UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Hiie-Helen Raju

Lab Package: Debugging

Bachelor's Thesis (9 ECTS)

Supervisor: Prof. Dietmar Pfahl

Tartu 2018

Lab Package: Debugging

Abstract:

The goal of this thesis is to create lab materials about debugging for the course “Software Testing” (MTAT.03.159) at the University of Tartu. The thesis gives an overview of the motivation for this lab, and the created lab materials and tasks. The lab materials were applied in the 2018 spring semester. Received feedback is reported and analysed and an overview of implemented improvements is given.

Keywords:

Debugging, software testing, lab package

CERCS: P170, computer science, numerical analysis, systems, control

Praktikumimaterjal: Silumine

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärgiks on luua praktikumimaterjalid silumise kohta Tartu Ülikooli kursuse “Tarkvara testimine” (MTAT.03.159) jaoks. Töös kirjeldatakse peamisi põhjuseid silumise praktikumi loomiseks ning töö käigus loodud praktikumimaterjale ja ülesandeid. Loodud materjale rakendati 2018 kevadsemestril. Töös antakse ülevaade ja analüüs saadud tagasisidest ning selgitatakse tehtud täiendusi.

Võtmesõnad:

Silumine, tarkvara testimine, praktikumimaterjal

CERCS: P170, arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1 Introduction

Software testing is a part of the software development process. The goal of software testing is to ensure a high quality of the software. There are several different approaches and tools used in testing, which, in Tartu University are introduced to the students in the course "Software Testing" (MTAT.03.159). This thesis focuses on one testing method, debugging.

The course "Software Testing" is mainly targeted at second year students of the Computer Science Bachelor's degree curriculum. The course consists of seven labs, each of which introduces a new approach or tool in software testing which are used to find or fix faults in the software. In previous years, debugging has not been included in this course and mostly the students have been expected to learn this skill themselves.

Debugging is a testing process with the main aim of localising program failures and fixing found faults. As secondary aim, debugging can be used to familiarise oneself with the source code of a program. Several heuristics and debugger tools are used for debugging. The term heuristic refers to techniques for problem-solving and solution discovering [1]. Debugging is an important part of software testing and in order to provide the students with necessary skills to use this, in the spring of 2018, it was added to the "Software Testing" course lab curriculum.

The aim of the thesis is to create lab materials and homework tasks to introduce debugging to the students. To do this, two programs were created: one to implement the heap sorting algorithm[1] and one to implement the genetic algorithm[2] to solve the problem of placing 8 queens on a chessboard with no threats [2]. The students are expected to debug these programs. The students are introduced to different debugging heuristics and IntelliJ and Eclipse IDEs (Integrated Development Environment) built-in debuggers. In the scope of the thesis, the lab materials were applied, and feedback was collected from students and teacher's assistants for further improvements.

[1] https://www.geeksforgeeks.org/heap-sort/
[2] https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/

The substantive part of this thesis consists of four main chapters. In the second chapter of the thesis, the background of software testing, debugging, and the "Software Testing" course is introduced. In the third chapter, the structure and the developing process of the lab materials are introduced. The fourth chapter consists of an overview of how the lab was applied in the 2018 spring semester. The fifth chapter analyses given feedback and explains applied improvements. The thesis ends with a summary stating the main results and outcomes of the created materials.

2 Background and Existing Materials

This section provides an overview of the software testing course that the lab is a part of and the topic of the lab (debugging).

2.1 The course "Software Testing"

"Software Testing" (MTAT.03.159) at the University of Tartu is a 3ECTS course taught in the spring semester at Tartu University. It is targeted at second year students of the "Computer Science" Bachelor's programme and the students of the "Conversion Master in IT" Master's programme [3]. The course outline states: "The course addresses the essential concepts of software quality control and testing and introduces various testing strategies and types of testing." [3] In the 2017/2018 spring semester the course consisted of 7 lectures and 7 labs. The topics of the labs were as follows:

1. Debugging
2. Black-Box & White-box Testing
3. Combinatorial Testing
4. Mutation Testing
5. Automated Web-Application Testing
6. Static Code Analysis
7. Document Inspection and Defect Prediction

The goal of this thesis is to provide lab and homework materials for lab 1 (Debugging).

2.2 Debugging

One possibility for ensuring a high software quality is to use debugging. Debugging is a part of the software testing process. The term comes from the software engineering term "bug" which is any problem "causing a program to crash or produce invalid output" [4]. Debugging is the process of localizing and removing these problems. It usually follows active testing where a problem has been noted. Debugging involves analysing the problem, locating the source of the problem and fixing the problem in the software program [5]. Debugging aims to localise and remove faults in the code [6, p. 41]. However, it is also used to understand the code. Problem-solving in software development is a wide yet integral component. Debugging is an improved version of static code analysis by making it faster, simpler and more efficient. In static code analysis, each line of the code is viewed and analysed one-by-

one without executing the program itself. In debugging, states of the program are viewed and analysed while the program is being executed.

