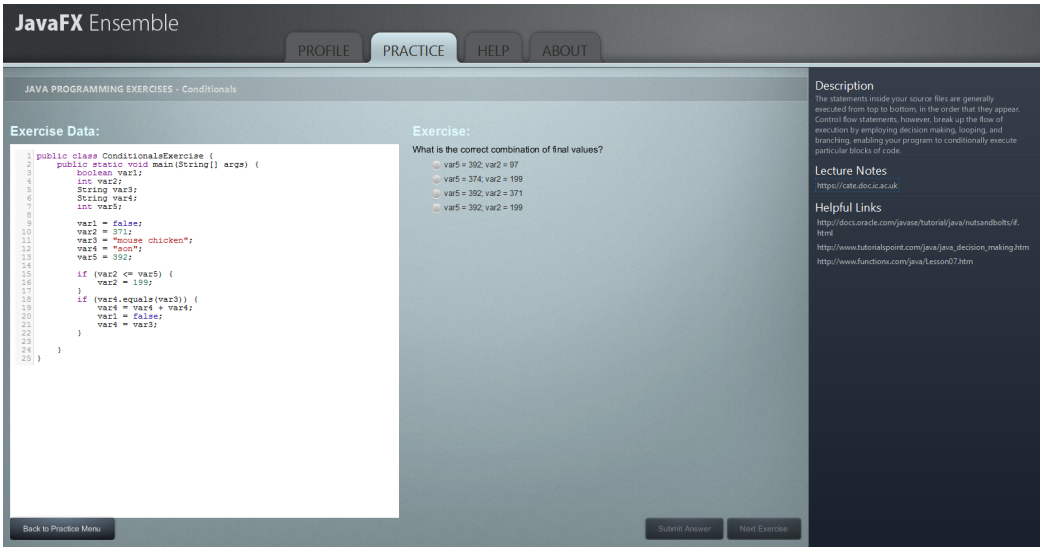

Figure 4.5: The Practice tab view showing an exercise on conditionals.

In the case of the exercise on conditionals (figure 4.5), exercise data, in the form of a code fragment, is displayed on the left of the window. On the right side of the window is the exercise itself, in this case it asks the student to examine the code and to determine the final values of two variables, and provides again four options to choose from.

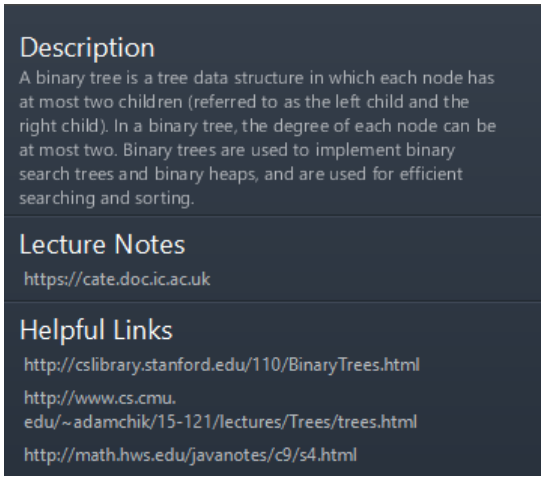

Figure 4.6: An example sidebar in the Practice tab.

On the far right of every exercise, there is a dark blue sidebar which displays a description of the programming concept or construct being practised, links to university or course lecture notes and some other websites on the Internet if further help is needed. An example sidebar is shown in figure 4.6, in this case the sidebar for the "Binary Trees" exercise category.

While answering exercises, students might notice how the difficulty of the exercises adapt to their performance. The system uses simple metrics to either increase or decrease the difficulty of exercises.

For instance, for exercises in the Binary Tree category, the binary trees will increase in the number of nodes, or become very unbalanced. An example of this progression is shown in figure 4.7.

For exercises in the Conditionals category, the number of `if` and `else` statements might increase or the exercise might ask for the final values of more variables.An example of this progression is shown in figure 4.8.

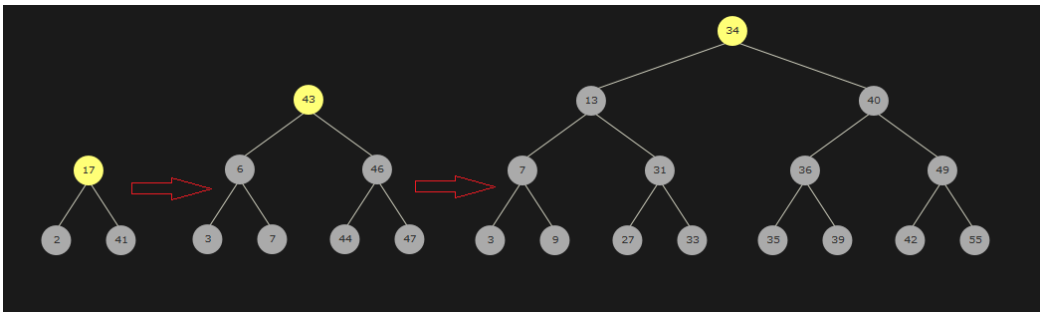

Figure 4.7: Progression of binary tree exercises.

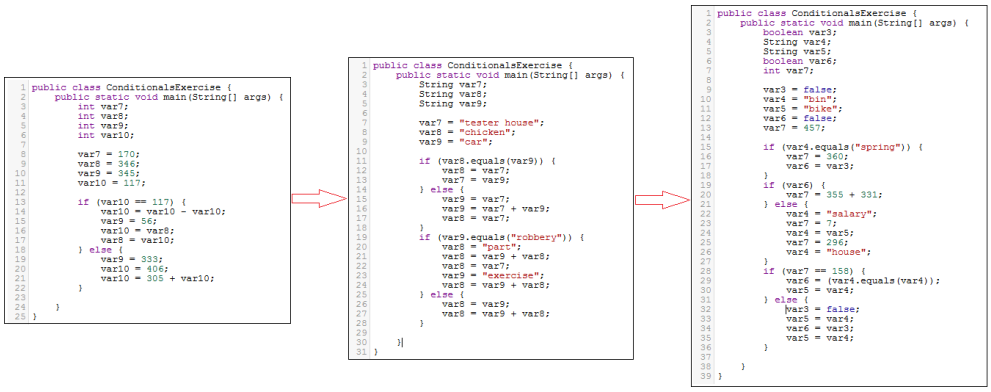

Figure 4.8: Progression of conditionals exercises.

More details about how jSCAPE adapts the difficulty of exercises are given in section 5.4.3 and section 5.5.3.

4.1.3 Tracking progress through statistical data

After a successful login the student lands on the Profile tab which presents information about the student, as well as statistical data on the student's performance and usage of the system.

Figure 4.9 shows the Profile tab after the student has logged in to the system. Profile information for the student is listed on the left hand side, in the light-blue rectangle. This information includes the student's first name, last name, user name, which class the student is in, the last time the student logged in, and the last time the student answered an exercise.

Figure 4.9: An overview of the Profile tab in jSCAPE.

The main part of the Profile tab is split horizontally between statistical data in the form of pie charts and tables, and graphical data in the form of bar charts.

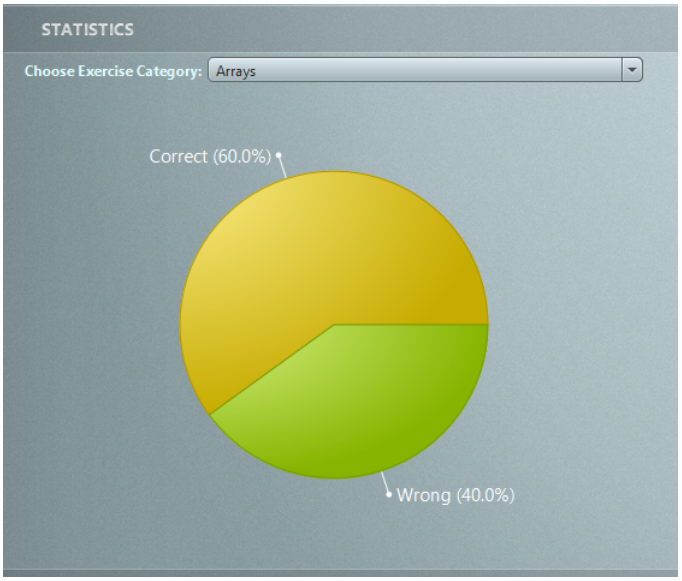

Figure 4.10: Pie chart statistics for exercise category.

Figure 4.10 shows the performance of the student in a particular exercise category, in this case "Arrays". In the example, the student has gotten 60% of array exercises correct and thus 40% of them wrong. The student can

view the performance pie chart for other exercise categories by changing the selected category in the combo box.

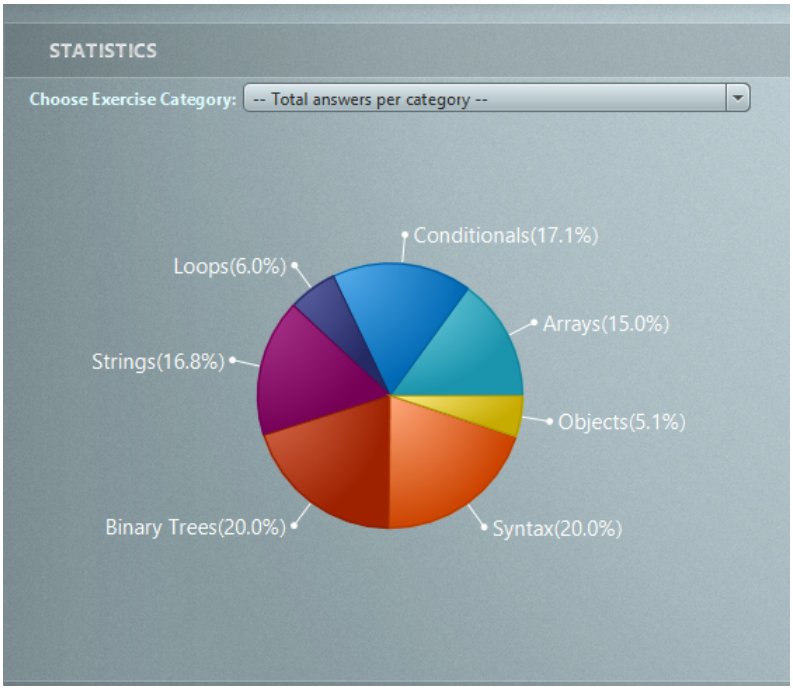

Figure 4.11: Pie chart statistics for distribution of answers.

Another type of pie chart available in jSCAPE can be seen in figure 4.11. This pie chart shows the distribution of answers per exercise category. This is a useful feature when a student is trying to get a balanced amount of practice in all exercise categories.

Next, performance data is presented to the student in the performance summary table, shown in figure 4.12. In this table, there is a row for every exercise category and a row for the total of each column. Each row contains the number of exercises answered, the number of correct answers and the number of wrong answers associated with a particular exercise category.

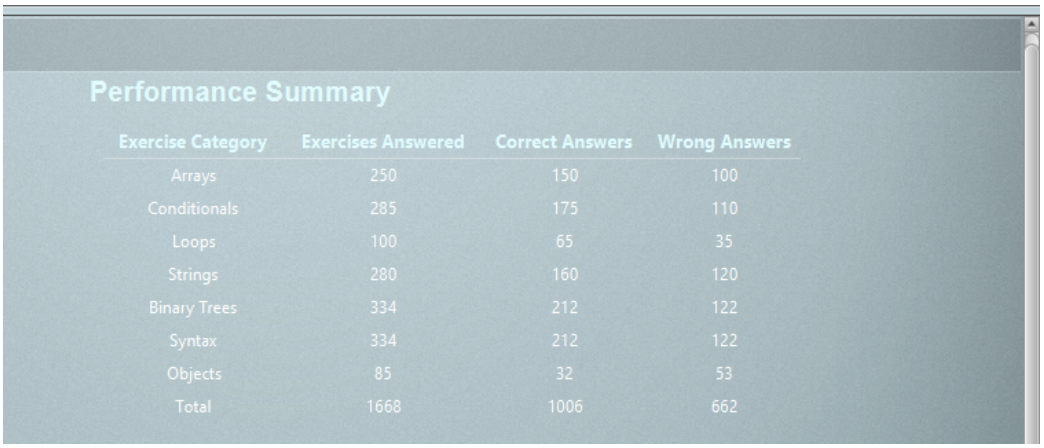

Figure 4.12: Performance summary table.

In the lower half of the Profile tab there is the possibility to view performance data in the form of stacked bar charts.

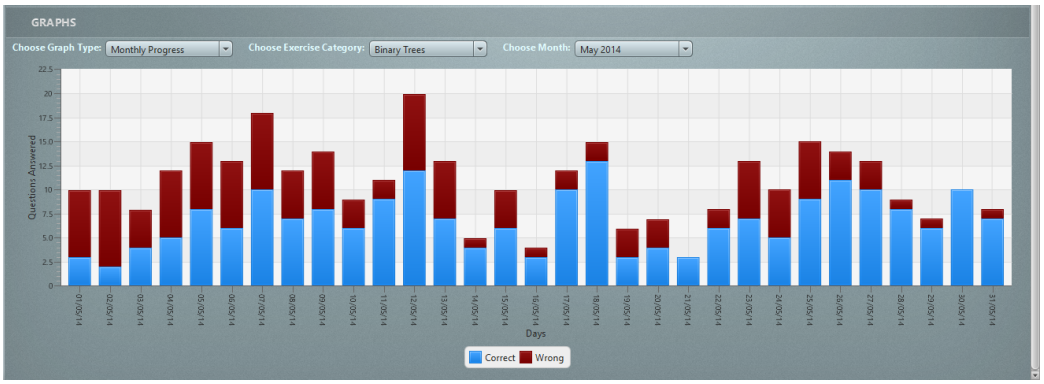

Figure 4.13: Graph data of monthly progress.

