Informatics in Education, 2024, Vol. 23, No. 2, 361–383

DOI: 10.15388/infedu.2024.17

Analysis and Evaluation of a Searchable Exercise Repository for Training Java Programming

Arjan J.F. KOK[1,*], Lex BIJLSMA[1], Cornelis HUIZING[2], Ruurd KUIPER[2,1], Harrie PASSIER[1]

[1]*Department of Computer Science, Faculty of Science, Open Universiteit*
Heerlen, The Netherlands
[2]*Faculty of Mathematics and Computer Science, Eindhoven University of Technology*
Eindhoven, The Netherlands
e-mail: arjan.kok@ou.nl, lex.bijlsma@ou.nl, c.huizing@tue.nl, r.kuiper@tue.nl, harrie.passier@ou.nl

Received: July 2023

Abstract. This paper presents the first experiences of the use of an online open-source repository with programming exercises. The repository is independent of any specific teaching approach. Students can search for and select an exercise that trains the programming concepts that they want to train and that only uses the programming concepts they already know, then submit their solutions, and get automatic feedback from the system. We analyzed quantitatively how students used the system by inspecting the logged actions of the students using the system. We also did a qualitative analysis by interviews, to find out how the students appreciated the use of the repository and to get feedback for improvements. We focused on how students select exercises as finding the exercise that fulfils the training needs of a student is the innovative part of our repository.

Keywords: Computer science education, programming, exercise repository, exercise selection, tool evaluation.

1. Introduction

Learning programming implies doing many programming exercises. For students to obtain the right training, there is a high demand for a large number of good quality programming exercises. Creating good quality exercises is not easy and a time consuming task for teachers. Sharing exercises between teachers of different institutions can be a solution to reduce the time for creating exercises by individual teachers. Giving feedback on the solutions of students is another time consuming task and should be automated.

[*] Corresponding author.

The Structured Exercise Repository with automated Feedback (SERF Repository) provides an online open source repository of Java programming exercises. Teachers at different institutions can upload their exercises to this system. Students can search for exercises that match their training needs, submit their solutions, and inspect the automatic feedback they get from the system.

The SERF repository differs from other systems that enable students to do exercises (see for an overview of these systems Paiva *et al.* (2022) and Keuning *et al.* (2019)) in the way the students select their exercises. In most other systems, the teacher selects the exercises the student has to solve, or the student can select the exercise from one or more lists, where exercises are grouped by one specific concept. For example when the concept array is chosen, all exercises training use of the array are shown. In some cases, an extra difficulty level is shown. However, these approaches do not take into account the use of combined concepts and the concepts a student has not mastered yet.

The SERF repository enables to match the training needs and the existing knowledge level of a student with the appropriate exercises more precisely. Each exercise in the repository is tagged with the concepts it trains and the prior knowledge needed to be able to solve the exercise. For example, an exercise may train several concepts like while, method and object, and may require prior knowledge like a thorough understanding of boolean conditions, and a glancing acquaintance with the main method as occurring in template code. A student searching for an exercise can specify the concepts he/she wants to train, for example loops. The repository will then show the appropriate exercises, with the knowledge each exercise requires. To reduce the set of exercises to the ones the student is able to do, the student can also specify the concepts he/she is not familiar with or has not mastered yet. The repository will then exclude all exercises that require knowledge of these unfamiliar concepts from the list of appropriate exercises. For example, when the student specifies that he/she has not mastered the concept object, the repository excludes all exercises that use objects. This way the repository is independent of specific teaching approaches (like the order of concepts introduced in a course). It accommodates, for instance, both ‘objects first’ and ‘objects late’ courses equally well. Therefore it is possible that teachers using different teaching approaches all can contribute exercises to the system. Through the advanced search method, one or more exercises will pop-up when the student is ready to solve it/them.

This search approach is not restricted to programming exercises. For any knowledge domain, where students have to search for exercises in a repository, based on concepts and existing knowledge of the student, this search mechanism can be applied.

Solutions to exercises can be submitted and, using teacher defined test cases, feedback is given to the student. Submitting solutions to the exercises and the form of feedback of the repository is similar to that provided by the online tool CodingBat. The feedback shows compilation problems, and when these are not present, the problems with the functionality, mostly in the form of expected versus calculated output.

Our research goal is to obtain insight in the use of the repository by students and find directions for improvement of the system. To obtain this insight we want answers to questions like: Do students use the repository as intended? Do they use the advanced search function? What do students like and dislike? What can be improved?

We propose an evaluation methodology to get answers to these questions. For this evaluation students are offered the repository during first-year programming courses at three different institutions. The exercises are supplied by several teachers of two of these institutions. At all institutions the use of the repository is voluntary. It provides extra training in addition to the exercises normally used in the courses. The use of the repository is observed quantitatively in terms of logged actions of students: number of searches, selections, submissions, etc. Interviews are conducted to supplement the results of the quantitative analysis with qualitative information. These interviews provide revealing anecdotal information.

Section 2 describes related work. In section 3 we describe the SERF repository in more detail. It shows how a teacher can add an exercise to the repository and how a student can select an exercise and submit a solution for it. Section 4 describes the methodology used to analyze the system. Section 5 shows the results of this analysis. A discussion of these results, and ideas for future work are given in section 6.

2. Related Work

A large number of systems for automated assessment of programming exercises exist. Several literature reviews give an overview of these systems, for example (Ala-Mutka, 2005; Ihantola *et al.*, 2010; Keuning *et al.*, 2019; Souza *et al.*, 2016; Paiva *et al.*, 2022). These reviews classify these systems based on technical and pedagogical features.

Ala-Mutka (2005) describes the characteristics of the systems by dynamic (e.g. functionality and efficiency) versus static (e.g. coding style, metrics) assessment, automatic versus semi-automatic assessment, and formative versus summative assessment. Ihantola *et al.* (2010) further discuss the features and approaches supported by automated assessment tools. Included features are which programming languages are supported, whether the system can interoperate with a learning management system, how tests are defined, how resubmissions are controlled, whether there is a possibility for manual assessment after automated assessment, how security is guaranteed, whether the system is freely available for others, and how to assess special types of programming assignments. Keuning *et al.* (2019) classify the tools based on how and which feedback is generated. A recent and comprehensive overview of the state of art of automated assessment tools in computer education is given by Paiva *et al.* (2022). Their focus is on five aspects of these tools, namely supported exercise domains, testing techniques utilized, security measures adopted, feedback produced, and the information they offer the instructor to understand and optimize learning.