Debugging must be differentiated from using a debugger tool. Debugging itself is a wider term and refers to the entire problem-solving process. Using debugger tools is one, and usually the biggest, component of the debugging process.

The main advantages of debugging can be seen in the following situations:

1. In combination with Unit-testing:
    - Unit tests let the tester know which methods of functions are not working properly.
    - During debugging the tester will analyse the information from unit tests and localise the problem within the method which caused the unit test to fail.
2. Large-scale systems
    - Large-scale systems are very difficult to handle by just reading the code exhaustively and/or printing out program states.
    - Debugging helps the tester navigate through the code to only the sections which use the input.
    - Debugging helps the tester or the new developer get to know any system they have not seen before.
3. Recursive functions
    - Debugging helps to see the values and states of the variables in each run so the tester can see where in the recursion the program currently is and what might be wrong there.
4. The tester or developer is new to the system.
    - Debugging can simplify and quicken the learning and familiarization process. Debugging can be used to gain information about reused methods that are otherwise difficult to understand [6, p. 434].
5. Systems that are not properly documented, especially if they are in a language that is not easy to read

- If there is no information about which function is supposed to do what debugging is a quick way to learn this.

6. Something has crashed

   - When something crashes in the program, but no valid and relevant error message is displayed, it is possible to try going over it with the debugger tool to see where the program fails and why.

This means that debugging offers a wide range of possibilities. The most commonly used part of debugging is using debugger tools. Although many different tools exist to help developers and testers solve problems, it is also crucial to have the proper mind-set when using these. An important part of debugging is using different testing heuristics. Heuristics can include basic common sense used in everyday life or learnt knowledge such as rules of thumb [7].

There are several publicly available tools for debugging. These can include separate tools or built-in tools for IDEs (Integrated Development Environments). WinDbg[3], Android Studio Debugger[4], built-in debuggers of Eclipse[5] and IntelliJ[6] to mention a few. The Eclipse and IntelliJ debugger tools are used in the scope of this thesis.

The developed lab materials aim to help the students learn to use debugger tools, some debugging heuristics and understand their thought process while debugging software.

[3] https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/
[4] https://developer.android.com/studio/debug/index.html
[5] https://www.eclipse.org/
[6] https://www.jetbrains.com/idea/

3 Lab Design

This section provides an overview of the structure of the lab and the materials provided for the students and lab assistants.

Before the execution of the lab, four third-year Computer Science students were asked to try out parts of the lab. Two of the students tested Task 1 and two tested Task 2. This was done to evaluate the difficulty and time consumption of the tasks. These students were timed while solving the tasks. The two, who solved Task 1, took about 90 min without reporting the findings and the two who solved Task 2, took about 120 min without reporting the findings. Timing showed that the time consumption of the tasks corresponded to the desired amount. The four student testers stated that the tasks were moderately difficult and interesting to solve.

3.1 Schedule

The tasks must be completed alone or in 2-person teams.

Each student is expected to spend 6 student hours (270 min) on each lab. Of these 6 hours, 2 (90 min) will be spent in the lab and 4 (180 min) will be spent on completing the homework assignment.

To test out the time consumption and clarity of the lab, a programmer was asked to complete the lab. The programmer is a personal acquaintance of the author and volunteered to try out the lab. The programmer has more experience in coding than the students taking the course are expected to have, however, he has very little experience in Java[7]. The tasks took him about 9 person (academic) hours. The time consumption matches the desired amount as the programmer has more experience and the task is meant to take a pair of students about 12 academic hours. They stated that the tasks were interesting to solve and that they learned some things they did not yet know about debugging.

3.2 Tasks

The lab consists of two tasks. For the purpose of this lab, two Java[7] projects were created: a heap sorting algorithm for Task 1 and a genetic algorithm for Task 2. The students are expected to debug both of the created projects and report their resolution process and findings. Both tasks and algorithms are explained further in 3.2.1 and 3.2.2.

[7] https://www.java.com/en/

The reporting structure is the same for both tasks. A completed task includes a list of all the faults introduced to the program, their relevant fixes, and a filled out resolution table for each bug. All of these described parts need to be organized under the relevant issue reports which are provided for the students.

The lab has two aims: to introduce the Eclipse[5] and IntelliJ[6] debugger tools to the students and to teach some debugging heuristics, thought processes and techniques. It is important to activate problem-solving skills, not just teach the handling of tools [8].

In order to be able to grade the second goal, a resolution table structure was created. This structure is further explained in 3.2.1.

To aid the learning of debugging heuristics, a table (see Table 1) was composed consisting of examples of heuristic approaches. Table 1 was created based on the author's personal experience and the features of the debugger tools used in the labs. The possible heuristic approaches in Table 1 are not conclusive and the students are encouraged to think of additional heuristics.

Table 1: Debugging heuristics.

