education sciences

MDPI

Article

Assessment Automation of Complex Student Programming Assignments

Matija Novak * and Dragutin Kermek

Faculty of Organization and Informatics, University of Zagreb, Pavlinska 2, 42000 Varaždin, Croatia; dragutin.kermek@foi.unizg.hr
* Correspondence: matija.novak@foi.unizg.hr

Abstract: Grading student programming assignments is not an easy task. This task is even more challenging when talking about complex programming assignments at university graduate level. By complex assignments, we mean assignments where students have to program a complete application from scratch. For example, building a complete web application with a client and server side, whereby the application uses multiple threads that gather data from some external service (like the REST service, IoT sensors, etc.), processes these data and store them in some storage (e.g., a database), implements a custom protocol over a socket or something similar, implements their own REST/SOAP/GraphQL service, then sends or receives JMS/MQTT/WebSocket messages, etc. Such assignments give students an inside view of building real Internet applications. On the other hand, assignments like these take a long time to be tested and graded manually, e.g., up to 1 h per student. To speed up the assessment process, there are different automation possibilities that can check for the correctness of some application parts without endangering the grading quality. In this study, different possibilities of automation are described that have been improved over several years. This process takes advantage of unit testing, bash scripting, and other methods. The main goal of this study is to define an assessment process that can be used to grade complex programming assignments, with concrete examples of what and how to automate. This process involves assignment preparation for automation, plagiarism (i.e., better said similarity) detection, performing an automatic check of the correctness of each programming assignment, conducting an analysis of the obtained data, the awarding of points (grading) for each programming assignment, and other such activities. We also discuss what the downsides of automation are and why it is not possible to completely automate the grading process.

Keywords: automation; grading; assessment; programming; education; student

Citation: Novak, M.; Kermek, D. Assessment Automation of Complex Student Programming Assignments. *Educ. Sci.* **2024**, *14*, 54. https://doi.org/10.3390/educsci14010054

Academic Editors: Mike Joy and Peter Williams

Received: 29 September 2023
Revised: 10 December 2023
Accepted: 11 December 2023
Published: 1 January 2024

1. Introduction

Assessment is an important part of a teacher's job that must be done correctly and on time. However, grading student assignments, especially those of a complex nature, presents a formidable challenge for educators. This is even more noticeable when the number of students increases to 50 students or more and teachers want to publish grading results within a reasonable time (i.e., not later than two weeks). In this work, we focus on the assessment of student programming assignments, which can be divided into basic (introductory) assignments usually completed by freshmen in introductory programming courses and advanced (complex) assignments completed by more advanced students in the later years of their information technology (IT) study program. This study focuses on the assessment of advanced programming assignments at university undergraduate and graduate levels.

Advanced assignments require the students to architect entire applications from the ground up. To be more precise, in the context of this study, this means to program complete Internet/web applications with client and server components. These applications can

include the following: implementations of multi-thread logic, building custom socket-based protocols, reading and storing data within designated repositories, facilitating data collection from diverse external sources, implementing new web services, bidirectional messaging, deployment on different servers, creating and running containers, etc.

Such complex assignments teach the students the process of developing real-world web applications, thereby providing invaluable insights into the application development life cycle. Grading such assignments is a substantial time investment, which is required for manual assessment, for the teacher. It is not unusual that grading such assignments manually takes up to an hour per student submission. In response to this challenge, various possibilities of automation have emerged to speed up the assessment process while keeping the integrity of grading standards. This study describes a process with automation techniques that was refined over multiple years. The implementation of this process draws upon methods such as unit testing, bash scripting, using specific software like Apache JMeter, and other such approaches. This process can use pre-built tools (like AutoGrader [1]) for assessment, but the use cases described in this paper show that such tools were not used since they were assumed to be unnecessary.

The main objective of this study is to *define* an assessment process that is tailored for the evaluation of complex programming assignments using custom built scripts with unit tests and specific software. This study aims to provide concrete examples of which aspects of assessment can be effectively automated and to define the possible methods for this automation. The proposed process encompasses several stages, including pre-automation assignment preparation, plagiarism (similarity) detection, performing an automatic check of the correctness of each programming assignment, conducting an analysis of the obtained data, the awarding of points (grading) for each programming assignment, and other such activities. With this process, *we want* to emphasize that assessment and assessment automation entails more than just building the tool.