This last topic, the collection and analysis of data about the behavior and actions of students (learning analytics), can be used to optimize the learning process of students and to improve the tools used. Examples of these data are submission history and completed tasks. Some learning analytics tools collect data of more fine-grained event streams, for example, every key-press made while a student is working on a task (Vihavainen *et al.*, 2014). Insight is gained into the extent to which students work on assignments that are not submitted. If only submissions are stored, no trace of such work is recorded, and the students that struggle on an assignment but never submit it are out of scope. Another development is that the analysis of collected data makes it possible to relate observational data, i.e. the time spent to complete exercises, coding style and ratio of syntax errors, to study success, final grades and learning motivation (Chen *et al.*, 2020; Munson and Zitovsky, 2018).

The repository presented and evaluated in this paper can be categorized as an automatic formative assessment tool that provides feedback regarding compiling and functional errors on (re-)submissions of Java programming assignments. It collects data of the actions of the students which enables analysis of the tool and the student's progress. The innovative feature of the repository is the search mechanism by which students can select the appropriate assignments.

Several exploratory studies evaluating the usefulness and effectiveness of assessment tools have been done. Some of them, e.g. Spacco *et al.* (2006), use surveys amongst students and/or teachers, to find out whether the tools were helpful in learning and/or teaching programming. Others, e.g. Spacco *et al.* (2015); Munson and Zitovsky (2018), analyze the data collected by the system, showing how students used the system. Like Yan *et al.* (2020), we evaluate our repository both by surveys and by analyzing the collected data. Of course, we emphasize in our evaluation the analysis of the unique search mechanism.

3. The SERF Repository

The Structured Exercise Repository with automated Feedback (SERF) is an online database containing Java exercises to support training of (Java) OO programming skills (SERF Repository). To make the repository teaching-approach independent, there is no approach-linked ordering or grouping of the exercises, but the SERF repository has a search function based on tags that enables to select individual exercises by training desire. To provide training in a manner that needs relatively little teacher support, e.g., in an online setting, solutions to the exercises can be submitted by a student to the SERF repository. JUnit tests are performed on the solutions, generating automatic feedback.

A short description of the ideas is given below. More details can be found in Bijlsma *et al.* (2021).

3.1. *Adding Exercises*

A teacher submits an exercise description, a code template, knowledge tags, and the unit tests to test the solutions of students to the system. The knowledge tags describe which knowledge is trained by this exercise and which knowledge a student needs to solve the exercise successfully. Template code consists minimally of the signatures of the methods the student has to write for the exercise. The exact signatures are needed to apply the unit tests for feedback.

To ensure the quality of the repository, each exercise is reviewed by another teacher before it is made available to the students.

3.2. *Tags and the Knowledge Graph*

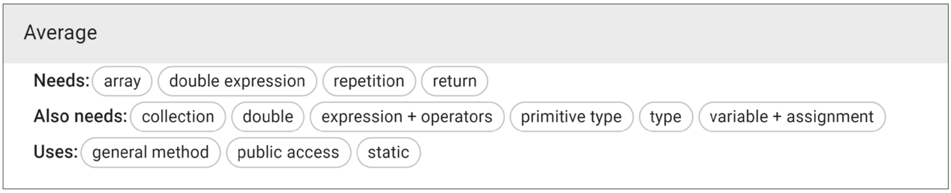
To enable to select individual exercises for the desired training and to indicate which knowledge is needed to solve the exercise successfully, tags are used. A tag is a word that characterizes a (programming) knowledge item. The set of tags in our system for Java is the, carefully selected, set of words that identify programming knowledge items that pertain to Java. Syntactic (e.g. while) as well as semantic or conceptual (e.g. repetition) knowledge items are used.

Knowledge tags can be applied at two levels. A 'needs' tag indicates that the student needs to know everything about the knowledge item and can apply it. A 'uses' tag indicates that the student needs some glancing acquaintance of the item. For instance, anyone writing a Java program with a main function will encounter the terms *public* and *static*, without the need to understand their full meaning.

Simply tagging each exercise with all knowledge needed or used is impractical. An exercise may require many knowledge items at very different levels. For example, an exercise in which the *while* is trained, requires also knowledge of *boolean expression*, which in turn requires knowledge of *boolean* and *expression*. These problems are solved by ordering the set of known tags in the system with a prior knowledge order between the knowledge items. This results in a knowledge graph that records the dependencies between knowledge items in terms of needs and uses relations. Only dependencies that are intrinsic to the domain (in our case Java programming) are recorded in the knowledge graph. That guarantees that the resulting knowledge graph is an acyclic directed graph (in the direction from high-level concept tags to low-level concept tags) that is independent of any teaching approach. Details and proof can be found in Bijlsma *et al.* (2021). Our knowledge graph is specific to Java. However, for similar languages, e.g., Python, similar graphs can easily be constructed. In fact, for any knowledge domain that has prior knowledge relations between items in the domain a similar graph can be constructed.

The notion of prior knowledge is used for the tagging of exercises. At submission of an exercise to the repository, the teacher only needs to provide the top-level tags (needs and uses tags). All other tags are automatically derived by the system using the knowledge graph. Selecting only top-tags in the ordering for tagging exercises keeps the task of tagging manageable for the teacher. Note that students do not need to be aware of the existence of the knowledge graph.

Fig. 1 shows the tags of an exercise to write the body of a method that computes and returns the average of the values in an array. To solve this exercise the student needs knowledge of arrays, repetition, return and double expression (for the calculation of the average). The signature of the method is given in the template, so the student needs some knowledge of the items that are part of this signature, but the student does not need to write it. Therefore, the knowledge items that are part of the given signature are



Average

Needs: array double expression repetition return

Also needs: collection double expression + operators primitive type type variable + assignment

Uses: general method public access static

Fig. 1. Tags for an exercise.

specified as ‘uses’ tags. The tags indicated by ‘also needs’ are automatically derived by the system using the knowledge graph. These items show prior knowledge needed to understand the teacher provided ‘needs’ concepts. Apart from ‘also needs’ tags the system can also derive ‘also uses’ tags when the student only needs a glancing acquaintance of the derived prior knowledge.

A student can use the tags to search for the exercises that train the desired knowledge, but also exclude the exercises for which the student does not have all knowledge (by using the tags negatively).