Figure 4.13 shows the monthly progress of a student for the month of May 2014 and for the exercise category "Binary Trees". The number of correct answers (in blue) and wrong answers (in red) are graphed for each day where the student answered exercises. The student can view his monthly progress in other exercise categories and other months by manipulating the appropriate combo boxes. This historical data goes back to the first month in which the student answered an exercise.

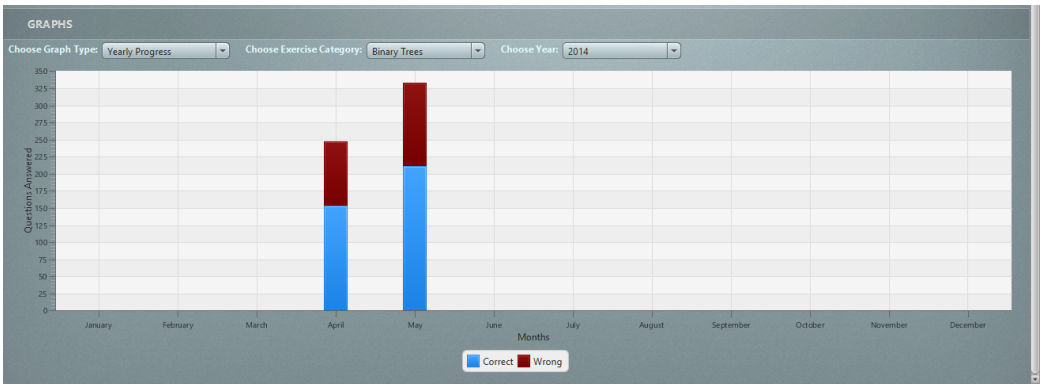

Figure 4.14: Graph data of yearly progress.

Figure 4.14 shows the yearly progress of a student in 2014 for the exercise category "Binary Trees". For each month where the student answered exercises, a stacked bar can be found containing the total number of correct answers (in blue) and the total number of wrong answers (in red) for that particular year and exercise category. The student can view his yearly progress in other exercise categories and other years by manipulating the appropriate combo boxes. This historical data goes back to the year in which the student first started answering exercises.

4.2 Teacher view

The teacher's main access to the system is through the jSCAPE admin tool. Essentially it provides an interface to the database where all information about students, performance, exercise categories and exercises is stored. This tool was developed to allow teachers, not familiar with SQL, to still be able to retrieve useful data about students in a presentable way and to facilitate management of the exercise bank.

4.2.1 Tracking student progress

Figure 4.15: An overview of the Analyze tab in the jSCAPE admin tool.

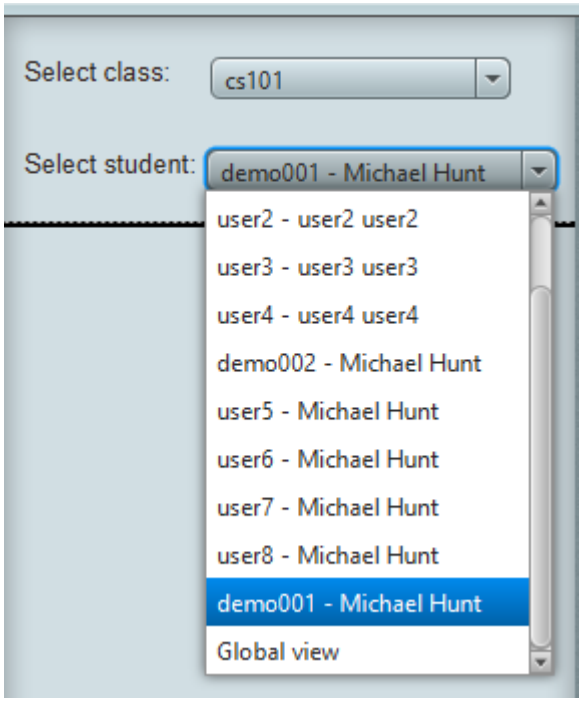

Figure 4.16: Selection possibilities in the Analyze tab.

The jSCAPE admin tool provides teachers with the ability to track student progress and performance over time. Figure 4.15 gives an overview of the Analyze tab, where statistical data about selected students can be displayed. The information displayed is identical to that displayed in the jSCAPE Profile tab (section 4.1.3). On the left hand side of the window, the light blue box contains options to filter which data is displayed in the main window. A close up of the filter options is shown in figure 4.16.

The jSCAPE system includes support for multiple classes to allow for both the separation of students and the separation of exercises available to a class. A teacher can select a class to view statistics about those students taking the class. This will update the list of students in the combo box, allowing the teacher to focus his attention on the performance of one particular student.

Selecting a student will show their profile information in the light blue window, along with the date of their last login, and the date of their last exercise answered. In addition, the pie charts, performance table and progress graphs will be updated to reflect the performance of the selected student. Finally, there is an option to obtain a global view of the class' performance by selecting the "Global view" option.

Student Name	Exercises Answered	Correct Answers	Correct Percentage	Wrong Answers	Wrong Percentage
demo001	334	212	63.47	122	36.53
demo002	334	212	63.47	122	36.53
user5	334	212	63.47	122	36.53
user6	334	212	63.47	122	36.53
user7	334	212	63.47	122	36.53
user8	334	212	63.47	122	36.53

Figure 4.17: Global statistics view of a class.

Figure 4.17 shows the table that is displayed after selecting the global view option. This table shows all the students who have answered exercises in a particular exercise category. In the example above, the data shown is for the "Syntax" exercise category. The student user names are listed along with the number of exercises they have answered, and a detailed breakdown of the number of correct and wrong answers in terms of raw values and percentages.

There is a combo box to select which exercise category to display, but this isn't shown in the picture to minimize the size of it. The global view feature is useful for teachers to identify which students may be facing difficulties.

They can then select the student in the Analyze tab to get more detailed statistics and information about the student's progress.

4.2.2 Managing the exercise bank

A central feature of jSCAPE is providing programming exercises to students. The Exercise tab of the admin tool allows teachers to manage the exercise bank and perform functions such as adding new exercise categories, viewing existing exercises, adding new exercises manually and automatically generating more exercises.

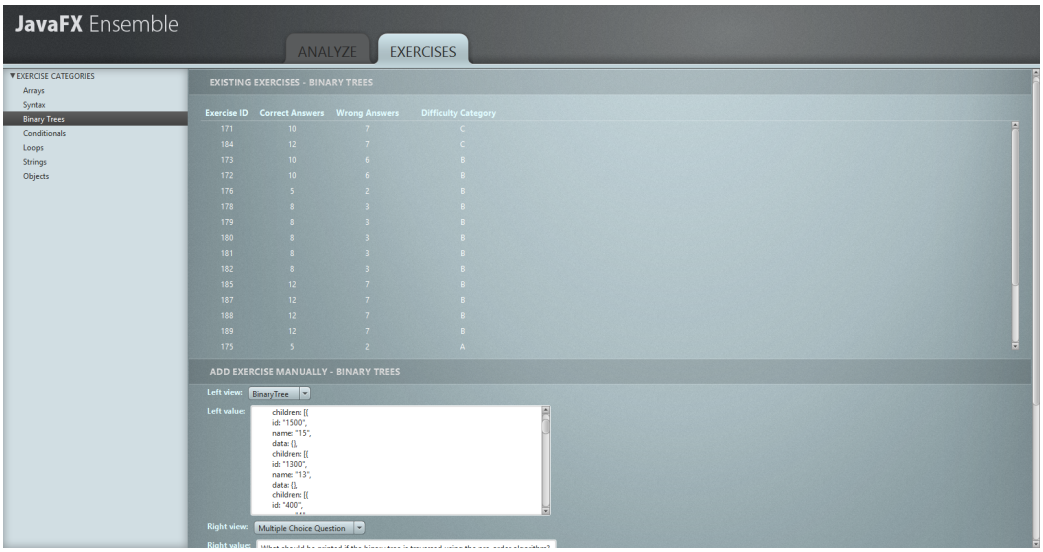

Figure 4.18: An overview of the exercise bank management tab.

Figure 4.18 gives an overview of the Exercise tab. The light blue window on the left contains all existing exercise categories allowing the teacher to select which category they want to manage. In this case, the teacher has selected to view the exercise category "Binary Trees", which displays the appropriate information in the main part of the tab, on the right.

Managing exercise categories

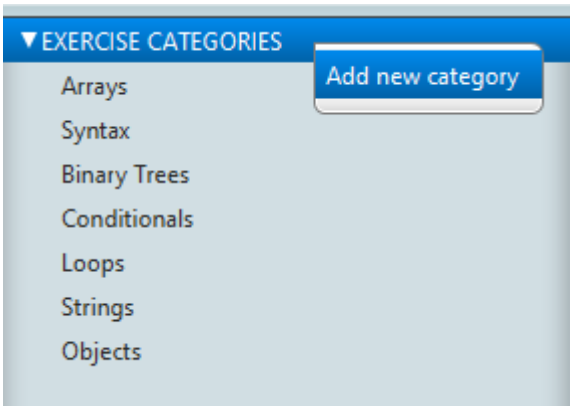

Figure 4.19: Adding a new exercise category.

Figure 4.19 shows a close up of the light blue window. It lists all the existing exercise categories and selecting them will update the information displayed. Right-clicking on the root of the list opens up a context menu showing the possibility of adding a new exercise category. The definition of a new exercise category involves adding a description of the programming construct, some links to lecture notes and other helpful websites. These are of course optional. This information appears as part of the sidebar to an exercise as in figure 4.6.

There is also the possibility of choosing whether to make an exercise category visible or not to students. This is a useful feature if the class isn't ready to answer exercises of a particular category because the relevant material hasn't been taught yet, but the teacher still wants to prepare the exercises in advance to save time.

Viewing existing exercises

Once an exercise category has been selected from the list, several pieces of information are displayed.

EXISTING EXERCISES - BINARY TREES

Exercise ID	Correct Answers	Wrong Answers	Difficulty Category
171	10	7	C
173	10	6	B
175	5	2	A
172	10	6	B
174	5	2	A
176	5	2	B
177	5	2	A
178	8	3	B
179	8	3	B
180	8	3	B
181	8	3	B
182	8	3	B
183	8	3	A
184	12	7	C
185	12	7	B

Figure 4.20: Viewing information about existing exercises.

Figure 4.20 shows one such piece of information. It is a table listing all the existing exercises of the selected exercise category which have been answered at least once by some student. The table lists some useful information about each exercise such as the exercise ID, the number of correct answers, the number of wrong answers and some information about difficulty.

At the time of writing this report, the feature of double clicking a row to show the actual exercise description hasn't been implemented. We think that this would be a useful feature to have, enabling a teacher to view exactly which exercises students are getting right and wrong. However, a teacher competent with SQL and familiar with the jSCAPE system can still use the exercise ID as an index into the database table to retrieve further information about the exercise, including the actual exercise description.

Automatically generating exercises

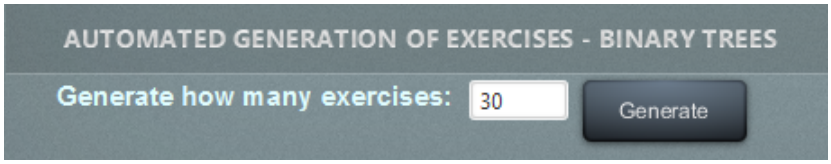

Figure 4.21: Automatically generating a number of new exercises for the Binary Tree exercise category.

In jSCAPE teachers can automatically generate new exercises for a selected exercise category. The teacher inputs the number of exercises he wishes to generate, clicks the Generate button which will call the appropriate exercise generator, create the exercises and add them to the exercise bank. For more details about the implementation of the automated exercise generation process, refer to section 5.5.3.

Adding an exercise manually

Although automatically generating exercises is a useful feature, it lacks the control offered by adding exercises manually. Indeed, the manual addition of exercises allows teachers to define exercises to target any identified student weaknesses. It also allows for more complex exercises which the automatic exercise generator wouldn't be able to produce.

Figure 4.22 shows an example of a teacher adding an exercise on binary trees to the exercise bank. Adding a new exercise manually consists in filling special tags which the jSCAPE system will recognize when trying to display the exercise to the student. These tags are:

- **Left view:** The name of the "handler" that should be used in the left side of the exercise window. At the moment, two handlers exist. One is called BinaryTree for binary tree exercises, and another is called CodeEditor for exercises wishing to display pieces of code in the left view.
- **Left value:** Code or text used by the handler to generate what appears in the left view. In the case of a binary tree exercise, it is the binary tree encoded in a JSON format. For code exercises, it is simply the code snippet to be displayed to the student.

- **Right view:** The format of the exercise. Currently the system only supports multiple choice questions, but it would be quite easy to extend the system to support other types of exercises (section 7.2).
- **Right value:** The exercise description and the four possible choices.
- **Solution:** The solution to the exercise.
- **Difficulty:** Some attributes to indicate difficulty.