This paper also discusses the downsides surrounding automation, including acknowledging its limitations, and outlining reasons why a transition to fully automated grading is not recommended. By strategically placed partial automation, educators can streamline the assessment process, thus ensuring a balance between efficiency and evaluation accuracy. The insights presented in this paper aim to equip educators with a comprehensive toolkit for handling complex programming assignments.

Section 2 describes the related work in the field of assessment automation. Section 3 presents the methodology with an overview of the proposed process. Section 4 describes each phase of the process in detail with special focus on the automation phase for which different cases are presented. Discussion and answers to the research questions are provided in Section 5, and, finally, Section 6 concludes the paper.

2. Related Works

The automation of programming assignment assessments is not a new concept, as shown in [2]. With the increasing number of massive open online courses (MOOCs), the research into assessment automation has become even more interesting. The research has not been limited to functional testing [3]; it has also focused on providing automated personalized feedback, as in [4]. If readers are interested in feedback automation, a good start would be [5]'s survey.

Researchers have developed many pre-built tools to automate the assessment of programming assignments. For reference, read the survey of [6,7] for an overview of the tools developed between 2006–2010. Another more recent survey [3] was published in 2016 and includes a nice comparison of 30 tools in a detailed feature table. Beyond the tools mentioned in such surveys, there are other articles where a new tool was developed. The most recent systematic review on the topic "Automated Grading and Feedback Tools for Programming Education" can be found in [8].

For example, one study that dealt with the assessment automation of C++ programs with different levels of complexity using static code analysis was presented in [9]. An

assessment source-code library was, at the moment of writing the paper, available for free (https://ucase.uca.es/cac/, accessed on 20 September 2023). Another interesting paper was [10], where the authors built a tool called the Flexible Dynamic Analyzer (FDA), which uses semantic analysis techniques to assess results. Similarly, in [1], a tool called AutoGrader was built that relies on program semantics. AutoGrader automatically decides the correctness of student programming assignments according to a reference implementation. Another tool, called DGRADER [11], uses a complex multi file program analysis that can be an advantage to handling complex programming and scaled projects that require more than one file for the program.

Although many tools are mentioned in articles, many of them are not available to the public. Note that finding the tools is not a problem. For example, on GitHub one can find many tools. The problem is that there are similar names for various tools, and it is not always clear which is the one the user is searching for. For example, with AutoGrader, there is one from coursera (https://github.com/coursera/coursera_autograder, accessed on 20 September 2023) and there is one from University of Michigan (https://github.com/eecs-autograder, accessed on 20 September 2023), but it is not clear if the AutoGrader from [1] is a yet another different tool with the same name.

With so many tools available one just has to choose one to use, but there is a problem with pre-built assessment automation tools. As nicely stated in [12] the problem is that: "many of the present assessment tools are developed for a local use and only for a certain type of assignments. Hence, they are often not available for a wider use and would be difficult to adapt to another university, anyway".

Another issue with tools, which is shown in [3,6], is that they are built for specific programming languages. Although "some of the systems are language independent. Especially if the assessment is based on output comparison" [6], what is most noticeable is that these tools can be divided into two categories, according to [6]: first, automatic assessment systems for programming competitions and second, automatic assessment systems for (introductory) programming education. Here, we want to put the emphasis on the word "introductory". Even though many tools and papers exist [13–15] on how to automate the assessment of programming assignments, their focus has been on simple introductory programming assignments.

In this paper, the focus is on more complex (advanced) programming assignments that are usually present in courses in later years of one study program. This leaves us with very few options; basically, tools like AutoGrader or DGRADER would be the available options in this case. One might think that using an existing tool is easier than doing the work manually, but there are some issues to think about before deciding.