3.3. *Searching for and Solving Exercises*

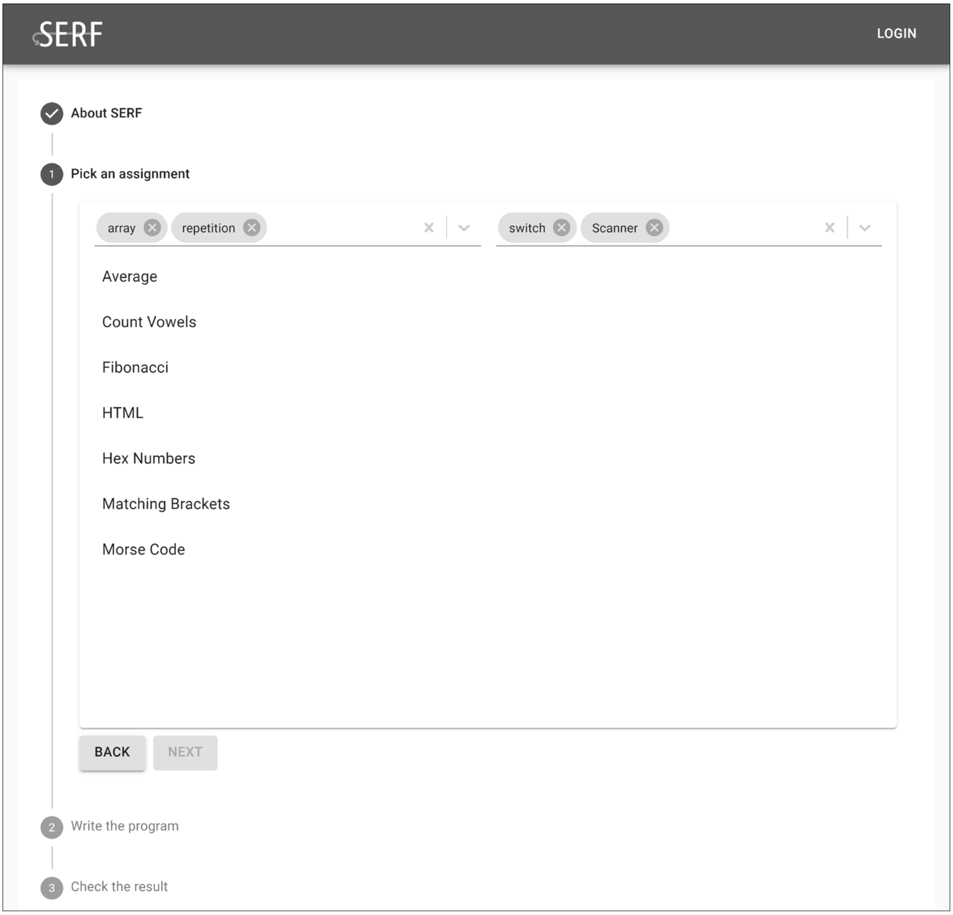
At the start of a student session, the system shows a list with the names of all available exercises. The student can browse over the names, and when moving over a name, the tags (knowledge items) for that exercise are shown, as in Fig. 1. With a large list of exercises, finding a suitable exercise this way is time-consuming.

Therefore, students can select one or more tags directly for the knowledge they want to train. Then, only the exercises in which this knowledge is needed, are shown. The student can inspect this new list and see what knowledge is needed for each of the exercises by moving over the exercises’ names of the new list.

When the list is still to large to judge which of exercises in the lists are appropriate, the student can also select negative tags. The system will then remove all items that require the direct or indirect knowledge of these negative tags. By repetitively selecting positive and negative tags, the student will find the most suitable exercises.

Fig. 2 shows a situation where a student wants to train iteration over the elements of an array (positive tags *array* and *repetition*), but has at this moment no knowledge of the switch-statement and the Scanner object (negative tags *switch* and *Scanner*). Seven exercises are available to fulfil the training needs with the student’s current knowledge.

When a student has found a suitable exercise, he/she can open it. The text of the assignment and the template code is shown, see Fig. 3. The student can now complete the template code to add the requested functionality. When finished entering code, the solution can be submitted. The system checks the code and gives feedback on the solution. This feedback consists of a fail/pass indication, any compiler errors, and the results of the unit test cases. The repository does not grade the submission. When the code does not pass all tests, the student can adjust the code and resubmit it to iteratively improve the solution.



SERF

LOGIN

About SERF

1 Pick an assignment

array repetition

switch Scanner

Average

Count Vowels

Fibonacci

HTML

Hex Numbers

Matching Brackets

Morse Code

BACK NEXT

2 Write the program

3 Check the result

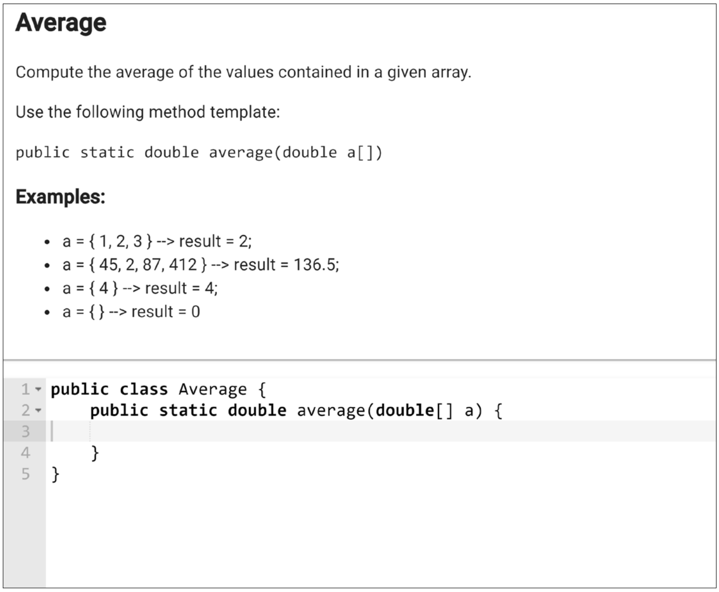
Fig. 2. Searching for an exercise. The tags on the left (*array* and *repetition*) are the selected positive tags, the tags on the right (*switch* and *Scanner*) the selected negative tags.

4. The Evaluation Methodology and Set-up

This study aims to get insight in the use of the SERF repository. We want answers to the following research questions:

1. To what extent is the repository used?
2. Do the students use the repository as we expected, i.e. as described in section 3.3?
3. Does the repository support the students in learning programming in Java?
4. How user friendly is the use of the repository?