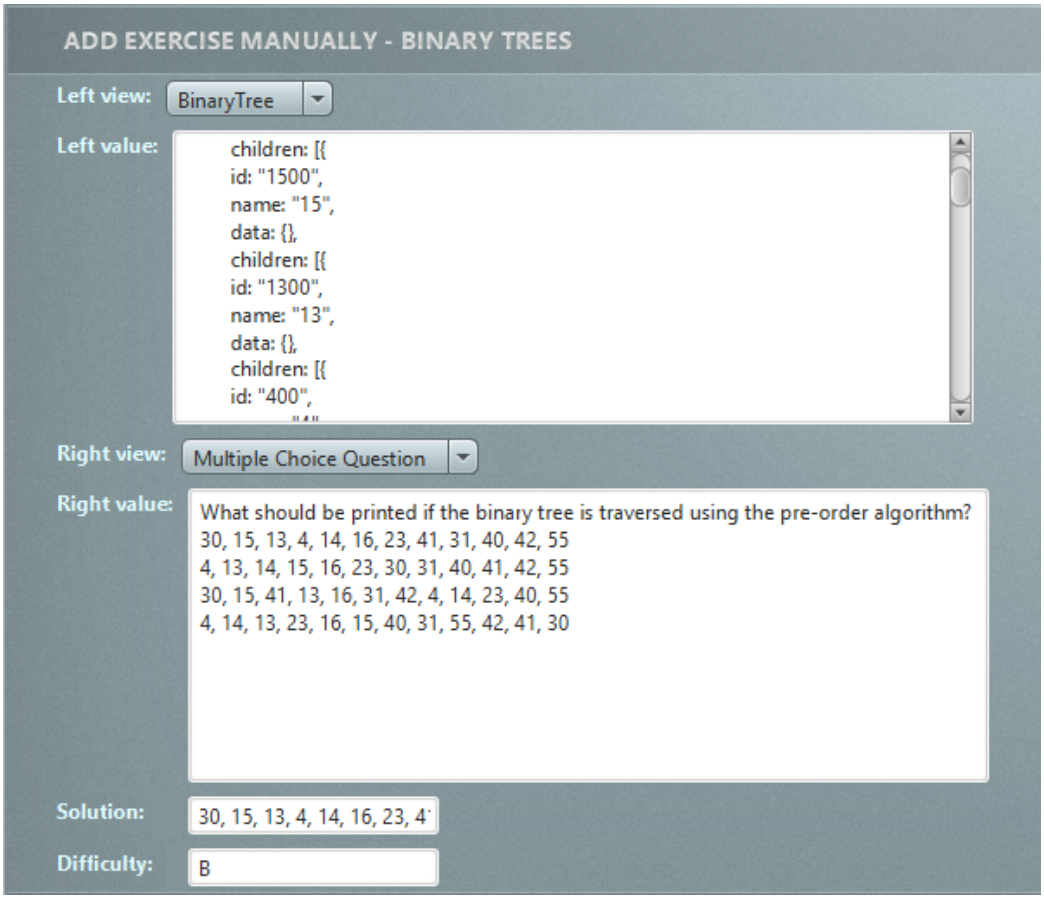

Figure 4.22: Adding an exercise on binary trees manually.

For more details about the implementation of exercises and the various possibilities offered by the jSCAPE exercise format, refer to section 5.5.

Once these fields have all been filled in, the teacher can click the Add button and the exercise will be stored in the exercise bank and made available to students.

4.3 Summary

In this section we gave an overview of the two applications developed as part of this project: jSCAPE and the jSCAPE admin tool.

We showed that jSCAPE provided an infrastructure for students to practice their understanding of programming concepts through exercises and to view their progress thanks to the extensive amount of statistical data collected by the system.

In addition, we presented the jSCAPE admin tool, which allows teachers to define exercise categories, and add or generate exercises to create an environment suitable for student self-assessment. Finally, we showed features that enabled teachers to track student progress through the same statistical data displayed in the main application.

In the following chapter, we talk more about the design of the system and various interesting implementation details and difficulties faced during the development of this project.

Chapter 5

Design and Implementation

Figure 5.1 gives an overview of the three-tier architecture of the jSCAPE system, the relationships between each of the main components, and some of the tasks that they perform. The student view is implemented as a JavaFX applet embedded into the web browser. It communicates with a custom written Java application server using a custom built communication protocol. Finally, the application server also connects to a PostgreSQL database, to perform the standard read and write operations.

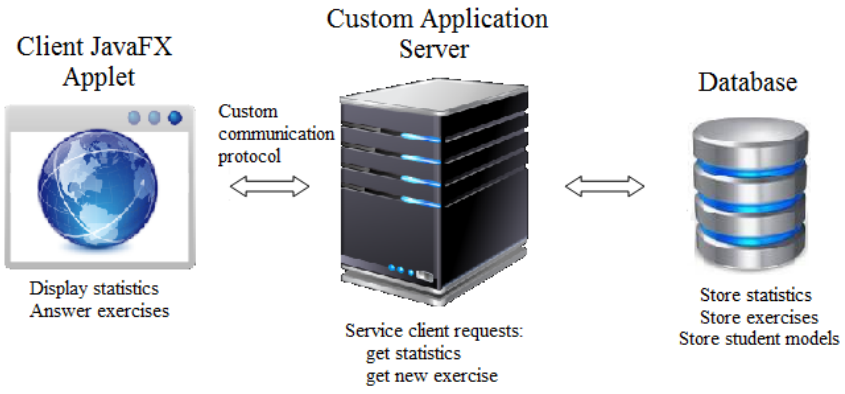

Figure 5.1: Three tier architecture of the jSCAPE system.

The jSCAPE admin tool isn't really part of the core system's three tier architecture. The component was written later on to facilitate certain functions such as analyzing results and exercise bank management. To do so, it connects directly to the database to read or write information. The admin tool is shown in figure 5.2, along with its sub-components: the Results Analyzer,

for displaying statistics in graphical form, and the Exercise Generator, for generating new exercises.

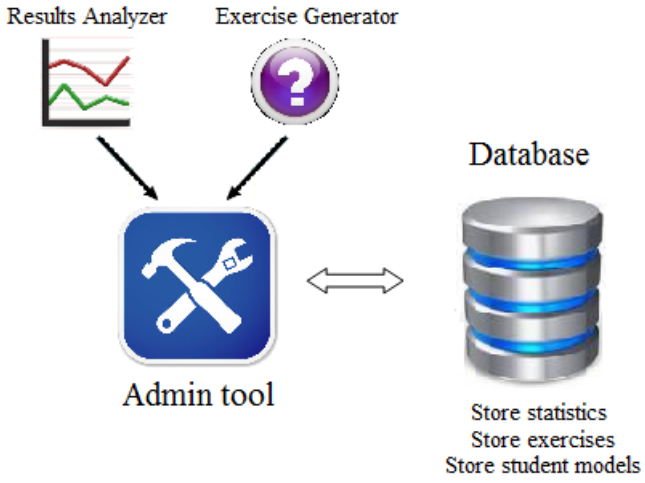

Figure 5.2: Architecture of the jSCAPE admin tool

In the rest of this chapter we justify our design choices and discuss the implementation of the various components and features of the jSCAPE system.

5.1 Technology choices

Most software in the area of computer based education is web-based, as demonstrated by the review of related work in chapter 3. We decided to follow this trend, as it makes deployment easier, and because students are usually quite familiar with web browsers. Therefore, the three-tier architecture of web client, server and database was the natural model of choice for creating such a system. There are multiple technologies which can be used to develop the client side. Some of these are HTML5, CSS, Javascript, Flash, and Java applets.

We decided to take the approach of Java applets because we have had a lot of experience with developing large scale programs in this language. In addition, in our opinion, Java applets are better for creating rich web applications which resemble desktop applications. Websites require the user to continuously click links, and load web pages to access a feature, and we think that this model isn't suitable for developing jSCAPE. Finally, implementing jSCAPE as an applet allows it to run in Java web start mode or as

a stand alone desktop application, thanks to the Java deployment framework.

However, regular Java applets use the Swing library for GUIs, and as a result, the interface doesn't end up being user-friendly or visually appealing. This is certainly a problem, because although the application can present powerful and useful features, students will only use it if the interface is intuitive and aesthetically pleasing[17].

This realisation led us to researching libraries which could improve the interfaces of Java applets, and discovering JavaFX. JavaFX is intended as a replacement for the Swing GUI library, and is designed to provide a lightweight, hardware-accelerated Java UI platform for creating rich internet applications[18]. The JavaFX library includes a powerful and visually appealing statistics package, with support for pie charts, bar charts, line charts, scatter charts, tables, etc... which was more than enough to implement the statistics tracking and displaying component of jSCAPE. In addition, this would allow us to easily add more displayable statistics in the future (section 7.2), if required.

JavaFX also provides the functionality of embedding a WebView, a browser component which supports HTML5, CSS and Javascript, into the applet. We found that this could be an interesting feature to use for developing exercises with variety and interactivity, For instance, this component is used in the Binary Tree exercises to display the binary trees (section 5.??).

Every web application needs some sort of server to handle web client requests. We looked at Java Server Pages (JSP) and Java Servlets, but the setup and organization required to make them work seemed to be too much work for the very little advantages they offered. Instead we opted to write our own Java server because this would offer more control for handling requests and more flexibility for extensions in the future. In addition, the time and amount of code required to write a functional server was very minimal. Indeed, the basic components of our custom built server fit into approximately 300 lines of code.

A database is needed for permanent storage of the system's state, student models, exercises, etc... Thus, to do this, we use a PostgreSQL database and connect to it using the associated Java Database Connectivity (JDBC) driver.

Throughout this chapter, we will give more detail about which tables are maintained by the system, and how they are used. In addition, we will introduce some other technologies and libraries used in this project later on,

when we look at the exercise generation and display processes.

5.2 Design

This section describes the design of the jSCAPE system.

5.2.1 Client

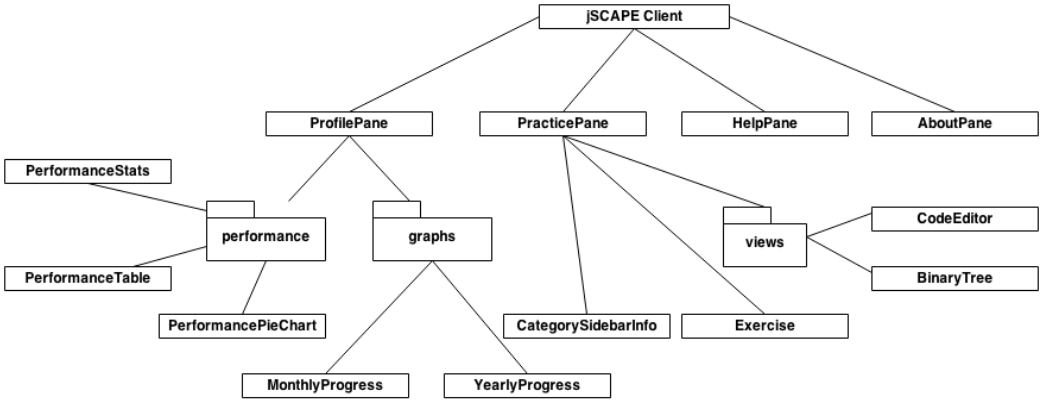

Figure 5.3: Class diagram of the jSCAPE client.

Figure 5.3 shows the class diagram of the jSCAPE client, which is what is embedded into the web browser, and displayed to students. We briefly describe the classes:

- ProfilePane: The Profile tab of the application.
  - performance package:
    - PerformanceStats: Stores performance statistics, i.e. exercise category, correct answers, wrong answers.
    - PerformanceTable: Table wrapper to display PerformanceStats objects.
    - PerformancePieChart: PieChart wrapper to display PerformanceStats objects.
  - graphs package:
    - MonthlyProgress: StackedBarChart wrapper to display monthly progress statistics.
    - YearlyProgress: StackedBarChart wrapper to display yearly progress statistics.

- PracticePane: The Practice tab of the application.
  - Exercise: Stores an exercise and information such as exercise ID, description, choices, solution, display values.
  - CategorySidebarInfo: Stores information to create the sidebar of an exercise window.
  - views package contains the classes to render exercises in the left part of the Practice tab:
    - BinaryTree: Component capable of drawing binary trees.
    - CodeEditor: Component capable of displaying programming code in a code editor.
- HelpPane: The Help tab of the application, displays the manual.
- AboutPane: The About tab of the application, displays information about what the application does.

The jSCAPE client accounts for approximately 2500 lines of code.

5.2.2 Server

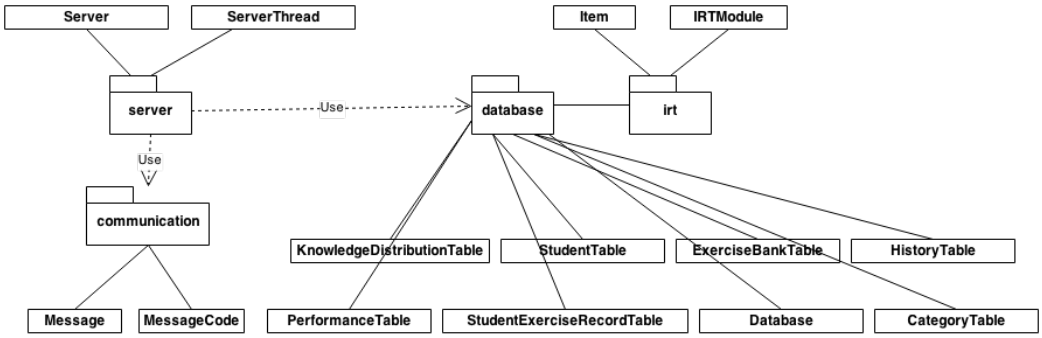

Figure 5.4: Class diagram of the jSCAPE server.

Figure 5.4 shows the class diagram of the jSCAPE server, and the database classes the server uses to satisfy client requests. We briefly describe the classes:

- server package:
  - Server: The server, creates ServerThreads to handle client connections.

  - ServerThread: Receives requests from the client(s), performs work and replies to the client(s).

- communication package:
  - Message: A data structure to hold information. This is the unit which "travels" between the client and server, and vice-versa.
  - MessageCode: Enum to identify structure and format of a Message.

- database package:
  - Database: Manages the connection info and hands out connections to the components that wish to access the physical database.
  - StudentTable: Stores profile information of students, login names, passwords, etc...
  - CategoryTable: Stores the exercise categories, their description and associated lecture notes or helpful website links.
  - PerformanceTable: Stores performance statistics, such as correct answers and wrong answers, of students for the different exercise categories.
  - HistoryTable: Stores performance statistics of students for each day they used the system.
  - ExerciseBankTable: Stores the exercises, some exercise metrics and difficulty parameters.
  - StudentExerciseRecordTable: Stores which exercises students have answered and whether they got it correct or incorrect.
  - KnowledgeDistributionTable: Stores the knowledge distribution of the student per exercise category, in the case that Item Response Theory is used.