First, these tools have a certain logic that needs to be studied and understood. Second, often the tools require a teacher to define tests to be used in grading or build a reference implementation that will be used to compare the solution against and decide if the solution is correct. Building reference implementation is a special issue. When it comes to complex assignments, implementing the whole solution can take up to several days of work. Third, the tool is written in a certain programming language, which the teacher might not be familiar with, so learning the language takes time. Next, these tools have limits, and it might be that they are unable to grade everything the teacher needs the tool for, or the tool might accomplish the task in a different way than the teacher expects or wants. Modifying the tool to fit the teachers' needs might be difficult or even impossible in existing tools. Considering all these issues, sometimes using existing pre-built tools is more difficult and unnecessary. Especially with complex assignments where things can be done in different ways, it might be easier to build custom assessment scripts.

This is why *in this study we try to focus on the process of automation with custom scripts (and in some cases together with some existing software for functional testing) rather than the pre-built tools*. In this way, every teacher can make their own process and automate part of the overall assessment. This does not mean that existing tools cannot be used, however. If teachers find an existing tool that does some part of the assessment, they can integrate it

into their own process. By doing it that way, teachers can code their own scripts by using the programming language that they want and combine that with existing tools if one fits. As is described later, building scripts is much simpler than it might sound at first.

3. Methodology

In this work the main goal is to define an assessment process that can be used to grade complex programming assignments, with concrete examples of what and how to automate. The proposed process was used and refined on multiple courses over ten years. Four courses were primarily where the process was used:

- Course 1 (Advanced Web technologies and Services) and Course 2 (Design Patterns) are programming courses on the first and second year of graduate level with 60 students on average each year.
- Course 3 (Introduction to Web Technologies) and Course 4 (Web Application Development) are programming courses at the second and third year of undergraduate level. Course 3 has on average 60 students each year, and Course 4 has 200 students each year.

The introduction identified what is meant by complex assignment, but here a full list is given of different features that are required of students to be built through various assignments. These features are:

- implementation of multi-thread logic
- building custom socket-based protocols with server and clients
- facilitating data collection from diverse external sources such as REST services and IoT sensors
- storing the data within designated repositories, which could be databases, files or similar
- implementing new REST or SOAP services
- bidirectional JMS/MQTT/WebSocket messaging
- multiple DBMS systems (MySQL, HSQLDB, Derby)
- multiple servers (Tomcat, Glassfish/Payara Micro/Web/Enterprise Server)
- Docker containers
- multiple programming languages (JavaScript, Java, C#)
- different user interface (command line, Web)

The main questions we want to answer in this study are:

- Which parts can be automated?
- What methods can be used for which part?
- Are custom scripts difficult to implement?
- How much time is required for a custom script implementation?
- How to integrate some software in mainly script based automation?
- What is the time benefit of using a semi-automated process over a fully manually process?
- What is the benefit at undergraduate level courses in comparison to graduate level courses?

Overview of the Proposed Process

The process to perform automated assessment includes more activities than just uploading data into some tool and getting the results. The proposed process consists of six main phases (Figure 1). Here is an overview of each phase with the main activities:

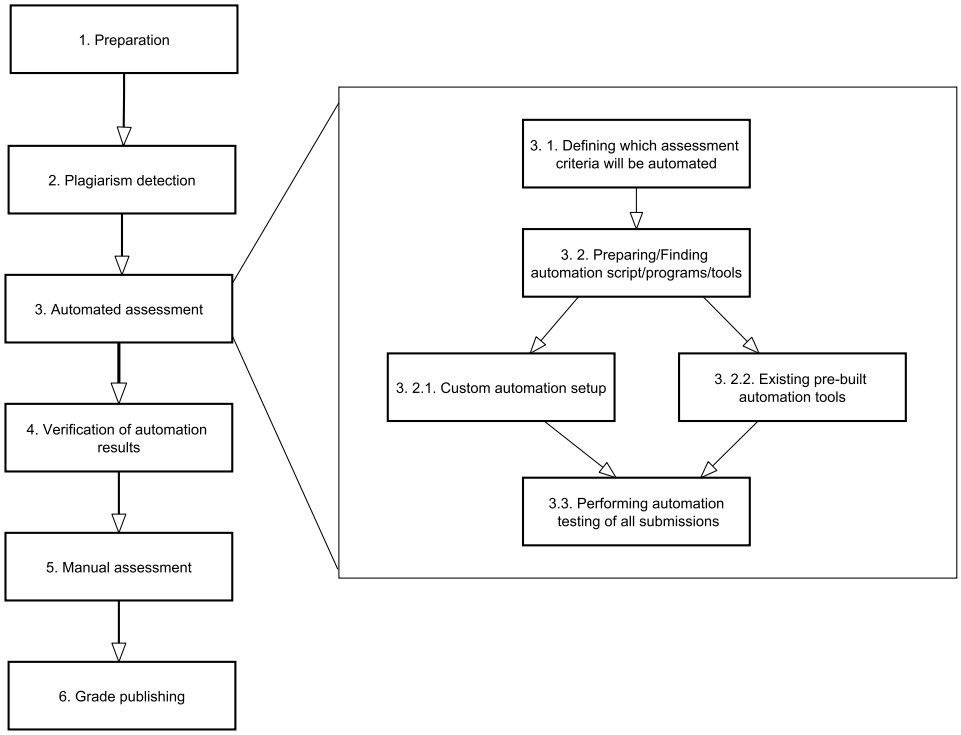

Figure 1. Overview of the assesment process.

1. Preparation—in this phase teacher has to prepare the assignments and the assessment criteria; the phase ends when all submissions are downloaded
   (a) Defining the assignment—here the teacher defines the assignment for students
   (b) Defining the structure of submissions—here the teacher defines the folder structure that has to be used when students submit their assignments
   (c) Defining assessment criteria and scenarios—here the teacher defines the exact criteria used to grade an assignment and the scenarios according to which the assessment of the assignments will be carried out (establishment of the environment for the execution of the assessment, order of execution/calling of individual requests, definition of the expected results for each request, permissible deviations of the results, structure of data records of the individual request results, etc.)
   (d) Answering student questions—here the teacher answers any questions that the students might have about the assignment during the implementation
2. Plagiarism detection—performing similarity detection analysis (e.g., Sherlock, jPlag); if there are some suspicious similarities the teacher should review the results, and if it is found to be a real case of plagiarism these assignments should be removed from further evaluation
3. Automated assessment
   (a) Defining which assessment criteria will be automated—not all criteria can be automated, so it is necessary to define which parts will be automated
   (b) Preparing/Finding automation script/programs/tools
      i. Custom automation setup
         A. Establishing assessment environment and specific software for functional testing (and possible load test and performance measurement) by writing custom scripts for establishing an assessment environment
         B. Implementing assessment scenarios

- Writing custom scripts/programs that will perform the assessment in the prepared environment
- Writing a test plan for selected software using provided components for needed test elements (e.g., In JMeter are thread groups, config elements, listeners, timers, samplers, assertions, etc). Usually, the test plan is written in the GUI part of the software and can be executed in the command line or GUI.

C. Verification of the prepared script/programs/tools on test data

ii. Existing pre-built automation tools

A. Setup the tool
B. Prepare assignments for the tool
C. If necessary and if supported by the tool, write custom tests and/or implement reference solutions

(c) Performing automation testing of all submissions

4. Verification of automation results
5. Manual assessment—manual assessment of program parts that were not automated
6. Grade publishing—calculating the total amount of points and publishing the results for students

Now that the process is known, a more detailed description of each phase will be described.

4. Description and Examples of Automation Process

In this section each phase is described in depth with a special focus on the "Automated assessment and verification" phase. For the "Automated assessment and verification" phase there are multiple subsections describing different cases of automation that were used in four courses.

4.1. Preparation

The automation assessment process begins with the preparation of assignments. There are many things that can go wrong in the automation if the assignment is not defined correctly. When writing the assignments there are three main points to consider: the assignment itself, the structure of the submission, and the assessment criteria.