It is important to note that in research question 3 we do not assess enhancement in student's programming skills. That is outside the scope of our current exploratory research. Our focus is solely on determining whether students perceive improved support



Average

Compute the average of the values contained in a given array.

Use the following method template:

```
public static double average(double a[])
```

Examples:

- a = { 1, 2, 3 } --> result = 2;
- a = { 45, 2, 87, 412 } --> result = 136.5;
- a = { 4 } --> result = 4;
- a = { } --> result = 0

```
public class Average {
    public static double average(double[] a) {

    }
}
```

Fig. 3. Example of an exercise with a template in which the student can enter the solution.

for training solving programming exercises, for example, that they are more confident and are better prepared for the exam.

We apply quantitative as well as qualitative analysis. The first two questions are answered by the quantitative analysis by means of logged data, the last two by the qualitative analysis through interviews.

The repository was offered to students following an introductory course in object oriented programming with Java at three different institutes, see Table 1. Students of Institute A and Institute C study full-time, where most students of Institute B study part-time alongside a (full-time) job. Each institute uses a different teaching approach, so the order in which programming concepts are introduced in the courses differ.

The repository was offered at the beginning of the course periods, all of a duration of three months. Students were introduced to the repository during the first lecture. It was presented as an option to do exercises in addition to the exercises in the course materials. So it was not obligatory to do these exercises. Scenarios of use were shown: how to search for exercises that fulfil the training needs, how to exclude exercises requiring

Table 1

Institutes in the evaluation

	Name	Description
Institute A	NHL Stenden	University of applied sciences
Institute B	Open Universiteit	Distance education university
Institute C	Eindhoven University of Technology	University of technology

unknown knowledge, how to open an assignment, how to fill in the exercise template, how to submit a solution, how to get feedback on the solution submitted, how to improve a solution after feedback, and how to submit the solution again. At the distance university (Institute B), a recording was made of this meeting. So, students not present at the lecture could watch this recording later.

A manual describing how to use the repository was available to the students via both the repository itself and the courses' websites. Lecturers from Institute A and Institute B provided the students with an overview of the meaningful prior knowledge search terms for each unit of study, as an aid in the search for suitable exercises in specific areas when studying the course.

During the course period the following actions of students were logged:

- Access of the main page with the overview of all exercises.
- Adding or removing a search tag to the search, both positive (exercise should train this knowledge) or negative (exercise should not require this knowledge).
- Opening an exercise, where the student gets the complete text of the exercise and a template to fill in the solution.
- Editing a solution.
- Submission of a solution, including the code the student provided and the feedback from the system.

All logged actions are time stamped. Therefore we can deduce all kinds of information from these logged actions, for example, did the student select the exercise by browsing the complete list, or did the student select the assignment after searching using tags, and if so, which tags the student used. A quantitative analysis of the logs of the actions can be found in section 5.1. With the logged data we are able to answer the first two research questions.

After the course period, a subset of participants was interviewed using a semi-structured interview to obtain more in-depth knowledge of the students' experiences with the repository. The interview guide followed during the interviews can be found in Appendix A. All interviews were conducted by two interviewers, one mainly the panel chair and one mainly making notes. All interview sessions were recorded. A qualitative analysis of the repository based on the answers from the students during these interviews can be found in section 5.2. This qualitative analysis answers research questions three and four. Participation in the interviews was voluntary.

Some notes on the implementation of the evaluation:

- The course of Institute A started first, and provided the first logged data. We noticed that not all important information to do a proper evaluation was logged. Therefore the logged data of this first run are not used in the quantitative analysis. The interviews with the Institute A students are included in the qualitative analysis. We used the acquired results of Institute A to improve the repository, mainly by extending the logging, for the evaluation runs at Institute B and Institute C.
- Due to COVID19, the students at Institute C were, atypically, forced to online instruction and use of the repository. Students at Institute B are used to online instruction.

Table 2

Student participation

Students	Institute B		Institute C	
offered repository	173		350	
accessed repository	31	18%	94	27%

5. Analysis of Use of the Repository

The courses at Institute B and Institute C where scheduled from September 2020 until November 2020. In that period the SERF database contained 39 exercises, constructed and reviewed by lecturers from both institutions.

Table 2 shows the number of students that were offered the repository versus the number of students that actually used the repository. Students used the repository less than expected. We have no direct explanation. It might be that the courses themselves contain sufficient exercises, so most students have no need for extra exercises (although each year several students ask for extra exercises). Or maybe the students do not have the time to do extra (not scheduled) exercises. An important part of the Institute B students already has experience with programming and some even program for their profession. These students have, in general, no problem passing the exam of the course, and therefore, do not need extra training. This might explain the lower participation of Institute B students compared to Institute C students.

5.1. *Quantitative Analysis*

We analyzed the logging from September 2020 until January 2021. That includes the use of the repository by students that failed their first exam and prepared for a second exam.

5.1.1. *Executed Actions*

Table 3 shows the most important actions executed by the students. It shows for each type of action (search, open and submit):

- The number of individual students that executed this type of action,
- The number of individual exercises on which this type of action was executed,
- The total number of actions of this type summed over all students.

The first line with totals shows the total of all students that used the repository, the number of available exercises in the repository, and the total number of all actions by all students. Note that this total number of all actions is larger than the sum of the different categories (search, open, submit), as we also logged other actions (edit, feedback, ...) that are ignored in this evaluation.

Table 3
Student actions

Action	Institute B						Institute C					
	Students		Exercises		Actions		Students		Exercises		Actions	
Total	31		39		1573		94		39		1471	
Search												
total	16	52%			167	11%	27	29%			135	9%
positive	13	42%			75	5%	25	29%			95	6%
negative	10	32%			71	5%	3	3%			32	2%
positive + negative	6	19%			21	1%	2	2%			8	1%
Open												
total	26	84%	29	74%	488	31%	63	67%	36	92%	509	35%
without search	19	52%	24	62%	385	24%	60	64%	35	90%	482	33%
after search	14	45%	19	49%	103	7%	12	13%	12	31%	27	2%
Submit												
total	19	61%	24	62%	409	26%	31	33%	28	72%	362	25%
without search	14	45%	20	51%	302	19%	23	24%	27	69%	312	21%
after search	9	29%	13	33%	74	5%	4	4%	3	8%	11	1%
without edit	15	48%	17	44%	33	2%	19	20%	17	43%	39	3%

Searching

Finding a suitable exercise can be done in two ways: using the search function by selecting positive and negative tags, or browsing through the complete list of exercises (where the tags are also shown, so students see directly the prior knowledge needed for an exercise).