No.	Remark	Why is it useful
1	Whenever you reach a new function call, step into it.	If you have reached a function call and don't know the source of the failure yet, it is likely that the problem is deeper in the code. This means you need to check whether the body for this function works as it should.
2	Whenever you reach a new function call that takes a parameter, check if the parameter given is logical.	The cause for a failure may just be that all the functions work properly, just that one of them is called with incorrect parameter value(s).
3	Check the documentation of the code for hints.	The comments in the code from the author can be very useful in determining what the code is supposed to do and how.
4	Change the input code from main method	Reducing input can make it easier to follow the run of the code or even cause an error that makes it easier to localise the problem. Changing it into something that you know the correct output for can also make it easier to make sure the code does what it's supposed to.

5	If the data is in a structure that is difficult to visualize, construct it on a paper while debugging	If the data is a tree, a matrix or some other structure that is difficult to keep visualizing all the time, it can be useful to draw out (with pen and paper) the state you have in your debugger while debugging (using the variables view). That makes it easier to be sure that you did not miss anything from the variables and states you were viewing.
6	Change the value of a variable in the debugger. ("Change value" command)	This can give you a more varied overview of the ways the code acts with different values without having to restart the debugger from the start with a new original call parameter.
7	Run only part of the program at a time, if possible.	If you run the full program you will have a lot of code to debug through, which does not save you much time when compared to exhaustive solving. Therefore find a way to only run some of the program and check that before continuing on with the rest.
8	If you already know something is going to work correctly in a loop, don't step through every iteration again.	You can skip for loops to a specific run of the cycle. This allows you to skip the runs of the loop that you already know will be correct and immediately get the run where you think a problem might be. To do this, set a breakpoint on the for loop line (You can do this even when the debugger is already running). In IntelliJ, right click on the breakpoint, write the iteration number in the Condition field (e.g. i==3), and then run the debugger. In Eclipse, right click on the breakpoint, choose Breakpoint properties, tick Conditional and write the condition in the field below (e.g. i==3).

In order to teach the debugging thought process, the students are provided a figure (Figure 1) depicting a thought process pattern they should follow when debugging. Figure 1 consists of an entry point titled "Intro", a middle section with titles "Analysing info" and "Deciding point", through which the students should loop, and an exit point titled "Final conclusion". Each division of Figure 1 contains questions that the students should ask and answer during their debugging process. By answering these questions, the students can analyse their own thoughts and the reasoning behind their actions. This self-analysis helps to root the useful steps and thoughts, and to cut out immaterial ones.

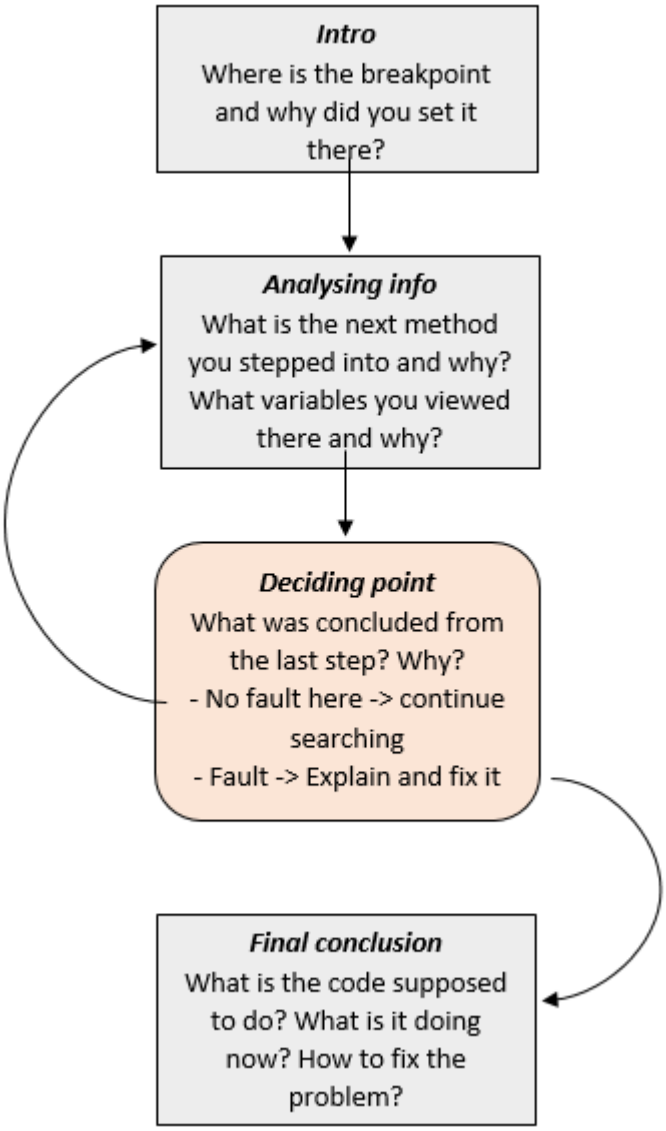

Figure 1. Thought process for localizing a bug.