The assignment itself has to be defined in a way that makes it possible later to have an interface for automated assessment. To describe what this means, here is an exaggerated example. Let's say students have to build a web service. One could say: "build it however you want just make it serve some data about computer games". This would give students complete freedom of how to do the assignment, as long as it is a correctly implemented web service. But for such submissions it would be impossible to automate the assessment because every student would make a different interface. Even more students might implement different kinds of web services (REST, SOAP, GraphQL). Instead of going the open route, it is better to define the exact web service type, define the exact structure of the requests and response, and define the methods that the service offers. This will take more time to define up front, but later on the assessment can be completely automated. *The more explicitly things are specified the easier the automation is; with more freedom, the automation is harder. It is up to the teacher to find a balance between the two.*

The structure of the submission is important because when assessment automation scripts are built it is necessary to know where to look for things like for the main file or the configuration files. One good practice is that a student's username or id should be the name for the root folder. This is useful so one can easily know to whom the assignment belongs, and this information can then be used when generating reports. In complex projects there might be several applications that need to be run, so it is useful to structure the applications into subfolders with specific naming conventions. *Such restrictions are not only useful for*

automation but also teach students how to behave in a team environment, where one must accept some agreements and do things in ways they may not initially prefer.

It is best to define the assessment criteria with the assignment description. Teachers might outline the criteria during the period as students work on their projects, but then it might be too late for students to incorporate all the criteria into their plans. This is especially true if there are some non-functional assessment criteria. For example, if there is a criterion that no function can be larger than 30 lines of code, then that has to be defined in the assignment ahead of time. Such assessment criteria must be told to students upfront so that students can implement them. Of course, there is an exception if the criteria are things that students should know on their own from previous courses. Here are some examples of criteria that are requested of the students in our courses:

1. Correctness of the program—for a given input the program returns a given output;
2. Correctness of the communication protocol—data exchange mechanism like REST request and response structure;
3. Interoperability—possibility of the system to communicate with other servers or clients that follow the same protocol
4. Clean code—readability of the code, number of lines in a function, and maximal length of the line;
5. Visual appearance—of the user interface and user experience when using the application;
6. Code optimization—speed of execution, database query processing;
7. Correct selection and implementation of design patterns.

The teacher does not have to give the assessment criteria to the students, but the students have to be able to see what is graded from the assignment description, aside from the program simply working as intended. For the above mentioned criteria, here is a simple overview of what can be automated based on our experience:

1. Correctness of the program—can be automated in various ways, including bash scripts, unit tests, and external programs;
2. Correctness of the communication protocol—can be automated in various ways, including bash scripts, unit tests, and external programs (e.g., postman);
3. Interoperability—can be automated in various ways, including bash scripts, unit tests, and external programs or we can build a dummy server/client that sends/receives correct and wrong messages and answers correctly or incorrectly;
4. Clean code—can be partially automated, things like number of lines of code can be automated by a simple counting program, but readability can not be automated since it focuses on the perception of another programmer, in this case a teacher. Maybe metrics can be defined but in our case it is just how easy it is to navigate through the code, comments, and the overall look and feel of the code;
5. Visual appearance—can not be automated because the focus is on the user look and feel, and this criteria is partially subjective;
6. Code optimization—partially can be automated by measuring execution time, but things like opening and closing connections to a database inside a loop could be problematic to automated recognition and requires the teacher to look at the code;
7. Correct usage and implementations of design patterns–can not be fully automated, as there are many design pattern mining/recognition techniques [16–18] and software exist, but there are certain expectations about which patterns should be used and for what purpose. Also, there is not necessarily a naming convention for everything. In our case, this is partially graded manually with the help of the attached documentation each assignment must have.