- Only a limited number of students actually finds an exercise through the search function (= open after search). Of them, only a few use the combined positive and negative search. That is rather disappointing as this kind of search should lead the student to the most suitable exercises. Also the number of search actions is rather low, as finding one exercise is an incremental process that consists of several individual actions. Each addition of a positive or negative search tag is counted as one action.
- Most students find an exercise by selecting it from the complete list of exercises (= open without search). Supposedly, the prior knowledge information displayed during browsing is sufficient to select the requested exercise. In that case the tagging is used in another way than we expected.

At this moment the number of exercises in the repository is limited to 39. Therefore, browsing to find an exercise is manageable. We expect that with more exercise in the repository the need for the search function, including combined tagging, increases.

Opening and submitting

On average 70% of the times an exercise is opened, the student also submits a solution (after editing this solution).

Remarkable is the rather large number of submissions of an exercise without editing. That means that students submit the given template of the solution. We think that there are two explanations for this behavior:

- Students only explore the system and are not interested in seriously training their programming skills at this time.
- Students want a hint to the solution before they try to make their own solution to the exercise. When a student submits a solution to an exercise, he/she gets some feedback, e.g. what the correct output should be.

Difference between institutes

There are some differences between Institute B and Institute C students:

- On average an Institute B student executes more actions (±50 total actions per student) than an Institute C student (±15 per student). For all separate types of actions, we also see that Institute B students execute more actions.
- The percentage of students that uses the search function is larger for Institute B students (52%) than for Institute C students (29%). Also more Institute B students than Institute C students use a negative search. It seems that Institute B students are better instructed to use all aspects of the repository.
- There is a rather large group of students (16% Institute B, 33% Institute C) that only opens the repository and browses the list of exercises, but never opens an exercise.
- After opening an exercise, institute B students submit a solution more often than Institute C students. Not only the percentage of submit actions compared to the open actions is higher (77% versus 63%), but also the percentage of students that submit after opening (73% versus 49%). So, Institute C students more often only inspect the exercise without solving it.
- Institute C students open more different exercises and submit more solutions to them from the collection of all available exercises (36-28) than Institute B students (29-24). An explanation is that several exercises require prior knowledge of a Java class (Scanner) that is used at the start of the Institute C course. As this knowledge is not part of the Institute B course, Institute B students are less inclined to select these exercises. The tag helps Institute B students to ignore these exercises.

5.1.2. *Student Behavior*

Not only the number of actions is important, but also the sequence of the actions of each individual student and the actual search tags used enable us to explain the behavior of the students. Therefore we also scanned the logged data. In general, we found two types of behavior:

- A large group of students only explores the repository. They browse through the exercises. Some of them open a few exercises, sometimes use a search tag (mostly randomly), sometimes submit the template but never do a serious attempt to solve

Table 4

Student actions; serious students only

Action	Institute B						Institute C					
	Students		Exercises		Actions		Students		Exercises		Actions	
Total	8		39		1156		9		39		913	
Search												
total	7	88%			98	8%	4	44%			10	1%
positive	6	75%			48	4%	3	33%			9	1%
negative	6	75%			38	3%	1	11%			1	0%
positive + negative	3	38%			12	1%	0	0%			0	0%
Open												
total	8	100%	27	69%	399	35%	9	100%	32	82%	316	35%
without search	8	100%	24	62%	320	28%	9	100%	32	82%	314	34%
after search	6	75%	15	38%	79	7%	1	11%	1	3%	2	0%
Submit												
total	8	100%	23	59%	347	30%	9	100%	26	67%	261	29%
without search	8	100%	20	51%	260	22%	8	89%	25	64%	233	26%
after search	6	75%	11	28%	64	6%	1	11%	1	3%	2	0%
without edit	7	88%	12	36%	23	2%	8	89%	13	33%	26	3%

an exercise. Several of these students exactly copied the actions from the instruction material. Most of them accessed the repository only once (of which several at the day of the instruction). Evidently, for these students the repository does not have added value.

- A limited group of 'serious' students (8 of 31 for Institute B, 9 of 94 for Institute C) shows the expected training behavior. They access the repository more than once during the course, and do several serious attempts to submit a good solution to more than one exercise. A number of these students even used the search facilities. Table 4 shows the actions of the students in this group. The differences between the two institutes are much smaller for this group than for all students. It only remains remarkable that the serious Institute C students almost do not use the search function.

5.1.3. *Student-exercise Submissions*

Table 5 shows the statistics of how students dealt with the different exercises, both for all students, and for the serious students only.

Figures 4 and 5 show for each exercise the number of students that made a submission for this exercise (did at least one submission), respectively the total number of submissions for this exercise. In both figures all students are counted (so not only the serious students). The exercises are in alphabetical order, the same order in which the exercises are shown to the students when they are browsing through the exercises. Note that exercises for which no solutions are submitted are not shown in these figures.

Table 5

Student-exercise submission statistics

Students	Institute B		Institute C	
	all	serious	all	serious
number of students that submitted an exercise	19	8	31	9
number of unique student-exercise submission combinations	65	52	78	47
number of different exercises a student submitted				
maximum	17	17	24	24
average	3.4	6.5	2.5	5.2
number of submissions of student to make one exercise				
maximum	28	28	22	22
average	6.3	6.7	4.6	5.6

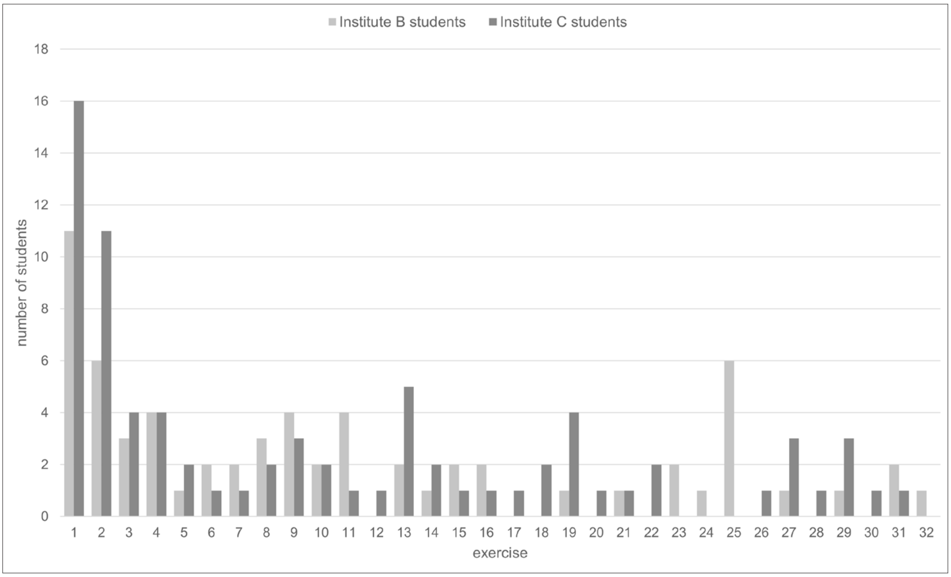


Fig. 4. For each exercise the number of students that submitted a solution.