3.2.1 Task 1

The first task is to debug a heap sorting algorithm program (see Appendix A). The aim of this task is to learn the tools and possibilities of debugging with an algorithm that the students are likely to be familiar with. The algorithm uses recursion, which is difficult to evaluate with static code analysis. This means that the students can see the benefits of debugging while completing the task.

Binary heap is a data structure where elements are kept as a binary tree. In a maximum binary heap (which the created program is), the root element has the highest value. Each element has a maximum of two child elements. On each lower level of the binary tree, the values are also lower than (or equal to) their parent. The program created for the lab only accepts integers valued from zero to the Java maximum integer value ($2^{31} - 1$).

In order for the students to be able to debug issues in this program, six bugs were introduced. These six bugs cause three different issues in the program which are presented to the students as issue reports.

The first bug is to be solved in-lab with the lab instructor and the findings are to be reported in a resolution table. As mentioned above, each bug resolution process should correspond to one filled out table. The sample solution to the first bug is seen in Table 2.

Table 2. Heap bug no 1, Issue 1, filled out.

Method	Variables	*Intro* and *Analysing info*	*Deciding point* and *Final conclusion*	Heuristics used
Main	Heap heap = new Heap(heapList);	Since the elements are all there in the first list, this is the first point where something is wrong.	This was a good entry point to the main program class as this is the highest level and allows me to go further into the program.	1,2,3,5
heapify()	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]; lastIndex – 12; parentIndex – 6	Need to check if the indexes that are used to heapify the list are calculated correctly.	Since I know that in heapsort the parent index should be 5 here, but it is 6, there must be something wrong in calculating this index.	
elemParentIndex (int index)	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]; index – 12	I found from last method, this one returns a wrong index so I need to see what is happening inside it.	As I know that in heapsort parent indexes are calculated with (currentIndex-1)/2 rounded down. I can see that is not the case here. Change *return index/2* to *return (index-1)/2*	

Table 2 is filled out as it is provided for the lab instructors as the sample solution. The contents of the table correspond to Figure 1 and Table 1. The first and second columns contain data about where in the program the student was at each point in the debugging process and what information they were able to view. The programmer who completed the entire lab mentioned that the connection between the thought process figure (Figure 1) and the resolution table could be more clear and therefore the titles were connected. The third column titled "*Intro* and *Analysing info*" and the fourth column titled "*Deciding point* and *Final conclusion*" correspond to the divisions in Figure 1 with the same titles. The fifth

column titled “Heuristics used” should contain references to the heuristics table (Table 1) and any other heuristics that the students have used.

The lab instructors are expected to provide the main introduction to filling out the resolution table. The students are provided with an empty table structure and they should fill it out in-lab with the instructors. The instructors should make sure that the table is clear for the students. The structure of this table was chosen from three options that the author created. The first option, that was not chosen, was much longer and contained a more detailed overview of the debugging process. The benefit of it would be that the students would have to show their process in more detail and therefore they would have to analyse it more thoroughly. The downside would be more work for the teachers’ assistants who would be grading it. The second option, that was not chosen, contained less data and would have been easier to grade, however it would have made it more confusing to understand the debugging process and the students’ reasoning behind it.

3.2.2 Task 2

The second task is to debug a genetic algorithm program. The reporting structure for this task is the same as it is for Task 1. The purpose of the program is to find a state of a chessboard where there are eight queens on the board, there are no other chess pieces and none of the queens threaten any of the others. Genetic algorithms mimic the evolutionary process of mating fittest specimens and applying random mutation [9]. The general flow of the algorithm can be seen in Figure 2.