- irt package: This package is a Java implementation of Item Response Theory.
  - Item: An IRT item, which stores an exercise ID and item parameters.
  - IRTModule: Implements IRT concepts such as the item response function, the item information function, ability estimation, etc...

5.2.3 Exercises and exercise generators

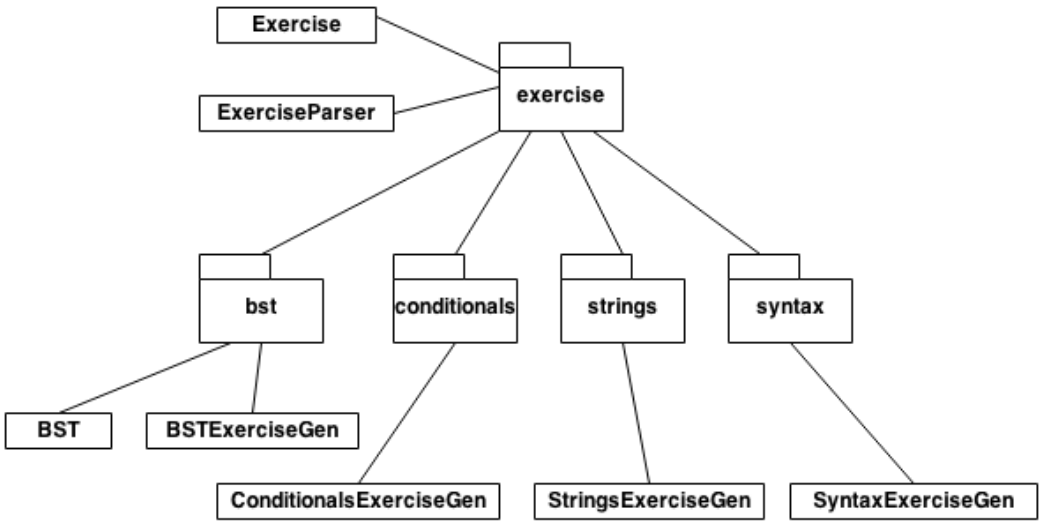

Figure 5.5: Class diagram of the currently implemented exercise generators.

Figure 5.5 shows the class diagram for exercises and the implemented exercise generators, at the time of writing this report. We briefly describe the classes:

- Exercise: Stores an exercise and information such as exercise ID, description, choices, solution, display values.
- ExerciseParser: Special XML parser to decode the exercise format and return an Exercise.
- bst package:
  - BST: Java implementation of binary trees with standard functions such as insert, traversal, height, etc...
  - BSTExerciseGen: Generates an exercise for the Binary Tree exercise category.
- conditionals package:
  - ConditionalsExerciseGen: Generates an exercise for the Conditionals exercise category.
- strings package:
  - StringsExerciseGen: Generates an exercise for the Strings exercise category.

- syntax package:
  - SyntaxExerciseGen: Generates an exercise for the Syntax exercise category.

The exercise and exercise generators account for approximately 2400 lines of code.

5.2.4 Admin tool

We won't go into the details of the admin tool design. Essentially it reuses classes from other components and adds a few modifications.

This component accounts for approximately 2000 lines of code, although most of those lines are copied from other components.

5.3 Server and client-server communication

The server is responsible for servicing all client requests, for instance, requesting performance statistics or requesting a new exercise. It is custom built, multithreaded and written in pure Java, using Sockets and Object input/output streams. The basic server class and the mechanism to communicate with clients accounts for approximately 300 lines of code.

```
1 public class Message implements Serializable {
2     private MessageCode messageCode;
3     private ArrayList<String> payload;
4 }
```

Listing 5.1: Serializable message object used for client-server communication.

The code in 5.1 shows the basic unit that travels between the client and the server, and vice-versa. A message consists of a message code, used to determine its structure, and a payload of request parameters, in the form of an ArrayList<String>.

The existing message request codes are shown in 5.2. The server uses these to determine what the client has requested, how the request message is formatted and which actions to perform to service the request.

```
1  public enum MessageCode {
2      PROFILE_INFO(0),
3      PERFORMANCE_STATS(1),
4      EXERCISE_CATEGORIES(2),
5      GET_EXERCISE(3),
6      ANSWER_EXERCISE(4),
7      LOGIN(5),
8      GET_DATE_LIST(6),
9      GET_MONTHLY_PROGRESS(7),
10     GET_TOTAL_PER_DAY(8),
11     GET_YEARLY_PROGRESS(9),
12     GET_TOTAL_PER_YEAR(10);
13 }
```

Listing 5.2: Message request codes.

The code snippet in 5.3 gives an example of how the client can construct a request. In this particular example the client is requesting the statistical data about the student's performance. On line 1, a Service is created. A Service is a task that can be performed over and over again by calling the restart() method, like on line 12. Lines 2 to 10 determine what the service should do when it is started. Lines 4 and 5 add the student's login name as a request parameter. On lines 6 and 7, the message is constructed with the appropriate message code, and the request parameters. On line 9, the message is sent to the server, and the reply from the server is stored in the Service, for the client to use later on.

```
1  Service<Message> fetchPerformanceStatsService = new Service<Message>() {
2      @Override
3      protected Task<Message> createTask() {
4          ArrayList<String> payload = new ArrayList<String>();
5          payload.add(myLoginName);
6          Message requestMessage
7              = new Message(MessageCode.PERFORMANCE_STATS, payload);
8
9          return new RequestServerTask(HOST, PORT, requestMessage);
10     }
11 };
12 fetchPerformanceStatsService.restart();
```

Listing 5.3: An example client request.

In addition to communicating with the client, the server also communicates with the PostgreSQL database. This is to retrieve the data requested by the client, and to update the state of the system as the student answers exercises, for instance. Therefore, a database module was created with methods

to perform the necessary functions.

```
1  public static ArrayList<String> getPerformanceStats(String loginName) {
2      PreparedStatement ps;
3      Connection connection = Database.getConnection();
4      ResultSet resultSet;
5
6      ArrayList<String> performanceData = new ArrayList<>();
7
8      String query
9          = "SELECT " + EXERCISE_CATEGORY + ","
10                     + EXERCISES_ANSWERED + ","
11                     + CORRECT_ANSWERS + ","
12                     + WRONG_ANSWERS +
13             " FROM "  + TABLE_NAME +
14             " WHERE " + LOGIN_NAME + " = ?";
15
16     ps = connection.prepareStatement(query);
17     ps.setString(1, loginName);
18     resultSet = ps.executeQuery();
19
20     while (resultSet.next()) {
21         performanceData.add(resultSet.getString(EXERCISE_CATEGORY));
22         performanceData.add("" + resultSet.getInt(EXERCISES_ANSWERED));
23         performanceData.add("" + resultSet.getInt(CORRECT_ANSWERS));
24         performanceData.add("" + resultSet.getInt(WRONG_ANSWERS));
25     }
26
27     return performanceData;
28 }
```

Listing 5.4: An example database retrieval method.

In listing 5.4 we give an example of a database retrieval method. Such methods form a large part of the database module. All methods which read from the database return a ArrayList<String>, so that this can be immediately put in the reply message from the server to the client. In this particular example, the method retrieves performance statistics for the student identified by loginName. Line 6 creates the data structure to hold the information which will be read from the database. Lines 8 to 18 create the query, and send it to the database to be executed. In lines 20 to 24, the result of the query is added to the data structure. Finally, on line 27, all this information is returned to the server, which can now encapsulate this in a reply message, and send that to the client.

The database module also includes methods to update the information stored in the database. These methods simply execute an SQL UPDATE query, and return no status value.

This database module is also used by the jSCAPE admin tool to analyze results and manage the exercise bank. The module comprises of 8 classes, one main database class and one for each database table, and accounts for approximately 1500 lines of code.

5.4 Implementing Computerized Adaptive Testing (CAT)

We refer back to the background chapter, section 2.2, where we outlined the components necessary to build a CAT. We also base this section on the work of SIETTE, for the reasons outlined in the related work chapter, section 3.2.

We decide to follow many of the implementation details of SIETTE, that is, to use the Bayesian estimation method, with a knowledge distribution consisting of 11 discrete knowledge levels, ranging from 0 to 10. In addition, the exercise selection algorithms used by jSCAPE will be presented. We now give more details about how the different CAT components are implemented in jSCAPE.

5.4.1 Calibrated item pool

We mentioned in section 2.2, that calibration was a complex process, and that to be done accurately it required a considerable amount of data. However, to obtain this data we would need to have hundreds of students answer jSCAPE exercises and record whether they got the right or wrong answer. Even if this data could be obtained, calibrating the exercises would then require to purchase specialized software.

Therefore, to show what jSCAPE would be capable of, given an accurately calibrated item pool, we require teachers to input item parameters, just as in SIETTE. In the case of a multiple choice question, the pseudo-chance parameter, c, will be equal to 0.25. The difficulty parameter, b, takes discrete values between 0 and 11, and the discrimination parameter, a, takes values between 0.5 and 1.5.

Evidently, this won't end up being as accurate as an item pool where the calibration process was based on thousands of responses to exercises, but it is the best we can do given the circumstances.

5.4.2 Starting point

The starting point refers to the state of the system when no exercises have been administered to a student. In jSCAPE, no prior information about students is known, so the system's initial ability estimate for the student corresponds to the mean on the ability scale, i.e. $\theta = 5$. In addition, the initial knowledge distribution will be a uniform distribution with $P(\theta) = \frac{1}{11}$, where $P(\theta)$ is the probability that the student's knowledge level is equal to θ. These initial assumptions are taken for every exercise category.

5.4.3 Implementing exercise selection algorithms

In this section we discuss the exercise selection algorithms which have been implemented in jSCAPE. We go through these in chronological order of implementation. In the final jSCAPE system, all these selection algorithms are still present, allowing the teacher to choose which procedure to use for administering exercises to students.

Random selection

Random selection was the first "algorithm" implemented. It allowed us to test the system and put everything in place to then develop the more sophisticated algorithms which will follow. This algorithm is quite simple, it picks an exercise at random from the exercises that haven't been answered yet by a student.

Selecting based on the difficulty category

The idea for the next selection algorithm arose after coming across PAT (section 3.1), software used for assessing Greek high school students' programming knowledge.

We decide to define difficulty categories for exercises. Category A is for easy exercises, category B is for intermediate exercises and category C is for difficult exercises. When an exercise is added manually to the exercise bank, the teacher can choose which difficulty category the exercise belongs to. In

section 5.5.3, we show some mechanisms which enable the exercise generators to determine an exercise's difficulty category, based on some simple metrics.

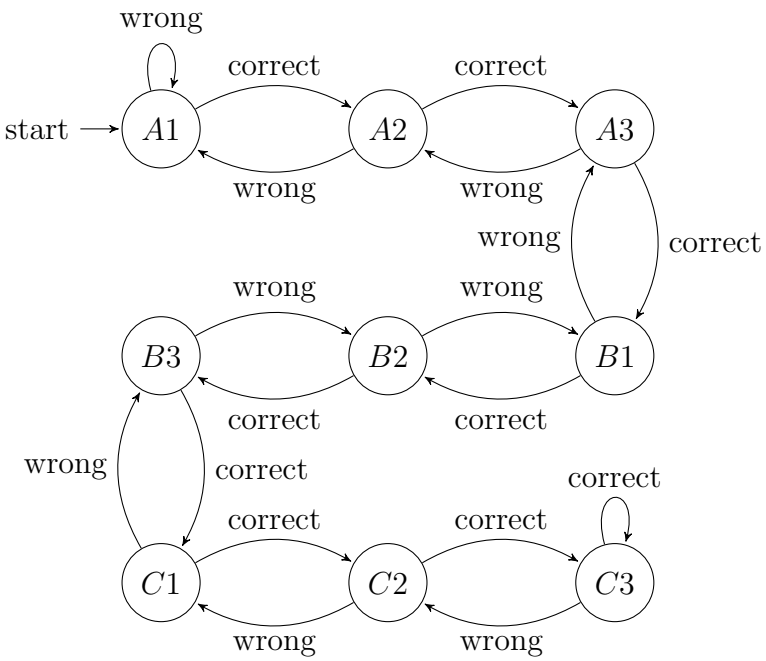

Figure 5.6: State machine of adaptive difficulty categories.

Figure 5.6 shows the state machine implemented by this exercise selection algorithm. For every exercise category, initially students will be presented with exercises from the A difficulty category. A correct answer gets the student closer to getting promoted to a harder difficulty category, for that particular exercise category only. On the other hand, a wrong answer gets the student closer to getting demoted to an easier difficulty category. When a student changes exercise category, or leaves jSCAPE, this difficulty category is saved so that exercises of the appropriate difficulty can be selected during the next session.

The algorithm implemented in jSCAPE waits a while to confirm that the student is in fact ready to move to the next difficulty category. Similarly, the algorithm waits a while before decreasing the difficulty, to account for careless slip ups. For these reasons, we think that the algorithm in jSCAPE is better than the one implemented in PAT.

Selecting using Item Response Theory

This algorithm is the most sophisticated exercise selection algorithm and it is based on the concepts of Item Response Theory.

```
1  public class Item {
2      private int    itemID;
3      private double a;
4      private double b;
5      private double c;
6  }
```

Listing 5.5: An item.

First of all, we define an item as shown in listing 5.5. The itemID is an identifier which points to an exercise in the exercise bank. The variables a, b and c refer to the item parameters a, b and c. These are also columns in the exercise bank database table, null values indicate that the exercise isn't an IRT exercise.