As expected both figures have more or less the same shape: when more students start with an exercise, the total number of submissions is larger. Some observations can be made:

- The number of students and number of submissions are highest for exercises 1 and 2. Most likely this is caused by the fact that these exercises are the first a student sees (exercises are shown in alphabetical order), when exercises are selected by browsing through the complete list of exercises (instead of by selecting tags), what most student do. However, both exercises are not very difficult and require not much prior knowledge. That might also be a reason why these exercises are used most.

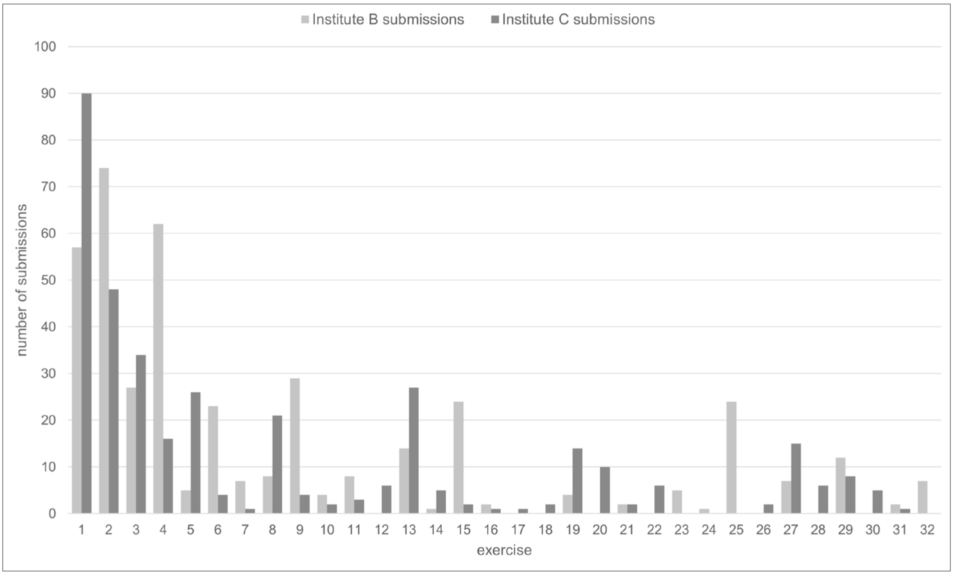


Fig. 5. For each exercise the total number of submit actions.

- There is a number of exercises solved only by Institute C students (exercises 14, 17, 18, 20, 26, 28 and 30). Earlier, it is mentioned that these exercises contained prior knowledge not known by Institute B students. Remarkable is that Institute C students did only a few submissions for these exercises.
- There is a number of exercises made only by Institute B students (exercises 23, 24, 25, and 32). Closer inspection shows that three of them are exercises that were made especially to support Institute B students at the start of the course. This indicates that students can find the exercises that suit them.

5.2. *Qualitative Analysis*

After the students used the SERF repository, six students were interviewed, about 20 minutes per student: four students of Institute B and two students of Institute A. Due to the COVID19 pandemic, we couldn’t interview students from Institute C. The selected students were ‘serious’ students. What follows is a summary of the results per main question. The interview set up can be found in Appendix A.

Opinion on functionality of the repository

All students regard the repository as ‘good’ and ‘supportive’. Exercises show clearly ‘what is the exercise about’: ‘not only *array*, but *array* with a *for* statement’. The repository is easy to use: ‘I did not need help for using the tool’, ‘no supervision was needed’, ‘it was a practical tool, easier than expected’.

Generally, students find the relevant exercises and mention they experienced the search function positively: ‘The tool is conveniently arranged, the tags, searching an exercise is clear’. For some of them, it is difficult at the beginning: ‘I did not understand it completely because all the terms were new, for example Scanner, but later I did understand it, due to explanation and knowing the concepts’. Clearly, this student does not understand the use of the search function of the repository, as he/she could have excluded the concepts he/she is not familiar with. In this case the student should have used tag ‘Scanner’ negatively. Two students sometimes experience difficulties with finding the right assignments: ‘... extra explanation is needed to be able to find the right exercise’. One of them mentions a solution: ‘... each exercise can be provided with a number, so that my teacher can point easily to an exercise ...’. Again, these students do not understand the use of the repository. It is the intention of the repository that each student can find their own exercises using the tags.

One student says: ‘I had to check many tags to select an exercise. This works in the end well. I got the exercises I want to make. In CodingBat this works more easier, because there the exercises are grouped in categories.’ However, the repository already provides this kind of functionality. The student can find the exercises in a category by selecting the tag representing this category.

One student misses a selection on level of difficulty. One student says: ’I did not use the search options of the search function. I think it can be useful, ... The number of exercises was not big, with more exercises I will use the functionality’.

Opinion on the content of the database

All students experience the collection of exercises as ‘good’, ‘very good’ or ‘relevant’. One student says: ‘All the topics were in the system. I did not miss a topic’. Another student says: ‘The number of exercises was not huge, but the exercises are challenging in comparison to CodingBat’.

Four students experience the exercises as clear: ‘the exercises were clear’, ‘description, everything was written down good’ and ‘no difficulties in understanding’.

Some students encounter difficulties with the exercises: ‘Exercises can be more concretely formulated’ and ‘Formulation is good, but not complete. For example, you have to consider robustness, but this was not clear from the exercise description. You can express this using some levels’. One student says: ‘It was always clear what to do, but not always which output was expected. Hints can be helpful here’.