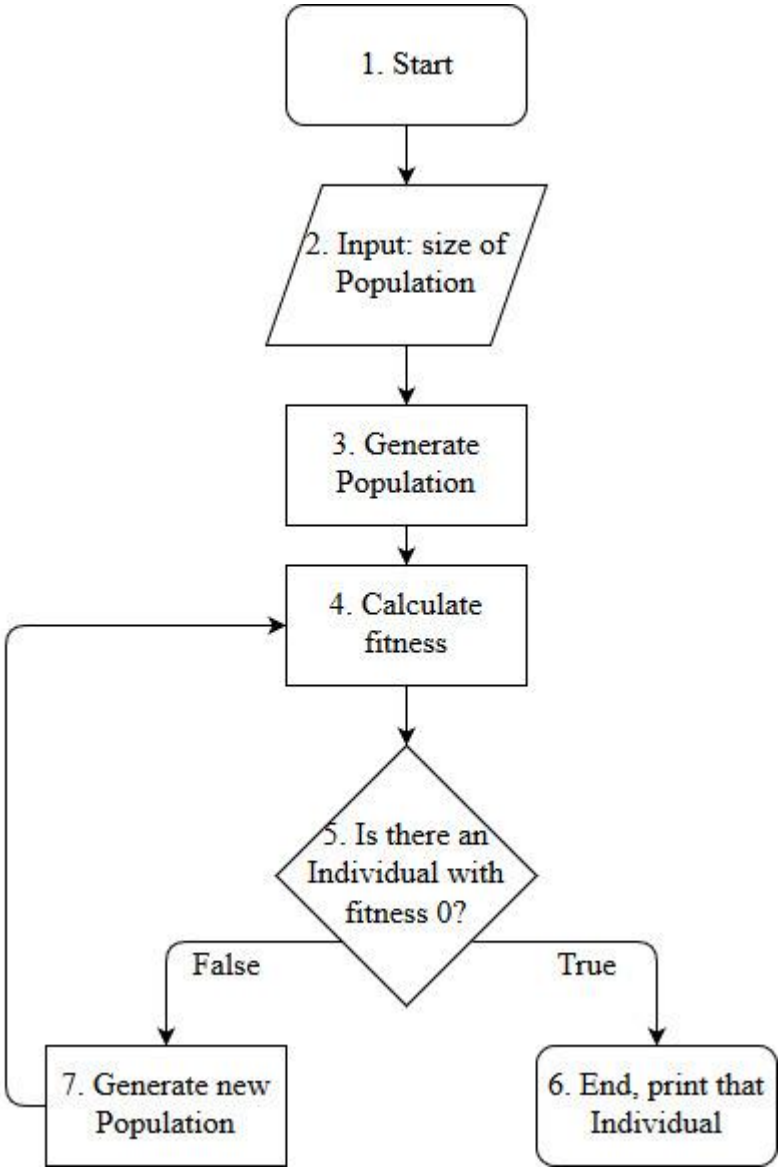

Figure 2: Flow of the Task 2 algorithm

This program is more complex than the program for Task 1 and it is most likely unknown to the students. The lab tries to capture the complexity of real issues that can occur in software development and having to familiarise and debug an unknown algorithm does that. However, as this may be difficult to debug, helpful methods and hints are provided for the students. The main class of the program contains methods that help the students run each part of the algorithm separately and to see the outcome of these parts. Some hints are provided in the lab instructions file (see Appendix A1.1) and later a description of the flow of the program was added to the instructions (see 5.4). The programmer who solved the full lab before execution said that Task 2 was a bit difficult, however, they managed to solve it nonetheless and found it interesting.

In order for the students to be able to debug issues in this program, four bugs were introduced. Three of these bugs correspond to one issue, the fourth bug corresponds to a different issue that only appears once the first three have been fixed. This task tries to introduce a more advanced approach to debugging as the algorithm uses randomness in its data and the effectiveness is related to performance.

3.3 Materials

The lab materials created for the purpose of this thesis can be divided into two sets, those for the students and those for the lab instructors.

An overview of the materials and their division can be seen in Figure 3. All these materials can be found in Appendix A.

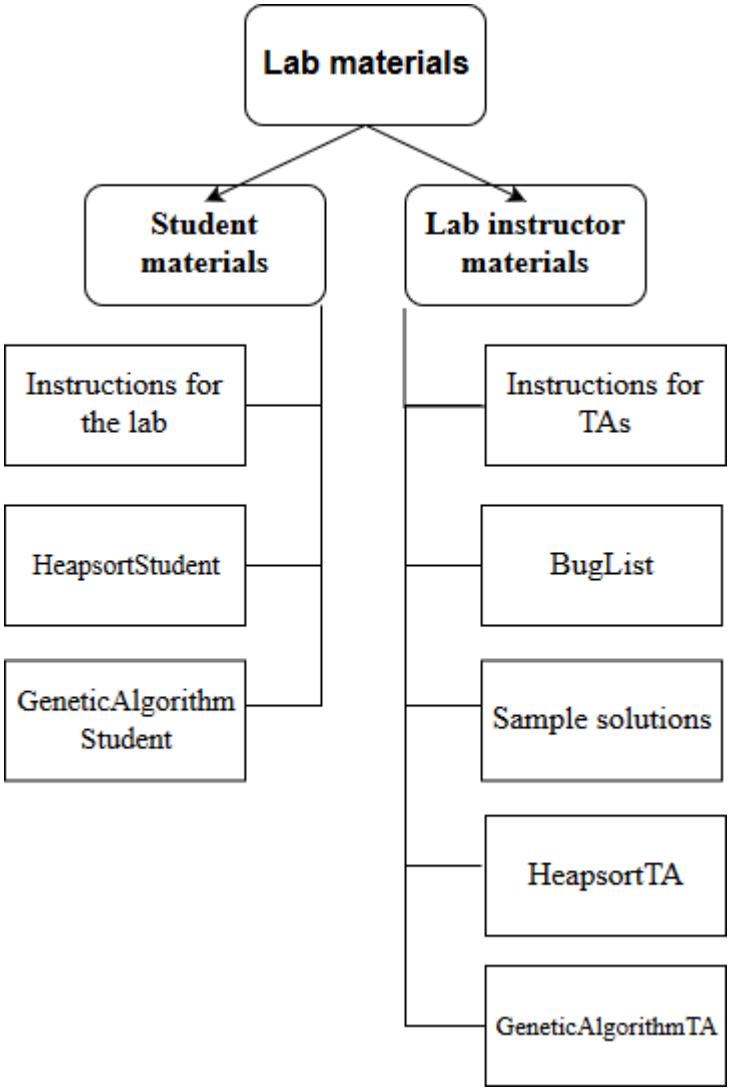

Figure 3. Structure of the lab materials

3.3.1 Student Materials

"Instructions for the lab" for the students is a guide for the lab and homework tasks provided as a PDF-document. It comprises of four sections:

1. An overview and an introduction to debugging, debugger tools, debugging heuristics, the systems under test and the lab report structure
2. Tasks
   2.1. Debugging the heapsort algorithm
   2.2. Debugging the genetic algorithm
3. Grading Scheme
4. Links and Appendixes

"Instructions for the lab", in addition to explaining the tasks, aims to make it easier for the students to understand what is expected of them and to reduce the time spent on learning how the systems are supposed to work.

The "HeapsortStudent" project is meant to be debugged under Task 1. It is as Java[7] project that implements the heapsort algorithm. The code is commented providing the students some information about what different methods in the program do.

The "GeneticAlgorithmStudent" project is meant to be debugged under Task 2. It is also a Java[7] project. It implements the genetic algorithm to find a solution to the eight queens' problem. The program tries to find a way to position eight chess queens on a chessboard (no other chess pieces) so that none threaten any of the others. The code for this program is also commented the same way as the heapsort algorithm program.

3.3.2 Lab Instructor Materials

"Instructions for TAs" is a PDF-document that consists of the following information:

1. What the lab instructors should do in-lab. The file gives steps what should be covered and a step-by step guide on how to show the solving of the first bug while filling out the resolution table (see Table 2).
2. What they should note when checking the tasks. Some possible situations that may arise are stated with proposed solutions.
3. How to provide points for the tasks. The point distribution is provided with relevant information.

"Example solutions" is a PDF-document consisting of filled out resolution tables and lists of faulty lines of code with their relevant fixes.

“Bug list” is a PDF-document that lists all the bugs in both projects with their line numbers in the student projects, the faulty lines and the correct lines.

“HeapsortTA” and “GeneticAlgorithmTA” are the same programs as those provided for the students, although they differ by a few details. By default, there are no bugs in these project. The projects have the faulty lines of code as comments, and it is possible to comment them in to check the issues. These projects also contain more comments explaining the program than the students’ versions in order to make it easier for the lab instructors.

3.4 Grading

The students can get a maximum of 10 points for each lab in the “Software Testing” course. For this lab the points are distributed as follows:

- 1 point for attending the lab session.
- 1 point for each fully completed bug solution and resolution table in Task 1, up to 5 points.
- 1 point for each fully completed bug solution and resolution table in Task 2, up to 4 points.

4 Lab Execution

The lab materials created were used in Lab 1 of the "Software Testing" course held in spring 2018. There were 95 students registered for the course and they were divided into 5 lab groups. Each lab group had one lab instructor to guide them.

It was expected of the students to be already familiar with sorting algorithms and the heap data structure. The Computer Science curriculum at Tartu University teaches the course "Algorithms and Data Structures" (LTAT.03.005) which includes sorting algorithms, during the semester preceding the one "Software Testing" course is taught. They are also expected to have some knowledge of the debugger tools as they are briefly introduced to the tools in the course "Software Engineering" (LTAT.05.003) which is a compulsory prerequisite subject for the "Software Testing" course. No other preliminary work is required from the students for the lab.

One lab group session was observed to collect data on the students' progress and the time management of the lab session. In the observed lab, everything seemed to go according to the plan. The instructor introduced the lab and the first resolution table was filled out in cooperation with the students. The setup of the projects did not cause any problems either.

All in all, the average amount of points for the lab was around 9. One point was granted for free due to an issue reporting problem discovered by one of the lab instructors in Task 2. This problem and the solution to it is explained in 5.3 and 5.4.

5 Feedback

This section gives an overview of received and acquired feedback, implemented improvements based on the feedback and suggestions for future improvements. Improvements from all the feedback and the fixes to the found problems were implemented to the lab package and are explained 5.4.

The received feedback is divided into two sections:

- Collected feedback from the students who took the lab.
- Personal feedback from the lab instructors who gave the lab.

5.1 Feedback from Students

Feedback was collected from students using a web-based questionnaire on the web application SurveyMonkey[8]. 7 of the students provided feedback. The questionnaire was based on a Likert Scale [10] where a person can express how much they agree with a statement based on a rating scale of five options. The statements given to the students were:

1. The goals of the lab were clearly defined and communicated.
2. The tasks of the lab were clearly defined and communicated.
3. The tasks of the lab were interesting to solve.
4. The maximum binary heap task was appropriate and useful.
5. The genetic algorithm task was appropriate and useful.
6. If I have a choice, I will use a debugger again.
7. If I have a choice, I will use debugging heuristics again.
8. The support received from the lab instructors was appropriate.
9. The grading scheme was transparent and appropriate.
10. Overall the lab was useful in the context of this course.