For item selection, we use the maximum information method, which consists in choosing the item with the maximum amount of information for the student's current ability estimate.

```
1  private double itemInformation(int thetaEstimate, Item item) {
2      double information;
3
4      double a = item.getA();
5      double c = item.getC();
6
7      double aSquared = Math.pow(a, 2);
8      double p = probabilityCorrectAnswer(thetaEstimate, item);
9      double q = 1 - p;
10
11     information = aSquared * (q / p);
12
13     double numerator = p - c;
14     double denominator = 1 - c;
15
16     information = information * Math.pow(numerator / denominator, 2);
17
18     return information;
19 }
```

Listing 5.6: Item information algorithm.

In listing 5.6 we show the method which computes item information for a given item.

```
1  private int maxItemInformation(int thetaEstimate, ArrayList<Item> itemList) {
2      double maxInformation = -10;
3      int itemID = 0;
4
5          for (int i = 0; i < itemList.size(); i++) {
6          double information = itemInformation(thetaEstimate, itemList.get(i));
7
8          if (information > maxInformation) {
9              maxInformation = information;
10             itemID = i;
11         }
12     }
13
14     return itemID;
15 }
```

Listing 5.7: Maximum information method.

In listing 5.7 we show the maximum information method, which takes a list of items and the current ability estimate for a student. The algorithm then searches the items for the item with the maximum amount of information, and returns the index of that item in the list, so that it can be presented to the student.

Normally, itemList would be all the exercises that a student hasn't answered yet, for a particular exercise category. However, this can be optimized to include only those exercises whose difficulty parameter is of distance 1 from the current ability estimate for that student. Another possible improvement would be to implement the item information function and maximum information method directly into the database, as user defined functions.

A simulation of jSCAPE in IRT mode is included in appendix B. Amongst other things, it shows how the items which are selected are those which maximize item information.

5.4.4 Scoring algorithm

The scoring algorithm refers to the steps taken to update the student's ability estimate after an exercise has been answered. We decide to use the Bayesian estimation method for this, just as in SIETTE. If an exercise with an assigned difficulty category is answered, then none of the following is done.

```
1  private double probabilityCorrectAnswer(int theta, Item item) {
2      double a = item.getA();
3      double b = item.getB();
4      double c = item.getC();
5
6      double probability = (1 - c) / (1 + Math.exp(-1.7 * a * (theta - b)));
7      probability = c + probability;
8
9      return probability;
10 }
```

Listing 5.8: Item response function (3PL).

First, we list the Java implementation of the 3PL item response function in listing 5.8. This corresponds to the probability of answering item correctly, given thetaEstimate as the current ability estimate for the student.

Recall, that the Bayesian estimation method computes the posterior knowledge distribution from the prior knowledge distribution and observed data in the student's answer to an exercise.

$$P(\theta = k|u_i) = \frac{P(u_i|\theta = k)P(\theta = k)}{\sum_{j=0}^{10} P(u_i|\theta = j)P(\theta = j)}, \qquad \text{for each } k = 0 \text{ to } k = 10 \tag{5.1}$$

Equation (5.1) shows how jSCAPE computes the posterior distribution, after a student has answered exercise i, with $u_i = 1$ if he answered correctly, and $u_i = 0$ otherwise. $P(u_i|\theta = k)$ is the item response function for item i, and $P(\theta = k)$ is the prior knowledge distribution. We apply Bayes' theorem using these two pieces of information to compute the new knowledge distribution.

The implementation of the scoring algorithm, that is, the item response function, the computation of the posterior distribution, etc... forms part of the Item response theory module. It contains quite a few lines of code, thus we include it in appendix C.

A simulation of jSCAPE in IRT mode is included in appendix B. Amongst other things, it shows how the knowledge distribution is updated after a student has answered an exercise.

5.4.5 Termination criterion

When discussing CATs, we mentioned the need for a termination criterion. However, for jSCAPE we wanted no limit to be imposed on the number of

exercises a student could answer. As long as exercises are available in the exercise bank, and as long as a student wants to practice, then he will be able to do so. The only termination criterion per say, is when a student exits an exercise category to practice exercises of another category. This means that when a student goes back to that previous exercise category, his knowledge distribution will continue to evolve from where it left off. A teacher could, for instance, reset the knowledge distribution every week or month to get more accurate and recent estimates of a student's knowledge.

5.5 Exercises

The possibility to practice understanding of programming concepts through exercises can be considered jSCAPE's central feature. In this section we discuss some of the design considerations for exercises, how they are implemented in jSCAPE and how exercise generators can be written to automatically generate these exercises.

5.5.1 Design choices

With jSCAPE's flexible exercise format (section 5.5.2), it is possible to handle multiple types of exercise. However, instead of trying to include a large variety of exercise types, we decide to refine our focus. Therefore, we come up with the following exercise specification:

- To illustrate the capabilities of jSCAPE, and to introduce variety in the exercises, we decide to define exercises which use different views (e.g. BinaryTree, CodeEditor). Research[19] has shown that a code snippet and an exercise asking for the behaviour of the code, provides effective assessment on a student's understanding of code semantics. Therefore, jSCAPE will include this type of exercise, as well as other exercises which don't display a piece of code, for variety purposes, as mentioned before.

- For those exercises which display code snippets, the code will be written in the Java programming language. However, we note that jSCAPE is language independent and that exercises with code snippets from other languages can easily be supported. This is in fact a possible extension to the system (section 7.2).

- Exercises won't ask students to write any code, because students usually have other assignments which ask them to do so. jSCAPE exercises

are intended to practice understanding of fundamentals and to reinforce mental models of what programming language constructs actually do. In addition, this would require jSCAPE to include a component for checking the style and correctness of code written by students. Writing this component would take time off implementing the other components and features of jSCAPE.

- Feedback to an exercise will only be very basic: indicating whether the student got the exercise right or wrong, and displaying the solution. Feedback will be shown immediately after the student has answered the exercise, because evidence[20] suggests that this is the most effective way for providing feedback. More advanced feedback is a possible extension to the system (section 7.2).

- Exercises will be multiple choice only, so that the three-parameter logistic (3PL) model of Item Reponse Theory can be used to implement adaptive difficulty.

5.5.2 jSCAPE exercise format

To represent an exercise entity, jSCAPE uses a simple XML tagged document. This is shown in listing 5.9.

```
1  <?xml version="1.0"?>
2  <exercise>
3      <display>
4          <view>........</view>
5          <value>.......</value>
6      </display>
7      <display>
8          <view>........</view>
9          <value>.......</value>
10         <choice0>.....</choice0>
11         <choice1>.....</choice1>
12         <choice2>.....</choice2>
13         <choice3>.....</choice3>
14         <solution>....</solution>
15     </display>
16     <display>
17         <difficulty>......</difficulty>
18     </display>
19 </exercise>
```

Listing 5.9: Exercise format.

To understand the format, recall how exercises are displayed to students in the Practice tab (section 4.1.2). Exercise data is shown in the left window, and the exercise description and choices are shown in the right window. We now explain the significance of these tags:

- The first <display> tag encloses information about the left window. The enclosed <view> tag corresponds to the component required to render the exercise data in the left window. Currently, two values are possible for this tag, BinaryTree and CodeEditor. The enclosed <value> tag holds the data that is passed to the rendering component to be rendered accordingly.

- The second <display> tag encloses information about the right window. The enclosed <view> tag corresponds to the type of exercise, and currently only one value is possible, Multiple Choice. The enclosed <value> tag holds the exercise description, i.e. the question asked. The <choice> tags correspond to the possible choices given to the student, and finally, the <solution> tag contains the solution to the exercise.

- The last <display> tag encloses a <difficulty> tag which gives the difficulty category of the exercise. Details about the difficulty category parameter will be given in the next section, on exercise generation.

Some examples of jSCAPE exercises are shown in appendix A.

5.5.3 Exercise generation

Although jSCAPE allows for exercises to be added manually, this isn't a feasible way to build a large exercise bank. Therefore, we introduce the process of exercise generation, and give details as to how we implemented exercise generators for some exercises. In simple terms, exercise generation requires filling the tags of the jSCAPE exercise format. In this subsection, we will focus on the exercise generators for two exercise categories: Binary Trees and Conditionals. The other exercise generators are implemented in the same fashion.

Binary Tree exercise generation

Recall the class diagram for exercises and exercise generators (section 5.2.3). The exercise generator for Binary Tree exercises is called BSTExerciseGen, and uses BST, a Java implementation of binary trees, to help in generating exercises. An example Binary Tree exercise, in the jSCAPE exercise format,

is shown in appendix A.

The first step to generating a Binary Tree exercise is selecting the appropriate view for the left window. For this exercise category, we take as an example the creation of exercises which display binary trees as exercise data, and a question that asks what will be printed for various traversal orders. Therefore, we use the BinaryTree view, and this goes into the <view> tag of the first <display> tag, as shown in listing 5.10.

```
1 private String xmlExercise =
2     "<?xml version=\"1.0\"?>\n"
3     + "<exercise>\n"
4     + "    <display>\n"
5     + "        <view>BinaryTree</view>\n"
6     + "        <value>";
```

Listing 5.10: First part of exercise generation.

Next, BST generates a binary tree in Java, from a set of random numbers. This binary tree is either balanced with probability $\frac{2}{3}$ or left unbalanced with probability $\frac{1}{3}$. Our reasoning for this is that students will probably be more familiar with balanced binary trees, therefore unbalanced binary trees can be left for exercises of greater difficulty.

The BinaryTree view is implemented using the JavaScript InfoVis Toolkit (JIT)[21]. This library provides tools for creating interactive visualizations for the web, and requires displayable data to be in the JSON format. Therefore, BSTExerciseGen converts the Java binary tree into the JSON format accepted by the BinaryTree view, and this goes into the <value> tag of the first <display> tag. This process is illustrated below.

```
1 BST<Integer> bst = createBST(randomNumbers);
2 xmlExercise += bst.toJSON();
```

Figure 5.7 shows an example binary tree and listing 5.11 is the JSON representation of that binary tree.

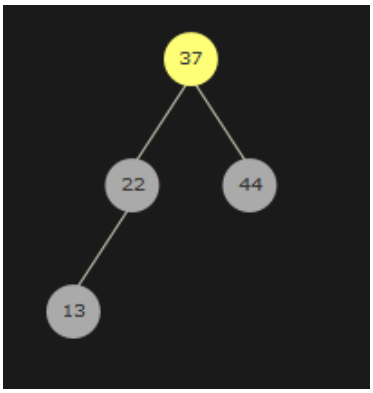

Figure 5.7: A binary tree.

```
{
    id: "3700",
    name: "37",
    data: {},
    children: [{
        id: "2200",
        name: "22",
        data: {},
        children: [{
            id: "1300",
            name: "13",
            data: {},
            children: []},
           {id: "2390",
            name: "null",
            data: {},
            children: []}
        ]},
       {id: "4400",
        name: "44",
        data: {},
        children: []}
    ]
};
```

Listing 5.11: JSON representation of the binary tree in figure 5.7.

Next, we create a data structure with the results of all the traversal orders. This is shuffled and used to complete the <choice> tags. We select a traversal order, at random, from the possible traversal orders, i.e. pre, in, post and level order. This traversal order is inserted into the question, and the result of that traversal is inserted into the <solution> tag. This step is illustrated in listing 5.12.

```
1  ArrayList<String> traversalOrders = new ArrayList<>();
2  traversalOrders.add(bst.inOrderTraversal());
3  traversalOrders.add(bst.preOrderTraversal());
4  traversalOrders.add(bst.postOrderTraversal());
5  traversalOrders.add(bst.levelOrderTraversal());
6
7  int rand = random.nextInt(4);
8  String traversalOrder = "post"/"pre"/"in"/"level" + " order";
9  String solutionTraversalOrder = traversalOrders.get(rand);
10 Collections.shuffle(traversalOrders);
11
12 xmlExercise += "</value>\n"
13     + "     </display>\n"
14     + "     <display>\n"
15     + "         <view>Multiple Choice</view>\n"
16     + "         <value>What should be printed if the binary tree is"
17     + " traversed using the " + traversalOrder + " algorithm?</value>\n"
18     + "         <choice0>" + traversalOrders.get(0) + "</choice0>\n"
19     + "         <choice1>" + traversalOrders.get(1) + "</choice1>\n"
20     + "         <choice2>" + traversalOrders.get(2) + "</choice2>\n"
21     + "         <choice3>" + traversalOrders.get(3) + "</choice3>\n"
22     + "         <solution>" + solutionTraversalOrder + "</solution>\n"
23     + "     </display>\n"
24     + "     <display>\n"
25     + "         <difficulty>" + difficulty + "</difficulty>\n"
26     + "     </display>\n"
27     + "</exercise>";
```

Listing 5.12: Generating question and choices.

For the <difficulty> tag, recall that we have three difficulty categories, A, B and C. We will assign a difficulty category based on some simple metrics. We decide not to use item response theory for automatically generated exercises because any estimation the exercise generator would make for the a and b parameters would be terribly inaccurate.

For this exercise category, we assume that the size, the height and whether the tree is balanced or not, can impact the difficulty of the exercise. For instance, when the number of nodes is less than 7, we assign the exercise to

difficulty category A, between 7 and 12 nodes, to difficulty category B and more than 12 nodes to difficulty category C.

Conditionals exercise generation

Recall the class diagram for exercises and exercise generators (section 5.2.3). The exercise generator for Conditionals exercises is called ConditionalsExerciseGen. An example Conditionals exercise, in the jSCAPE exercise format, is shown in appendix A.

The first step to generating a Conditionals exercise is selecting the appropriate view for the left window. For this exercise category, we take as an example the creation of exercises which display a code snippet, and ask for the final value of either one, two, or three variables. Therefore, we use the CodeEditor view, and this goes into the <view> tag of the first <display> tag, as shown in listing 5.13.

```
1 private String xmlExercise = "<?xml version=\"1.0\"?>\n"
2     + "<exercise>\n"
3     + "     <display>\n"
4     + "         <view>CodeEditor</view>\n"
5     + "         <value>";
```

Listing 5.13: First part of exercise generation.