After everything is defined and given to the students, the preparation phase includes answering student questions about the assignment. With complex assignments there might be some parts that students do not understand completely. Everybody that has worked on a real project will know that no specification is perfect, and real questions come up when one is deep into the coding phase. To solve this problem, we found that a forum is a very good way

to answer questions. The benefit of the forum for the students is that everybody sees the questions and the answers. The benefit for the teacher is that they do not have to repeat themselves. Also, students can answer each other, and the teacher can intervene if they see that somebody posted the wrong answer. *The reason this is in the preparation phase is that while answering questions the teacher might see that they have to tweak some criteria if some requirements need to change. Therefore, we see it as preparation.*

Once the students have done their assignments, they have to submit their solutions in some way, usually through a Learning Management System (LMS). In our case, Moodle LMS is used. Accessing the students' submissions means downloading a zip archive file that contains all submitted assignments. Since the students are required to submit their assignments in a curtain structure, one just has to unzip all the assignments. Once the submissions are extracted, we are ready for the next phase.

4.2. Plagiarism Detection

Before going into the assessment itself, a plagiarism detection check should be performed. There are numerous tools available for different languages, like:

- Sherlock [19] (http://warwick.ac.uk/iasgroup/software/sherlock, accessed on 20 September 2023),
- jPlag [20] (https://github.com/jplag/jplag, accessed on 20 September 2023),
- MOSS [21] (http://theory.stanford.edu/~aiken/moss, accessed on 20 September 2023),
- MPC [22] (https://github.com/matnovak-foi/MPC, accessed on 20 September 2023),
- Spector [23] (https://gitlab.com/vimino/spector, accessed on 20 September 2023), just to name a few that are open source.

Such tools detect the similarity between programs and give reports. It is out of the scope of this article to go into how everything is done during that check, but for reference one could read [24,25] for the process of plagiarism detection or [26] for an overview of the source-code plagiarism detection field.

By doing the detection first, it is possible to eliminate the students who are considered as having plagiarized so as not to waste time on their assessments. Of course, the tools just give the similarities in the findings, and the teacher must review the results and decide if it is real plagiarism.

In our case, this phase is automated almost completely, and only the verification of suspicious similarities is done manually. This step can be time consuming if there are multiple suspicious similarities, but it is a necessary step to ensure the integrity of the course and university. Once this is done, the assessment phase can begin.

4.3. Automated Assessment—Case 1

In Section 4.1 different criteria categories and what can be automated were described. But since the above mentioned criteria are generic, the following is a real example of an evaluation form that was filled out by the teacher for each student. These criteria were done for an assignment where students had to build a console client application that communicates with a server with a custom protocol on a socket. There are many details involved in this assignment, but the main logic is to validate the input of a starting client application, validate the input of commands, parse the commands, and return the correct response. In addition, the server needed to communicate with another server also using a custom protocol.

Here are the assessment criteria for this assignment, with info if it is automated or manually assessed:

- GENERAL
    1. Self Evaluation form (manually)—students have to do a self-evaluation where they describe what was done, what parts have bugs, and other information. The form must be submitted as a pdf with the assignment. This form helps the

teacher to grade, especially with partial or buggy solutions. Therefore, students get credit if they fill it out and if it corresponds to the actual state of the program.
2. JavaDoc (manually)—create a complete JavaDoc for their programming code and ensure the quality of the JavaDoc.
3. Prepare different types (txt, xml, bin, json) of configuration files (automated)—all files have to have the same data, and there is a required naming convention.

- WEATHER (METEO) SIMULATOR SERVER AND DISTANCE SERVER
  4. Checking of input parameters during server startup (automated)
  5. Loading and using configuration files (txt, xml, bin, json) (automated)
  6. Loading of weather data from file (automated)
  7. Periodical sending of weather data i.e., simulation of reading weather data over a certain period of time (automated)
  8. Save errors to file after x wrong attempts (automated)
  9. Loading of serialized data from file (automated)
  10. Command: DISTANCE (automated)
  11. Command: DISTANCE SAVE (automated)
- MAIN SERVER
  12. Checking of input parameters during server startup (automated)
  13. Loading and using configuration files (txt, xml, bin, json) (automated)
  14. Implemented multi thread logic (mutual exclusion) (partially automated)
  15. User authentication (automated)
  16. Command: SENSOR (automated)
  17. Command: METEO (automated)
  18. Commands: MAX, ALARM (automated)
  19. Commands: DISTANCE, DISTANCE SAVE (automated)
  20. Command: END (automated)
  21. jUnit tests for main server class (partially automated)
  22. jUnit tests for network worker class (partially automated)
  23. ERRORS in code not related to above logic
- CLIENT APPLICATION (CONSOLE APP)
  24. Checking of input parameters during startup (automated)
  25. Sending auth data at every request (automated)
  26. Sending weather, max temp, max humid, max pressure requests (automated)
  27. Sending alarm, distance, end requests (automated)