In addition, the students were asked two more questions in free form. These were “Please, in your own words, say what you liked/did not like about the lab!” and “Do you have anything else to add?”

5.2 Feedback from Lab Supervisors

The lab instructors who executed the lab provided some written and verbal feedback on the lab. They provided information about what they thought to be good about the lab and what

[8] https://www.surveymonkey.com/

could be improved. The discussions with the lab instructors occurred after the lab and helped to analyse whether the tasks were clear, the difficulty level suitable for the students, and what could be done to improve the lab.

5.3 Analysis

Since only 7 students gave feedback out of 95 registered to the course, the sample is not big enough to make a conclusive analysis. Nevertheless, some analysis was done on the feedback of the 7 students. In general, the feedback was above average.

From the qualitative questions in the questionnaire, it was mentioned that mostly everything was fine. The students pointed out that that Task 1 was interesting and useful. They found it good that they had learned the heap sorting algorithm before the lab. For Task 2 they said that it was a bit too difficult as they did not know how the algorithm was supposed to work.

Some students said that the instructions were thorough while some said it was tiresome to go back to the thought process figure (Figure 1) and heuristics table (Table 1). It was mentioned that it was really good that the students can practice reading code as they do not usually have a chance to work with real code.

Results of the quantitative part of the online questionnaire is summarized in Table 3. As mentioned before, such a small sample cannot give a conclusive overview of the lab. Based on this sample, there were problems with Task 2 where the algorithm remained somewhat unclear. They understood the goals of the lab and plan to use debugger tools and heuristics in the future. They seemed to be happy with the grading scheme and the lab instructors. It can be noted that there was some confusion about the tasks' descriptions.

Table 3: Questionnaire feedback

	Disagree entirely	Disagree somewhat	So-so	Agree somewhat	Agree entirely	Total
The goals of the lab were clearly defined and communicated	0.00% 0	0.00% 0	14.29% 1	42.86% 3	42.86% 3	7
The tasks of the lab were clearly defined and communicated	0.00% 0	14.29% 1	28.57% 2	28.57% 2	28.57% 2	7

The tasks of the lab were interesting to solve	14.29% 1	14.29% 1	0.00% 0	57.14% 4	14.29% 1	7
The maximum binary heap task was appropriate and useful	0.00% 0	14.29% 1	0.00% 0	42.86% 3	42.86% 3	7
The genetic algorithm task was appropriate and useful	14.29% 1	28.57% 2	14.29% 1	28.57% 2	14.29% 1	7
If I have a choice I will use a debugger again	0.00% 0	0.00% 0	0.00% 0	57.14% 4	42.86% 3	7
If I have a choice I will use debugging heuristics again	0.00% 0	14.29% 1	14.29% 1	57.14% 4	14.29% 1	7
The support received from the lab instructors was appropriate	0.00% 0	14.29% 1	42.86% 3	28.57% 2	14.29% 1	7
The grading scheme was transparent and appropriate	0.00% 0	0.00% 0	14.29% 1	71.43% 5	14.29% 1	7

The lab instructors said that the lab was interesting and varied. They said that it was good that the lab focused on the thought processes and heuristics in addition to asking for the fixes to faults in the code.

The lab instructors mentioned that the lab instructions could be more specific in some parts and that the resolution table and reporting structure could be explained more thoroughly. They expressed that their understanding of how the first resolution table (see Table 2) should be filled was not as complete as they would have liked. They also agreed with the students, that Task 2 was a bit too difficult as the algorithm was new to the students.

An issue reporting failure was found by the lab instructors in Task 2. The fault in the code did not correspond to the respective issue described in the report and the issue from the fault was not visible. Due to this, the students were given a free point for that specific fault in the code.

There was also a small problem with an issue report for Task 1. The input value used in the code masks the issue and therefore the issue was not reproducible without changing the input. The students are encouraged to try different input values for any part of the tasks in the lab instructions. Some students had found the fault by using a different input value and fixed it. Others were given a free point for this fault in the code, if they said that the actual output matched the expected output with the given input. If they did not say anything, they did not get a point.

5.4 Improvements

This section provides an overview of the improvements and fixes made to the lab package based on the feedback analysis. It also gives suggestions on what improvements could be implemented in the future.

5.4.1 Implemented improvements

One possible improvement mentioned by both students and lab instructors was that the instructions' documents could be more specific. The documents were supplemented with clarifications for the tasks' and reporting descriptions. The connections between the thought process figure (Figure 2) and the resolution table (Table 2 for the first fault) structure were clarified more in the students' lab instruction document.

A step-by-step structure for filling out the first resolution table (Table 2) was added to the "Instructions for TAs" document to make it easier for the lab instructors to explain the reporting to the students.

The minor issue reporting problem in Task 1 was fixed by changing the input value in the code and the relevant outputs in the lab instructions so that the failure was viewable.