Next, the code snippet is generated. To generate Java code, a code generator was written specifically for this exercise category. A template file is used to help in the process, it is shown in listing 5.14.

```
1 public class ConditionalsExercise {
2     public static void main(String[] args) {
3         %VARIABLES_DECLARATIONS%
4         %VARIABLES_ASSIGNMENTS%
5         %IF_STATEMENTS%
6     }
7 }
```

Listing 5.14: Code generating template

Since this code generator consists of approximately 400 lines of code, we won't be able to go into extreme detail, so instead we will give the intuition behind the code generating process.

The code generator starts by replacing %VARIABLES_DECLARATIONS% with a random number of variable declarations. Currently, it will declare between

3 and 5 variables, however this can be tweaked very easily. The declared variables are stored in a HashMap with the variable name as the key, and the type (String, int, boolean) of the variable as the value.

Next, %VARIABLES_ASSIGNMENTS% is replaced with random assignments for each declared variable in the previous step. This is accomplished by using the HashMap to figure out the type of each variable.

Finally, the code generator replaces %IF_STATEMENTS% with several if and else branches. It considers each declared variable in turn, and creates an if statement for that variable with probability $\frac{4}{10}$. Then a random number, between 1 and 5, of random assignments are made within that if statement. Also, for each if branch created in this way, the code generator will create an associated else branch with probability $\frac{7}{10}$. Similarly, a random number, between 1 and 5, of random assignments are made within that else statement. The code generation process is over after the last declared variable has been considered for if/else branch creation.

```
1 String randomCode = createRandomCode();
2 xmlExercise += randomCode;
```

The listing above shows how the code snippet is created and then added to the <value> tag of the first <display> tag.

Depending on the type of the exercise, either one, two or three declared variables are chosen, at random, to be included in the question. The code snippet is then interpreted using BeanShell[22] to determine the final value of the variables. During this code interpretation stage, the initial value of a variable and its intermediate values are recorded so that combinations of these can be used as distractors in the multiple choice question. When the exercise asks for the final value of one variable only, this usually isn't enough to create three distractors, so random values for that variable type are chosen instead.

Listing 5.15 shows the final stage of exercise creation for this exercise category. The choices are gathered in an ArrayList and shuffled to be then added to the <choice> tags. The solution is also added to the <solution> tag, and the question is completed depending on the nature of the exercise.

```
1  String randomVar = ...
2  String solutionValue = ...
3  String difficulty = ...
4
5  ArrayList<String> choicesList = ....
6  Collections.shuffle(choicesList);
7
8  xmlExercise += "</value>\n"
9      + "     </display>\n"
10     + "     <display>\n"
11     + "         <view>Multiple Choice</view>\n"
12     + "         <value>What is the final value of variable "
13     + randomVar + "?</value>\n"
14     + "         <choice0>" + choicesList.get(0) + "</choice0>\n"
15     + "         <choice1>" + choicesList.get(1) + "</choice1>\n"
16     + "         <choice2>" + choicesList.get(2) + "</choice2>\n"
17     + "         <choice3>" + choicesList.get(3) + "</choice3>\n"
18     + "         <solution>" + solutionValue + "</solution>\n"
19     + "     </display>\n"
20     + "     <display>\n"
21     + "         <difficulty>" + difficulty + "</difficulty>\n"
22     + "     </display>\n"
23     + "</exercise>";
```

Listing 5.15: Generating question and choices.

For the <difficulty> tag, we will assign a difficulty category based on some simple metrics. There are several factors which we assume affect difficulty, these being the number of if/else branches and the number of final values to determine as part of the exercise.

For instance, in an exercise asking for the final value of two variables, we add the number of if branches and the number of else branches together. If this value is less than 3, then we assign difficulty category A. If it is equal to 3 or 4 then we assign difficulty category B, and finally, if it is greater than 4 then we assign difficulty category C.

The metrics we used for assigning difficulty categories work to some extent, but the randomness of the generated code snippet makes it difficult to get an idea of the difficulty of an exercise. The current method for assigning difficulty categories to the exercises present in jSCAPE, is evaluated in section 6.??.

5.6 Collecting statistical data

One of the main features of jSCAPE is tracking student progress through statistical data. In this section we briefly look at what statistical data is collected and how it is organized in the database.

login_name	exercise_category	exercises_answered	correct_answers	wrong_answers	difficulty_category
demo001	Arrays	250	150	100	A1
demo001	Loops	100	65	35	A1
demo001	Strings	280	160	120	A1
demo001	Syntax	334	212	122	A1
demo001	Objects	85	32	53	A1
demo001	Conditionals	298	183	115	C1
demo001	Binary Trees	357	226	131	C1

Figure 5.8: Performance database table.

Figure 5.8 shows that, for every student, the system records the number of exercises answered, the number of correct answers and the number of wrong answers for each exercise category defined by the teacher. In addition, the system keeps track of what difficulty category the student has reached for every exercise category. This is the information that gets displayed in the pie charts and performance table.

login_name	day	exercise_category	exercises_answered	correct_answers	wrong_answers
demo001	2013-11-01	Binary Trees	10	3	7
demo001	2013-11-02	Binary Trees	10	2	8
demo001	2013-11-03	Binary Trees	8	4	4
demo001	2013-11-06	Binary Trees	7	3	4
demo001	2013-11-08	Binary Trees	12	7	5
demo001	2013-11-11	Binary Trees	15	7	8
demo001	2013-11-17	Binary Trees	13	5	8
demo001	2013-11-19	Binary Trees	10	6	4
demo001	2013-11-25	Binary Trees	8	6	2
demo001	2014-04-01	Binary Trees	10	2	8
demo001	2014-04-02	Binary Trees	14	4	10

Figure 5.9: History database table.

Figure 5.9 shows that the system records, for every student, the days where exercises were answered, the number of exercises answered answered on that day, the number of correct answers and the number of wrong answers, for every exercise category. This data is used to plot the stacked bar charts of monthly and yearly progress.

exercise_id	exercise_category	correct_answers	wrong_answers
6	Binary Trees	10	6
4	Strings	8	1
3	Conditionals	4	8
7	Conditionals	4	8
1	Syntax	5	3
5	Syntax	5	3
9	Syntax	5	3
8	Strings	8	1
12	Strings	8	1
11	Conditionals	4	8
2	Binary Trees	10	6
10	Binary Trees	10	6

Figure 5.10: Exercise bank database table.

Figure 5.10 shows that the system records the number of correct answers and wrong answers for each exercise. This is a useful feature for teachers to identify trends in which type of exercises students are having trouble with and which type of exercises students are having little trouble answering correctly.

5.7 Summary

In this section, we gave details of the design and implementation of jSCAPE.

We started off by justifying the different technologies we used in the project. We then showed the design of the system and its various components through class diagrams and explanations.

Next, we looked at how computerized adaptive testing was implemented in jSCAPE. More specifically, we mentioned that it is possible to choose between random, difficulty category based and IRT based exercise selection algorithms.

We then talked about the types of exercises available in jSCAPE, the exercise format and we showed how exercise generators could be coded to generate exercises automatically.

Finally, we briefly discussed how jSCAPE collected statistical data by maintaining several database tables with various statistics on exercises, student performance and system usage.

Chapter 6

Evaluation

In this chapter we evaluate the strengths and weaknesses of jSCAPE and its various components and features, with respect to the objectives we set at the beginning of the project.

6.1 Evaluating jSCAPE

Evaluating the graphical user interface

When justifying our choices for technologies, we mentioned that one of the reasons for using Java applets and JavaFX was to provide a user-friendly and visually appealing user interface. Evidence[17] suggests that students will prioritise a visually appealing user interface over powerful features. Therefore, we must make sure that our interface is generally well received amongst students, for jSCAPE to be actually useful.

A small amount of data was gathered during an undergraduate fair on 02/06/2014, and during a meeting with student friends on 07/06/2014. In total we asked 10 students what they thought of the jSCAPE interface. The comments were mostly positive, however a few students mentioned that the interface was too large, requiring full screen at times. Two students also mentioned that we should be careful not to overload the view with information, particularly the Profile tab where all the statistics are displayed. We would need more data to identify possible improvements, if any, in the user interface.

We didn't ask for feedback on the user interface of the admin tool as we feel this is less of a factor when it comes to whether teachers actually use the tool. Nonetheless, we think that the GUI of the admin tool is sufficiently user friendly for its purpose.

Evaluating current jSCAPE exercises

At the time of writing this report, we have implemented four exercise generators for the exercise categories of Binary Trees, Conditionals, Syntax and Strings. Some example exercises are shown in appendix A.

For Binary Tree exercises, we supply a binary tree as exercise data, and ask the student to select what is printed after a specific traversal order. This type of exercise was intended to show that it was possible to bring variety to jSCAPE exercises. We think that this exercise is useful for learning the different traversal orders, however the exercise loses its usefulness very quickly once a student has mastered traversal orders. Perhaps this exercise would be more effective if we asked the student to type in the whole traversal instead of selecting the correct answer among the four different traversals.

Next, for Conditionals and Strings exercises, we display a code snippet as exercise data, and ask the student to determine the final value of one or more variables. The usefulness of this type of exercise has been demonstrated in research. Indeed, [19] has shown that this type of exercise provides effective assessment on a student's understanding of code semantics.

Thanks to our random code generators, jSCAPE is able to provide many of such exercises, with very different code snippets. This means that a teacher could automatically generate 1000 Conditionals exercises and all of them would include very different code snippets, thus providing students with a large supply of exercises to self-assess their understanding of control flow in Java. However, sometimes the generated code can end up being too random, and the control flow doesn't resemble anything that would appear in a "real" program. As such, more work would be needed in the code generating component, to make the generated code more useful in the learning process.

Finally, Syntax exercises also present a code snippet, but this time containing syntax errors. The exercise asks the student to spot the syntax error, or to identify how many syntax errors the piece of code contains. We believe that this is a useful exercise for students to familiarise themselves with the Java language, especially in the early stages, when students are discovering programming languages for the first time.

We would like to point out that teachers and educators have more knowledge of what constitutes a useful exercise in assessing students knowledge of a particular concept. These exercises were only intended to show what can be

achieved in the jSCAPE system.

Evaluating the jSCAPE exercise format

The jSCAPE exercise format was sufficient for us to implement four exercise generators for multiple choice type questions. The format is quite flexible to allow for different views to be used in providing exercise data to accompany the question. The format was also designed with other exercise types in mind. For instance, if the exercise displayer were to read FillInBlank in the <view> tag of the second <display> tag, then it would ignore all the <choice> tags and simply create a text field for the student to type in his answer.

However, in general we are not that satisfied with the exercise format. It isn't very elegant and we could see it posing some restrictions in the future.

```
1  <?xml version="1.0"?>
2  <exercise>
3          <display>
4                  <view>........</view>
5                  <value>.......</value>
6          </display>
7          <display>
8                  <view>........</view>
9                  <value>.......</value>
10         </display>
11         <display>
12                 <view>........</view>
13                 <value>.......</value>
14         </display>
15 </exercise>
```

Listing 6.1: A modified exercise format.

We believe that it should be modified to allow for more exercise types to be added to jSCAPE. The listing in 6.1 shows a possible modification, which makes the exercise format more straightforward. Each possible <view> could have a component implemented in Java which takes as parameter the value of the <value> tag. This is how the BinaryTree and CodeEditor were implemented, but for some unknown reason, we didn't think about doing the same with other views such as MultipleChoice. This would offer a more general purpose exercise format than jSCAPE currently has at the moment.

In addition, we designed exercises as consisting of exercise data in the left

window, and the question and choices in the right window. This format is also a bit restrictive as it doesn't allow more interactive types of exercises which would require drag and drop functionality, for instance.

Evaluating automated exercise generation

Exercises of the Binary Trees category were very suitable for automated generation. Indeed, generating random binary trees with a random number of nodes is something relatively easy to do, and the randomness of the binary tree doesn't impact the quality of the exercise.

For exercises which display code snippets, we mentioned before that this generated code can sometimes be too random, and that "strange" code can sometimes be produced. This can definitely impact the quality and effectiveness of the exercise. For instance, sometimes an `if` branch would contain the statement `var6=true` followed by the exact same statement on the next line. This kind of statement would never appear in a program written by a competent programmer, thus presenting this code snippet to a student could very well be counter productive.

We believe that we have shown how exercise generators can be written to provide, theoretically, endless amounts of exercises. This was one of the objectives we set for the project at the beginning of development. However, it still remains that an exercise generator must be written manually for each new type of exercise. This isn't a very scalable solution, and thus we discuss some possible improvements for automated exercise generation, in the next chapter.

Evaluating computerized adaptive testing in jSCAPE

As part of adapting the difficulty of exercises to the student's ability, we implemented two exercise selection algorithms, one based on difficulty categories and another one based on Item response theory.

The difficulty category idea was inspired by PAT[14] and we improved the algorithm to include it in jSCAPE. The improvement was based on evidence[23] which suggests that at most three exercises of a given difficulty are needed to remove uncertainty from assessing a student's knowledge.

The algorithm performed well, thanks to the state machine implementation it is correct in that it will always choose an exercise of the appropriate diffi-

culty category. It is less sophisticated than the Item response theory based algorithm, however it is much easier for teachers, because all they have to do is assign the exercise to one of three categories. But the fact that there are only three exercise categories means that there is less distinction between the difficulty of exercises, as opposed to IRT where we had 11 difficulty levels and the discrimination parameter which also affected perceived difficulty.

Difficulty categories could also be assigned through the automated exercise generation process. To do this we used simple metrics such as binary tree size, code length or number of variables. This worked well under the assumption that Binary Tree exercises become more difficult as they grow in size or as they become unbalanced, however we aren't sure this assumption is valid. It may be valid early on when students are learning about binary trees for the first time.

Exercises with code snippets were much more suitable for automatically assigning to a difficulty category because more metrics were available. For Conditionals, we used the number of `if/else` branches and the number of variables asked for. This worked quite well, however the randomness of the code would still interfere sometimes. A more sophisticated code analysis would be needed to determine the difficulty category and we discuss some improvements in the next chapter.