Generally, the degree of difficulty is appreciated. One student says: ‘The exercises are a bit difficult. Some easier ones should be added’. Students often mentioned CodingBat spontaneously and compared the SERF repository with it. Students mention further that SERF has a greater variety of assignments with a higher level of complexity. Codingbat on the other hand has more exercises, but these are often similar to each other and are of a low level of complexity.

Topics missed are Java’s memory model and GUI programming with Swing. Both topics are important parts of the Java course of Institute B. Also, the number of exercises about designing and implementing classes, i.e. adding methods, constructors, and attributes, is experienced as low. It should be noted that it is difficult to automatically test the results of these kinds of exercises. It borders on what is technically possible and feasible in practice.

Opinion on the feedback

Two students are positive about the feedback functionality: ‘It was good, you get insight in the quality of your solution, for example the solution can be less complex or you missed test cases’ and ‘In comparison with CodingBat it is more clear. Besides correctness you get also information about the quality of the your solution. But, more feedback is desirable, although I know that will be a difficult job.’.

The other students are less positive about the feedback functionality: ‘I often found it disappointing, it was not clear what was wrong’, ‘The feedback was very short, you have to find out things yourself’, ‘Compile errors were clear, but the messages about semantic errors were not clear’, ‘Codingbat gives more information by showing all test cases and indicating which of the test cases succeed or not succeed’. Indeed, there is a difference in feedback between assignments at the moment. In some assignments, the feedback is based on only one test case.

It appears that feedback is appreciated even in the present limited form. It also is clear that the quality of feedback is important for student use and acceptance. It would be useful to invest more research effort in both clarifying what exactly the student needs are and how to provide for these in the tool. Improvement of feedback is surely planned as future work. We will discuss this issue further in Section 6.

Opinion on added value for study

All students find the SERF repository helpful for their study: ‘It is helpful to understand the subjects’, ‘more opportunity for practicing’, ‘I advise this tool to others’.

Advice on future improvements

Students mention three improvements:

1. More explanation and supervision, for example by means of a video, especially at the beginning about the use of the tags.
2. More connection between the repository and the course: which exercises can be made after a certain chapter?
3. The possibility to get the standard solution in case you are not able to solve an exercise.

Improvements 1 and 2 were present at Institute B. There was a video recording available of the meeting in which the use of the SERF repository was shown. For each learning unit of the course a list of the relevant tags was provided.

Concerning improvement 2, one can doubt whether the student understands the tool. The student should know which topics have been lectured and can therefore search for those topics with those tags.

6. Discussion and Future Work

6.1. *Discussion*

In section 4 we formulated our research questions. We now discuss the answers to these questions.

To what extent is the repository used?

Each year students ask their teachers for supplementary exercises. So we expected that they would benefit from a repository with extra exercises. However, the repository is less used than expected. Only 24% of the students accessed the repository, and more dramatically, only 3% used the repository seriously to train their skills by accessing the repository more than once and submitting several solutions to the exercises. There might be several reasons why students do not use the repository:

- The courses themselves contain sufficient exercises. This is in contrast with the demand for more exercises by a few students each year. Maybe we overestimated the number of students that needs extra exercises. Also, a number of students, especially from Institute B, already has programming experience and do not need extra training.
- Students do not have the time to do extra exercises.
- The repository is not easy to use or students do not know how to use the tool.
- The exercises in the repository are not interesting or not of the correct level. However, all interviewed students were satisfied with the collection of exercises.

The last two items might explain why students only investigate the repository and not continue submitting solutions.

Do the students use the repository as expected?

Generally, the students do not use the repository as expected. The main feature of our repository, the search function, is not used (as much) as we expected. However, students find the exercises they want to make by browsing through the list of exercises. Apparently, at this moment the information during browsing, that includes the learning goals and prior knowledge, is sufficient to find the suitable exercises. The analysis shows that some exercises are mainly used by Institute B students and other mainly by Institute C students, according to the order of instruction of the concepts in both courses. The tags assigned to the exercises seem to work, but in another way than expected. Likely the search function will be used better when the number of exercises in the repository increases.

From the interviews it became clear that a few students did not understand the ideas behind the repository, and therefore did not use is as required. Institute B students that got the most elaborate instruction used the repository more in accordance with our expectations than Institute C students. Clearly, good instruction is important.

Does the repository support the students in learning programming in Java?

All interviewed students regard the repository as useful and an added value to their study. However, the interviewed students are all students that used the repository extensively. We do not know the meaning of the students that did not use the repository.

How user friendly is the use of the repository?

In general students regard the repository to be easy to use and user friendly.

Other observations

Apart from the answers tot the research questions, the execution of the evaluation and the analysis produced other remarks and insights.

Students ask for easier small exercises. We agree that more small exercises that are focused on one concept, so with only one (or a few) top-level tag(s), will benefit the range of exercises in the repository, and will help the student master that concept.

Students miss some topics in the repository. The missed topics are primarily issues regarding the design of the program. For feedback we use JUnit that tests input-output combinations independently of the design of the program. We have to conclude that we cannot give automatic feedback on exercises that focus on design, where students have to design classes and their methods, as long as we only use JUnit for feedback.

A suggestion is made to add difficulty levels to the exercises. However, difficulty is subjective. Therefore it is hard to assign a difficulty level to an exercise. Also, because we support different orders in which concepts are treated, an exercise can be difficult for a student of one course, while the same exercise can be easy for a student of another course. On the other hand, students already have an indication of the difficulty of an exercise: the number and complexity of the concepts (tags) assigned to it.

The quality of the feedback differs from exercise to exercise. For some of the exercises it is minimal (result of one test case without any explanation), while for others it is very extensive (results of several test cases including some explanation). There is a review process before an exercise is submitted, but apparently the test code and feedback is not always reviewed properly. The review process should be applied more strictly.

6.2. *Future Work*

Current evaluation of the repository has led to several directions of future work. We will improve the repository to make it more interesting for students to use it. This means improving the tooling, the exercises and procedures how to use the repository. And after the improvements we have to evaluate whether they contributed to the acceptance of the repository by the students.

Tool improvement

We discovered that quite a lot of information was present in the logs, but also that some was missing. The logging of a submission should therefore be extended with a ‘pass’ notification. Then we can measure the progress of individual students, and also get information on the suitability of exercises, for example the average number of submissions before an exercise is successfully solved.

Apart from functional correctness, a submission by a student can also be judged on programming style. We will integrate a tool that assesses the submission on programming style, for example Checkstyle. As different institutions impose different requirements for programming style we will investigate the possibility to automatically apply the style rules imposed by the institute of the student. For students of unknown institutions a default set of rules will be used.