The issue reporting problem in Task 2 was fixed in a more extensive way. As Task 2 was considered to be a bit too difficult as well, the faults in the code and issue reports were rethought. As a result, there were still four faults introduced to the code, however, one of them was changed and corresponding issue reports were constructed. It was tested several times, so that all the faults were now visible in the output and they corresponded to the expected and actual outputs in the issue reports. The students are also told, as a hint, how many faults in the code each issue report corresponds to.

In addition to fixing the issue reports, it was concluded that some additional information should be provided for the students to simplify learning the genetic algorithm program structure. To do this, the flow of the program was documented and illustrated with figures (e.g. Figure 2), and added to the students' lab instructions. The description of the flow also includes a non-conclusive list of questions which the students could ask while examining the code.

5.4.2 Future improvements

Other improvements could be implemented to the lab. For example, if Task 2 turns out to be too easy after the added information and hints, the amount of help given to the students could be reduced.

In the feedback, it was mentioned that the reporting table structure was a bit confusing. It was clarified and improved but could still be further modified. For example, the table could be replaced with some sort of diagram or an interactive table to be filled out online.

Other programs could be implemented and given to the students to debug, using the same reporting structure. For example, K-means clustering could be implemented and faults introduced. K-means clustering algorithm is versatile and can be applied in different ways and, therefore, can be used to teach different aspects of debugging [11]. This algorithm is complex, but understandable and could be used to replace the Task 2 algorithm.

6 Conclusions

The aim of this thesis is to create lab (and homework) materials for the “Software Testing” (MTAT.03.159) course at the University of Tartu. In the scope of this thesis, lab materials about debugging were created and used in the 2018 spring semester for the course. Debugging was previously not taught in this course.

To ensure that the lab is appropriate and useful, feedback was collected from the students, lab instructors and other volunteers. The feedback was mostly positive, although some improvements were suggested and a couple of small problems became visible. These problems were fixed and the suggested improvements applied. The tasks were deemed somewhat difficult but interesting to solve. All in all, the lab materials were satisfactory and after the introduced improvements, they can be applied in the future in the “Software Testing” course.

References

[1] K. N. Johnson, “Software Testing Heuristics & Mnemonics,” 2012. [Online]. Available: http://karennicolejohnson.com/wp-content/uploads/2012/11/KNJohnson-2012-heuristics-mnemonics.pdf . [Accessed 04 04 2018].

[2] V. Mallawaarachchi, “Introduction to Genetic Algorithms—Including Example Code,” *Towards Data Science,* 2017.

[3] “Course outline (2018),” [Online]. Available: https://www.is.ut.ee/rwservlet?oa_ainekava_info_ik.rdf+1342573+PDF+84492113649293868485+application/pdf. [Accessed 04 04 2018].

[4] “Techopedia,” [Online]. Available: https://www.techopedia.com/definition/24864/software-bug-. [Accessed 08 05 2018].

[5] M. Rouse, “Debugging,” 10 2016. [Online]. Available: https://searchsoftwarequality.techtarget.com/definition/debugging. [Accessed 04 04 2018].

[6] I. Sommerville, Software Engineering, Pearson, 2011.

[7] W. Herbert, “Heuristics Revealed,” 10 2010. [Online]. Available: https://www.psychologicalscience.org/observer/heuristics-revealed. [Accessed 09 04 2018].

[8] D. H. Odell, “The Debugging Mind-Set,” *Communications of the ACM,* vol. 60, no. 6, pp. 40-45, 2017.

[9] S. Jain, “Introduction to Genetic Algorithm & their application in data science,” *Analytics Vidhya,* 2017.

[10] S. Macleod, “Likert Scale,” *Simply Psychology,* 2008.

[11] A. Trevino, “Introduction to K-means Clustering,” *Data Science,* 2016.

Appendix

A. Lab materials

I Student Materials

A1.1 “Instructions for the lab”, pdf-document – https://courses.cs.ut.ee/MTAT.03.159/2018_spring/uploads/Main/SWT2018L1Inst.pdf

A1.2 “HeapsortStudent”, zip-file – https://courses.cs.ut.ee/2018/SWT2018/spring/uploads/Main/SWT2018L1HeapsortStudent.zip

A1.3 “GeneticAlgorithmStudent”, zip-file – https://courses.cs.ut.ee/2018/SWT2018/spring/uploads/Main/SWT2018L1GeneticAlgorithmStudent.zip

II Lab Instructor Materials

A1.4 “Instructions for TAs”, pdf-document

A1.5 “Bug List”, pdf-document

A1.6 “Sample Solutions”, pdf-document

A1.7 “HeapsortTA”, zip-file

A1.8 “GeneticAlgorithmTA”, zip-file

For confidentiality reasons, lab instructor materials are not made available in the thesis but will be made available on request.

B. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Hiie-Helen Raju**,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Lab Package: Debugging,

(title of thesis)

supervised by Dietmar Pfahl,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **11.05.2018**