The more sophisticated exercise selection algorithm we implemented was based on Item response theory. We managed to test the correctness of the implemented algorithm by running a simulation, whose results are included in appendix B. It shows that the algorithm does indeed choose the most appropriate exercise according to the student's estimated knowledge level. The knowledge distribution is also updated correctly.

However, this solution did present some limitations, most notably, it requires teachers to input the item parameters, that is the difficulty and discrimination parameters. Although guidelines were given for this step, the results will be a lot less accurate than what could be achieved through using calibration software and data. Still, it provides a bit more options than the difficulty category based method.

We also evaluate our system in terms of the advantages and disadvantages mentioned in CAT (section 2.2). Most of the complaints for CATs are for when they are used as means to assess students for a grade which counts towards their end of year total. For instance, not being able to go back

and change one's answer. In a self-assessment setting, where the results don't really count towards anything, this isn't much of a problem. However, jSCAPE does present the most common weakness of CATs, in that a large bank of calibrated exercises is needed to be effective. Finally, we mentioned that exercise exposure could be an issue in CATs, but in jSCAPE we aren't concerned about this, because the system is only intended for self-assessment.

Evaluating collection of statistical data

To determine which statistics should be collected by jSCAPE, we inspired ourselves from other software, in particular video games, with statistics such as win/losses, history over time, etc...In addition, we regularly received feedback from our supervisor, Dr. Tristan Allwood, in terms of which statistics would be useful for a teacher or lecturer in monitoring students' progress. Asking more lecturers or teachers could help in determining more useful statistics that jSCAPE could record and display to students or teachers. It would be quite easy to add more statistics, however we believe that the core useful statistics have been implemented.

6.2 Summary

In this chapter we evaluated the different components that form the jSCAPE system. We showed that collection and display of statistical data was implemented quite solidly, and that the system was able to support provision of exercises very well.

However, jSCAPE did present certain limitations in its exercise format, in its exercise generators and in the adaptive difficulty component.

In the next chapter we conclude the project and give some possibilities of future work which could address the weaknesses of the system discussed in this section.

Chapter 7

Conclusion

In this chapter, we summarise the achievements of the project and discuss possible extensions that can be made to the system in the future.

7.1 Summary

The main objective of the project was to produce a web-based application to allow students to practise their programming skills and to provide teachers with the possibility of monitoring student performance. We delivered a product, in the form of jSCAPE, which we feel strongly meets the main objective of the project.

To differentiate the system from other applications, where exercises are presented randomly to students, we introduced the concept of adaptive testing. We implemented two different algorithms to vary the difficulty of exercises according to the student's performance. The algorithms were successful in doing so, however they did have shortcomings. For instance, we learnt that adaptive testing using Item Response Theory is quite complex to implement in practice, and that large amounts of data are needed for the system to be effective.

We realised that developing an exercise bank by only adding exercises manually was a cumbersome and time consuming process. Therefore, we introduced automated exercise generation as a feature. We showed how the jSCAPE exercise format was defined and how exercise generators can be written to automate this process. We learnt that automated generation is quite a complex task, and as such, much improvement is still possible in this area of jSCAPE.

In conclusion, we believe that jSCAPE is a complete system in the sense that it implements the core features which can be useful in the setting of computer based self-assessment of programming knowledge. However, the adaptive difficulty and automated exercise generation components only give a small flavour of what is possible, and as a result, they require more work to make jSCAPE a solid implementation of these ideas.

7.2 Future Work

First of all, before anything else is done, security in jSCAPE needs to be improved. Since the purpose of this project was to provide an application with adaptive and automatically generated exercises, we didn't take normal security measures during the development of the system. For instance, student passwords aren't stored securely in the database, and all communication between the client, server and the database is done in plain text. Therefore, students could intercept exercises as they are sent from server to client, which would reveal the solution of the exercise. To fix this, we would use hashing and salt for password storage, and SSL for communication. These measures should allow jSCAPE to be used in universities or schools without any cheating or hacking passwords.

In addition, to help with implementing extensions a code refactoring of jSCAPE could be considered. Indeed, JavaFX applications are based on the model-view-controller pattern, so a nice separation of these components can be done in the code. However we only learnt about this possibility three weeks into the project, so all the of the GUI components are created in the Java code as opposed to in a separate FXML file, a JavaFX format created specifically for this purpose.

The main features of jSCAPE are automated exercise generation, adaptive difficulty, exercise provision and statistical tracking. Any of these areas could be chosen to be explored in more depth. Given more time we list some of the things which could be accomplished in the future:

- The system we developed is language independent, thus we could add support for other programming languages. This could be done in jSCAPE itself, or by cloning it into a separate application, e.g. cSCAPE, for C and hSCAPE for Haskell.

- We could add support for more exercise types, such as multiple-response questions, fill-in-the-blank, or even more interactive types of exercises thanks to the possibilities offered by JavaFX.

- Currently, feedback after a student has answered an exercise is very basic. The system displays whether the student got the exercise right or wrong, and it displays the solution. We could add more useful feedback to exercises. Examples of more advanced feedback on current exercises could include execution traces or animations of traversal algorithms on the binary tree, etc...

- We could work on a more general-purpose code generator for exercises which use code snippets as exercise data. Currently code generators are written specifically for exercise categories, but it would be more useful and less time consuming to have a code generator module, where certain attributes could be specified such as number of variables, number of loops, if any, etc... This seems to be quite a complex task, and a solid implementation of this could probably be a project on its own.

- We could approach automated generation differently, and instead develop a feature where teachers create exercise templates, perhaps by defining a template grammar and then exercises are automatically generated from these templates. This would remove the need for writing an exercise specific generator each time.

- We could work more extensively on the adaptive component of the system, i.e. improving the algorithm which selects exercises for students based on their estimated ability. We mentioned that obtaining a large calibrated item pool wasn't feasible in the scope of this project, so this would have to be a priority if we had more time. In addition, we could use more complex IRT models which take into account the time spent on each exercise, for instance.

- We could do some more research on what statistical data would be useful for students and teachers and implement components to display this data in jSCAPE, e.g. longest streak of correct/incorrect answers, etc...

Bibliography

[1] Codecademy - Learn to code interactively for free. `http://www.codecademy.com/`.

[2] Coursera - Take the world's best courses online, for free. `https://www.coursera.org/`.

[3] Udacity. `https://www.udacity.com/`.

[4] Conejo, R., Guzmán, E., Millán, E., Trella, M., Pérez-de-la-Cruz, J. L., & Rios, A. (2004). SIETTE: A Web-Based Tool for Adaptive Testing. *International Journal of Artificial Intelligence in Education, 14*, 29-61.

[5] Computerized adaptive testing. `http://en.wikipedia.org/wiki/Computerized_adaptive_testing`. Accessed: June 17, 2014.

[6] Thompson, Nathan A., & Weiss, David A. (2011). A Framework for the Development of Computerized Adaptive Tests. *Practical Assessment, Research & Evaluation*, 16(1).

[7] IRT-Based CAT. `http://www.iacat.org/content/irt-based-cat`. Accessed: June 17, 2014.

[8] Weiss, D. J., & Kingsbury, G. G. (1984). Application of computerized adaptive testing to educational problems. *Journal of Educational Measurement*, 21, 361-375

[9] Weiss, D.J. (1985). Adaptive Testing by Computer, *Journal of Consulting and Clinical Psychology*. 1985, 53, 6, pp. 774-789.

[10] Wainer, H., & Mislevy, R.J. (2000). Item response theory, calibration, and estimation. In Wainer, H. (Ed.) Computerized Adaptive Testing: A Primer. Mahwah, NJ: Lawrence Erlbaum Associates.

[11] Item Response Theory. `http://en.wikipedia.org/wiki/Item_response_theory`. Accessed: June 17, 2014.

[12] Baker, Frank (2001). The Basics of Item Response Theory. (2nd Edition). Available at http://info.worldbank.org/etools/docs/library/117765/Item%20Response%20Theory%20-%20F%20Baker.pdf. Accessed: June 17, 2014.

[13] Partchev, Ivailo (2004). A visual guide to item response theory. Available at www.metheval.uni-jena.de/irt/VisualIRT.pdf. Accessed: June 17, 2014.

[14] Chatzopoulou, D. I. & Economides, A. A. (2010). Adaptive assessment of student's knowledge in programming courses. *Journal of Computer Assisted Learning*, Vol. 26, No. 4.

[15] Guzman, E.; Conejo, R., Self-assessment in a feasible, adaptive web-based testing system, Education, *IEEE Transactions*, vol.48, no.4, pp.688,695, Nov. 2005.

[16] E. Guzmán and R. Conejo, A brief introduction to the new architecture of SIETTE, in *Lecture Notes in Computer Science. Proc. 3rd Adaptive Hypermedia Adaptive Web-based Systems (AH 2004)*, Berlin, Germany, 2004.

[17] Christine Phillips & Barbara S. Chaparro. Visual Appeal vs. Usability: Which One Influences User Perceptions of a Website More? http://psychology.wichita.edu/surl/usabilitynews/112/aesthetic.asp. Accessed: June 17, 2014.

[18] JavaFX - The Rich Client Platform. http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html. Accessed: June 17, 2014.

[19] Lister, R. (2001). Objectives and objective assessment in CS1. *ACM SIGCSE Bulletin*, Vol. 33, No. 1, pp. 292-296.

[20] A. Mitrovic, "SINT—a symbolic integration tutor," in *Proc. 6th Int. Conf. Intelligent Tutoring Systems (ITS 2002), Lecture Notes in Computer Science*, S. Cerri, G. Gouardères, and F. Paraguacu, Eds., New York, 2002, LNCS no. 2363, pp. 587–595.

[21] JavaScript InfoVis Toolkit. http://philogb.github.io/jit/. Accessed: June 17, 2014.

[22] BeanShell - Lightweight Scripting for Java. http://www.beanshell.org. Accessed: June 17, 2014.

[23] Abdullah, S C, and Cooley, R E, 2000, Using Constraints to Develop and Deliver Adaptive Tests, *4th Computer Assisted Assessment Conference*, 21 – 22 June, Loughborough.

Appendix A

Example jSCAPE exercises

Binary tree exercise

```
<?xml version="1.0"?>
<exercise>
    <display>
        <view>BinaryTree</view>
        <value>{
        id: "3300",
        name: "33",
        data: {},
        children: [{
        id: "800",
        name: "8",
        data: {},
        children: [{
        id: "300",
        name: "3",
        data: {},
        children: [{
        id: "200",
        name: "2",
        data: {},
        children: []
    }, {
        id: "400",
        name: "4",
        data: {},
        children: []
    }]}, {
        id: "1800",
        name: "18",
        data: {},
        children: [{
```

```
32          id: "1700",
33          name: "17",
34          data: {},
35          children: []
36      }, {
37          id: "3100",
38          name: "31",
39          data: {},
40          children: []
41      }]}]}, {
42          id: "3800",
43          name: "38",
44          data: {},
45          children: [{
46          id: "3500",
47          name: "35",
48          data: {},
49          children: [{
50          id: "3490",
51          name: "null",
52          data: {},
53          children: []},{
54          id: "3600",
55          name: "36",
56          data: {},
57          children: []
58      }]}, {
59          id: "5600",
60          name: "56",
61          data: {},
62          children: [{
63          id: "5500",
64          name: "55",
65          data: {},
66          children: []
67      }, {
68          id: "5800",
69          name: "58",
70          data: {},
71          children: []
72      }]}]}]};</value>
73      </display>
74      <display>
75          <view>Multiple Choice</view>
76          <value>What should be printed if the binary tree is traversed
77                 using the level-order algorithm?</value>
78          <choice0>2, 3, 4, 8, 17, 18, 31, 33, 35, 36, 38, 55, 56, 58</choice0>
79          <choice1>2, 4, 3, 17, 31, 18, 8, 36, 35, 55, 58, 56, 38, 33</choice1>
80          <choice2>33, 8, 38, 3, 18, 35, 56, 2, 4, 17, 31, 36, 55, 58</choice2>
```

```
81          <choice3>33, 8, 3, 2, 4, 18, 17, 31, 38, 35, 36, 56, 55, 58</choice3>
82          <solution>33, 8, 38, 3, 18, 35, 56, 2, 4, 17, 31, 36, 55, 58</solution>
83      </display>
84      <display>
85          <difficulty>C</difficulty>
86      </display>
87  </exercise>
```

Listing A.1: Example exercise for the Binary Tree exercise category

Strings exercise

```
1  <?xml version="1.0"?>
2  <exercise>
3      <display>
4          <view>CodeEditor</view>
5          <value>public class StringExercise {
6      public static void main(String[] args) {
7          String s1 = "PLASTIC";
8          String s2 = s1;
9
10         s1 = s1.substring(0, 5);
11         s2.toLowerCase();
12
13         String s3 = "TESTER";
14         s1.charAt(1);
15         String s4 = s3;
16         s3 = s1.substring(0, 2);
17         s3.replaceFirst("L", "m");
18         s4.replace("T", "f");
19     }
20 }</value>
21     </display>
22     <display>
23         <view>Multiple Choice</view>
24         <value>What is the correct combination of final values
25                 for each String?</value>
26         <choice0>s4 = fESfER; s2 = plastic</choice0>
27         <choice1>s4 = PL; s2 = PLAST</choice1>
28         <choice2>s4 = PL; s2 = plastic</choice2>
29         <choice3>s4 = TESTER; s2 = PLASTIC</choice3>
30         <solution>s4 = TESTER; s2 = PLASTIC</solution>
31     </display>
32     <display>
33         <difficulty>B</difficulty>
34     </display>
35 </exercise>
```

Listing A.2: Example exercise for the Strings exercise category.