Improvement of contents of repository

The evaluation made clear that one of the most important tasks is to increase the number of exercises. For these new exercises the following conditions should apply:

- The new exercises should include many small exercises that focus on one concept only.
- The feedback of all exercises should be as extensive as possible. For example, all results of all test cases should be shown at each submission. So when one test fails, all other tests should still be executed. Furthermore the reason for failing to pass a test should be explained if possible.

Extended evaluation

When all previously mentioned improvements have been implemented, the evaluation as described in section 4 will be extended and repeated, to see whether the improvements made the repository more interesting for the students, and whether they will use it as we expect. This time, apart from interviews with a number of serious students, we will also perform a survey among all students of all relevant courses, to get more information, for example why students do not use the repository.

The SERF tool is currently used in the teaching at the developing institutes. Several other institutes have expressed interest in deploying the tool. We will support and monitor such use continuously, for further improvement and development of the tool as well as for the important aim to build a substantial repository of shared exercises.

Acknowledgements

This work has been done in the context of the SERF project that has been funded by SURF in the ‘stimuleringsregeling Open en online onderwijs’ 2018. We would like to thank Stefano Schivo and Nikè van Vugt-Hage (Open Universiteit) for helping us making and reviewing exercises for the repository, and Erik Scheffers (Eindhoven University of Technology) for implementing the tool.

References

Ala-Mutka, K.M. (2005). A survey of automated assessment approaches for programming assignments. *Computer science education*, 15(2), 83–102.

Bijlsma, L., Huizing, C., Kok, A., Kuiper, R., Passier, H., Scheffers, E., Schivo, S., Vos, T. (2021). Construction of a knowledge graph for exercise selection. Technical Report OUNL-CS no. 2, Open Universiteit.

Checkstyle `checkstyle.org`

Chen, H.-M., Nguyen, B.-A., Yan, Y.-X., Dow, C.-R. (2020). Analysis of Learning Behavior in an Automated Programming Assessment Environment: A Code Quality Perspective. *IEEE access*, 8, 167341–167354.

CodingBat `codingbat.com/java`

Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O. (2010). Review of Recent Systems for Automatic Assessment of Programming Assignments. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pp. 86–93.

Keuning, H., Jeuring, J., Heeren, B. (2019). A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education*, 19(1), 1–43.

Munson, J.P., Zitovsky, J.P. (2018). Models for Early Identification of Struggling Novice Programmers. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, New York, NY, USA, pp. 699–704.

Paiva, J.C., Leal, J.P., Figueira, A. (2022). Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Transactions on Computing Education*, 22(3).

SERF Repository `serf.win.tue.nl`

Souza, D.M., Felizardo, K.R., Barbosa, E.F. (2016). A Systematic Literature Review of Assessment Tools for Programming Assignments. In: *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, pp. 147–156.

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J.K., Padua-Perez, N. (2006). Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. *SIGCSE Bull.*, 38(3), 13–17.

Spacco, J., Denny, P., Richards, B., Babcock, D., Hovemeyer, D., Moscola, J., Duvall, R. (2015). Analyzing Student Work Patterns Using Programming Exercise Data. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, New York, NY, USA, pp. 18–23.

Vihavainen, A., Luukkainen, M., Ihantola, P. (2014). Analysis of Source Code Snapshot Granularity Levels. In: *Proceedings of the 15th Annual Conference on Information Technology Education*. Association for Computing Machinery, New York, NY, USA, pp. 21–26.

Yan, Y.-X., Wu, J.-P., Nguyen, B.-A., Chen, H.-M. (2020). The Impact of Iterative Assessment System on Programming Learning Behavior. In: *Proceedings of the 2020 9th International Conference on Educational and Information Technology*. Association for Computing Machinery, New York, NY, USA, pp. 89–94.

A. Kok is assistant professor computer science at the Open Universiteit, where he is responsible for courses on object oriented programming. He studied computer science at the Delft University of Technology, The Netherlands. He received his Ph.D. in computer graphics from the same university. Before he joined the Open Universiteit, he worked on computer graphics and virtual reality as software engineer in industry, as research scientist for TNO, and as assistant professor at the Eindhoven University of Technology.

A. Bijlsma is professor emeritus of computer science at the Open Universiteit. After obtaining a Ph.D. in mathematics at University of Amsterdam in 1978, he has worked in both mathematics and computer science at the Institut des Hautes Etudes Scientifiques (Bures-sur-Yvette, France), and at universities in Eindhoven and Utrecht (Netherlands), and Cochabamba (Bolivia).

C. Huizing works at the Eindhoven University of Technology as teacher and researcher in computer science and education of computer science. His research is about formal verification of programs with tooling and about teaching of programming. His next interest is to apply program derivation and other formal techniques in the teaching of programming.

R. Kuiper worked at the Mathematical Centre (Amsterdam) and Victoria University (Manchester), and currently works at the Eindhoven University of Technology and the Open Universiteit. Research is centered around the concepts of decomposition and abstraction: for programming and specification languages, semantics, logics, proof systems and tooling. The next interest is to apply these concepts in the various stages and at various levels in programming and development. The final challenge is to teach programming and program development based on such insights: identify and articulate guiding principles, design exercises, and support the learning process with tooling.

H. Passier is assistant professor computer science at the Open Universiteit. He has experience in teaching object oriented programming, functional programming, web programming, object oriented design, and software engineering. His research interests include the use of procedural guidance in software development, testing and refactoring.

Appendix A: Interview guide

Each interview consisted of questions in five categories. Each category consisted of one main question that always was asked. Depending on the answers, one or more of the sub questions were asked to get a deeper insight into student's opinion:

Functionality SERF repository. What is your opinion on the functionality of the repository, i.e. what was good, what did put you off, what did you miss, what happened?

- Did you find the relevant exercises?
- In particular, was the search function satisfactory?
- Went things well/wrong in searching to exercises?
- Did you use the exclude option?
- Did you do the exercises in the repository or first in your own system.

Content of the repository. What is your opinion on the content of the database?

- Where the exercises useful and/or relevant?
- Did you miss exercises on a specific subject?
- Was the quality of the formulation of the exercises sufficient?

Quality of the feedback. What is your opinion on the feedback?

- Was the feedback relevant?
- Was the quality of the feedback sufficient?

Impact on study. Did the repository help you with your study?

- Would you advice it to other students?

Improvements. Do you have any advice on future improvements?