Conditionals exercise

```
<?xml version="1.0"?>
<exercise>
    <display>
        <view>CodeEditor</view>
        <value>public class ConditionalsExercise {
    public static void main(String[] args) {
        boolean var1;
        boolean var2;
        int var3;
        int var4;

        var1 = true;
        var2 = false;
        var3 = 150;
        var4 = 300;

        if (var2) {
            var2 = var1;
            var1 = true;
        } else {
            var3 = var4;
        }

    }
}</value>
    </display>
    <display>
        <view>Multiple Choice</view>
        <value>What is the correct combination of final values?</value>
        <choice0>var1 = true; var4 = 266</choice0>
        <choice1>var1 = false; var4 = 348</choice1>
        <choice2>var1 = true; var4 = 300</choice2>
        <choice3>var1 = false; var4 = 300</choice3>
        <solution>var1 = true; var4 = 300</solution>
    </display>
    <display>
        <difficulty>A</difficulty>
    </display>
</exercise>
```

Listing A.3: Example exercise for the Conditionals exercise category

Appendix B

Item Response Theory Simulation

```
run:
Item 1, a=0.600000024, b=2.0, c=0.25
Item 2, a=1.10000002, b=5.0, c=0.25
Item 6, a=1.20000005, b=6.0, c=0.25
Item 4, a=1.39999998, b=10.0, c=0.25
Item 7, a=1.29999995, b=8.0, c=0.25
Item 8, a=1.29999995, b=9.0, c=0.25
Item 9, a=1.29999995, b=8.0, c=0.25
Item 10, a=1.20000005, b=7.0, c=0.25
Item 3, a=0.899999976, b=4.0, c=0.25
Item 5, a=1.10000002, b=6.0, c=0.25
Item 11, a=0.899999976, b=5.0, c=0.25
Item 13, a=1.20000005, b=6.0, c=0.25
Item 12, a=1.20000005, b=7.0, c=0.25
Item 14, a=1.20000005, b=5.0, c=0.25
Item 19, a=1.10000002, b=8.0, c=0.25
Item 16, a=0.899999976, b=4.0, c=0.25
Item 20, a=1.10000002, b=6.0, c=0.25
Item 17, a=1.0, b=6.0, c=0.25
Item 18, a=1.0, b=6.0, c=0.25
Item 15, a=1.10000002, b=8.0, c=0.25

Initial ability estimate is 5.0
Response pattern: 1,1,0,1,1,0,1,0,0,1,1,1,1

ItemID with max information=14....item difficulty = 5.0
```

```
Answering item correctly...
Level 0; p(X) = 0.03636769115505164
Level 1; p(X) = 0.037206623471527936
Level 2; p(X) = 0.038260317342103396
Level 3; p(X) = 0.04081036191593649
Level 4; p(X) = 0.05349299152794003
Level 5; p(X) = 0.10165259009990486
Level 6; p(X) = 0.1468438899647491
Level 7; p(X) = 0.1455581552386823
Level 8; p(X) = 0.13531360399937065
Level 9; p(X) = 0.12710657491229954
Level 10; p(X) = 0.12100612155365803
After answering item, theta estimate is now 6

ItemID with max information=6....item difficulty = 6.0
Answering item correctly...
Level 0; p(X) = 0.013124740793219158
Level 1; p(X) = 0.013542401445264645
Level 2; p(X) = 0.014057388340824017
Level 3; p(X) = 0.015215361137360064
Level 4; p(X) = 0.02100213737667263
Level 5; p(X) = 0.051797448304890685
Level 6; p(X) = 0.14268364973784606
Level 7; p(X) = 0.20760489960552594
Level 8; p(X) = 0.1916296772375679
Level 9; p(X) = 0.16853841728581867
Level 10; p(X) = 0.15231185483440784
After answering item, theta estimate is now 7

ItemID with max information=10....item difficulty = 7.0
Answering item incorrectly...
Level 0; p(X) = 0.03426092158943807
Level 1; p(X) = 0.03350256540359772
Level 2; p(X) = 0.03313907735209839
Level 3; p(X) = 0.03431631472252459
Level 4; p(X) = 0.04532468027557683
Level 5; p(X) = 0.10467277882139388
Level 6; p(X) = 0.23442418103313054
Level 7; p(X) = 0.16747623219413782
Level 8; p(X) = 0.03676628018469355
Level 9; p(X) = 0.004815402028441693
```

```
Level 10; p(X) = 5.768602180674826E-4
After answering item, theta estimate is now 6

ItemID with max information=13....item difficulty = 6.0
Answering item correctly...
Level 0; p(X) = 0.02028756239063162
Level 1; p(X) = 0.020005962646609306
Level 2; p(X) = 0.019964567998580075
Level 3; p(X) = 0.02095836406139418
Level 4; p(X) = 0.029109831216476114
Level 5; p(X) = 0.08704278780210205
Level 6; p(X) = 0.3676788788796302
Level 7; p(X) = 0.31762619226094463
Level 8; p(X) = 0.058659118650767436
Level 9; p(X) = 0.007504854463641741
Level 10; p(X) = 8.965732158816038E-4
After answering item, theta estimate is now 6

ItemID with max information=5....item difficulty = 6.0
Answering item correctly...
Level 0; p(X) = 0.007921854133211843
Level 1; p(X) = 0.007851530701417237
Level 2; p(X) = 0.007884102039918992
Level 3; p(X) = 0.008393036691102489
Level 4; p(X) = 0.012396255823329205
Level 5; p(X) = 0.048885794487403086
Level 6; p(X) = 0.376655920288866
Level 7; p(X) = 0.4641986540411844
Level 8; p(X) = 0.0770970854357723
Level 9; p(X) = 0.009974176795030516
Level 10; p(X) = 0.0011669386976239583
After answering item, theta estimate is now 7

ItemID with max information=12....item difficulty = 7.0
Answering item incorrectly...
Level 0; p(X) = 0.011877887469279843
Level 1; p(X) = 0.011702978802794266
Level 2; p(X) = 0.011684072562684555
Level 3; p(X) = 0.012365574101473823
Level 4; p(X) = 0.018122648551965423
Level 5; p(X) = 0.06984984926009075
```

```
Level 6; p(X) = 0.47022057826311203
Level 7; p(X) = 0.29318522593170404
Level 8; p(X) = 0.012563089704461445
Level 9; p(X) = 2.3258255458602479E-4
Level 10; p(X) = 3.6644645983882904E-6
After answering item, theta estimate is now 6

ItemID with max information=20....item difficulty = 6.0
Answering item correctly...
Level 0; p(X) = 0.004855919229956476
Level 1; p(X) = 0.0047992460547199075
Level 2; p(X) = 0.004811970748426527
Level 3; p(X) = 0.0051542622134070825
Level 4; p(X) = 0.008016402606395278
Level 5; p(X) = 0.040641160894608244
Level 6; p(X) = 0.49678042634666586
Level 7; p(X) = 0.43378398292523146
Level 8; p(X) = 0.016802005427202126
Level 9; p(X) = 3.139215345169494E-4
Level 10; p(X) = 4.956930909540798E-6
After answering item, theta estimate is now 6

ItemID with max information=17....item difficulty = 6.0
Answering item incorrectly...
Level 0; p(X) = 0.012857979860137338
Level 1; p(X) = 0.012442169512784512
Level 2; p(X) = 0.0122216494348138
Level 3; p(X) = 0.012785562238543855
Level 4; p(X) = 0.019000760832745923
Level 5; p(X) = 0.08203210562478577
Level 6; p(X) = 0.547234697520948
Level 7; p(X) = 0.13984693166143292
Level 8; p(X) = 0.0012510896478464977
Level 9; p(X) = 4.391050717578476E-6
Level 10; p(X) = 1.2729688571107645E-8
After answering item, theta estimate is now 6

ItemID with max information=18....item difficulty = 6.0
Answering item incorrectly...
Level 0; p(X) = 0.029678874782892165
Level 1; p(X) = 0.027641126423564
```

```
Level 2; p(X) = 0.02624051311476951
Level 3; p(X) = 0.026517157227254406
Level 4; p(X) = 0.037304834991676875
Level 5; p(X) = 0.135841766493143
Level 6; p(X) = 0.49202963517146353
Level 7; p(X) = 0.04087346140850065
Level 8; p(X) = 7.872920151215608E-5
Level 9; p(X) = 5.1851911423047384E-8
Level 10; p(X) = 2.7597504092165085E-11
After answering item, theta estimate is now 6

ItemID with max information=2....item difficulty = 5.0
Answering item correctly...
Level 0; p(X) = 0.01218621570180688
Level 1; p(X) = 0.011447973091770907
Level 2; p(X) = 0.011042318093585394
Level 3; p(X) = 0.011882275557934254
Level 4; p(X) = 0.02203393938987174
Level 5; p(X) = 0.1445151599655469
Level 6; p(X) = 0.7467401069214272
Level 7; p(X) = 0.04885256915234908
Level 8; p(X) = 9.460065717340942E-5
Level 9; p(X) = 6.244833073308885E-8
Level 10; p(X) = 3.324920495966649E-11
After answering item, theta estimate is now 6

ItemID with max information=11....item difficulty = 5.0
Answering item correctly...
Level 0; p(X) = 0.003788639987107719
Level 1; p(X) = 0.0035868035884545025
Level 2; p(X) = 0.0035494662597438304
Level 3; p(X) = 0.004215996938004986
Level 4; p(X) = 0.010600922414678006
Level 5; p(X) = 0.11394109994797201
Level 6; p(X) = 0.8364204181699381
Level 7; p(X) = 0.05545773866937719
Level 8; p(X) = 1.0946553006657731E-4
Level 9; p(X) = 7.268882619609612E-8
Level 10; p(X) = 3.875146904084948E-11
After answering item, theta estimate is now 6
```

```
ItemID with max information=3....item difficulty = 4.0
Answering item correctly...
Level 0; p(X) = 9.795168022923113E-4
Level 1; p(X) = 9.497394751603832E-4
Level 2; p(X) = 0.0010356521315317583
Level 3; p(X) = 0.00164706600190618
Level 4; p(X) = 0.006828712076886231
Level 5; p(X) = 0.10200537508217888
Level 6; p(X) = 0.844156542398744
Level 7; p(X) = 0.057034236666960246
Level 8; p(X) = 1.1306244281980126E-4
Level 9; p(X) = 7.517390516847282E-8
Level 10; p(X) = 4.008750839724127E-11
After answering item, theta estimate is now 6

ItemID with max information=16....item difficulty = 4.0
Answering item correctly...
Level 0; p(X) = 2.5500873149534046E-4
Level 1; p(X) = 2.5309431275512853E-4
Level 2; p(X) = 3.039646180160208E-4
Level 3; p(X) = 6.608577852321254E-4
Level 4; p(X) = 0.004419795981432214
Level 5; p(X) = 0.0916758728146013
Level 6; p(X) = 0.8540744186749987
Level 7; p(X) = 0.05867119922756449
Level 8; p(X) = 1.1680139881184215E-4
Level 9; p(X) = 7.775981216800935E-8
Level 10; p(X) = 4.1478074941804744E-11
After answering item, theta estimate is now 6

BUILD SUCCESSFUL (total time: 1 second)
```

Appendix C

IRTModule.java

```
1  public class IRTModule {
2      private void computePosteriorDistribution(double[] knowledgeDistribution,
3                                                Item item, int u_i) {
4          for (int i = 0; i < knowledgeDistribution.length; i++) {
5              knowledgeDistribution[i]
6                      = numerator(i, item, knowledgeDistribution, u_i)
7                      / denominator(item, knowledgeDistribution, u_i);
8          }
9      }
10
11     private double numerator(int theta, Item item,
12                              double[] knowledgeDistribution, int u_i) {
13         if (u_i == 1) {
14             return probabilityCorrectAnswer(theta, item)
15                     * knowledgeDistribution[theta];
16         } else {
17             return (1 - probabilityCorrectAnswer(theta, item))
18                             * knowledgeDistribution[theta];
19         }
20     }
21
22     private double denominator(Item item, double[] knowledgeDistribution,
23                                int u_i) {
24         double denominator = 0;
25
26         for (int i = 0; i < knowledgeDistribution.length; i++) {
27             denominator += numerator(i, item, knowledgeDistribution, u_i);
28         }
29
30         return denominator;
31     }
32
33     private double probabilityCorrectAnswer(int theta, Item item) {
```

```
34          double a = item.getA();
35          double b = item.getB();
36          double c = item.getC();
37
38          double probability = (1 - c) / (1 + Math.exp(-1.7 * a * (theta - b)));
39          probability = c + probability;
40
41          return probability;
42      }
43
44      private int maxItemInformation(int thetaEstimate, ArrayList<Item> itemList) {
45          double maxInformation = -10;
46          int itemID = 0;
47
48          for (int i = 0; i < itemList.size(); i++) {
49              double information = itemInformation(thetaEstimate, itemList.get(i));
50
51              if (information > maxInformation) {
52                  maxInformation = information;
53                  itemID = i;
54              }
55          }
56
57          return itemID;
58      }
59
60      private double itemInformation(int thetaEstimate, Item item) {
61          double information;
62
63          double a = item.getA();
64          double c = item.getC();
65
66          double aSquared = Math.pow(a, 2);
67          double p = probabilityCorrectAnswer(thetaEstimate, item);
68          double q = 1 - p;
69
70          information = aSquared * (q / p);
71
72          double numerator = p - c;
73          double denominator = 1 - c;
74
75          information = information * Math.pow(numerator / denominator, 2);
76
77          return information;
78      }
79
80      private int abilityEstimate(double[] knowledgeDistribution) {
81          int abilityEstimate = 0;
82          double max = -10;
```

```
83
84          for (int i = 0; i < knowledgeDistribution.length; i++) {
85              if (knowledgeDistribution[i] > max) {
86                  max = knowledgeDistribution[i];
87                  abilityEstimate = i;
88              }
89          }
90          return abilityEstimate;
91      }
92  }
```

Listing C.1: Item Response Theory module.