One can see that there were 27 different criteria for this one assignment; most were functional and automated. The reason why so many elements were assessed was to check most of the required functionality. This process is done for all courses, with the biggest difference being the functional criteria students have to implement depending on the course subject.

Before any kind of testing could begin, since in this case the programming language Java is used, all assignments needed to be extracted and compiled, that is, we needed to prepare the assessment environment. To do that, a bash script was run that read all student usernames and did the compilation. To be able to do that more easily, the students were required to use Apache Maven as "a software project management and comprehension tool" (https://maven.apache.org/index.html, accessed on 20 September 2023) (earlier it was Apache Ant). The messages for each student were stored in separate files. If the program is run and does not start, then these files are useful to see if there was an error during compilation. The bash script is very simple and is given in full in Listing 1.

4.3.1. Assessment Automation of the Client Application

The assignments have a client side and a server side. The client side can be a console application or web application. For the console application, there are requirements on how to start the application, and the students were required to have input validation for running the program. If the parameters are incorrect, then the program should stop and print the

message. This part of the testing can be easily done by a bash script that tries to run the program with correct and incorrect parameters. First, 20 incorrect attempts were made, and for each attempt it was expected to get a user-friendly error message. Stacktrace message prints were not allowed to be printed. The same tests were made for every student. Since there is no clear description of what the error message will be, the output is stored into a file and manually examined by the teacher. This checking does not consume much time since by simply scrolling through the output one can see if there are some stack trace printouts.

Listing 1. Bash script for compilation.

```
#!/bin/bash
java_dir="/usr/lib/jvm/java-17-openjdk"
input="students.txt"
home=$(pwd)

while read username
do
 echo "SCRIPT DIR: $home"
 cd $home
 echo "COMPILING STUDENT: $username"
 logFile="$home/compile/$username.txt"
 touch $logFile
 echo "COMPILING STUDENT: $username" > $logFile
 gotoStudentDir="cd $home/../homework/"$username"_homework_1"
 echo $gotoStudentDir >> $logFile 2>&1
 eval $gotoStudentDir >> $logFile 2>&1
 echo "----MAVEN INSTALL----" >>$logFile 2>&1
 maven='JAVA_HOME="$java_dir" mvn clean package install -DskipTests'
 echo $maven >> $logFile 2>&1
 eval $maven >> $logFile 2>&1
done < $input
```

Another thing is that this client application communicates with a server. For the purpose of this test, the server was a dummy server built by the teacher. It stored the received message that a student's client application sent and answered with OK. The protocol defines how the server and client have to communicate. It was expected that the server would receive only messages from the client when the client application was started with correct parameters. All messages are stored in a log file.

When the testing was done, the teacher simply went through the log file and looked at the received messages. The commands always have a username, and the testing program set it to the students' usernames to make it easy to recognize which message was received by which student client program. An issue might appear if some applications do not send this information, in which case it is necessary to look at the username of the student name before and after and see who was in between. Checking the log file takes a few minutes, but it is much faster than doing the testing manually. This automation ensures objectivity in testing all student assignments. Building such scripts is very easy and does not require much time. The script has two parts: preparing the environment and running the test scenario. Here is an example of the script for testing the client application (called 02_testMainClient.sh) with the scenario consisting of intentionally wrong parameters. Only two are shown, but others were done in the same way so the script (Listing 2) is not given in full.

Building a server did not require extra time since the skeleton of the main server was built during lab sessions together with students. Students have the basic server given, and the difference is that the teacher's server just answers "OK" and students have to parse the messages further on. By carefully examining the script above, one can see it is to test only one student, but this is intentional. This gives the teacher the possibility to run this script for each student individually. But there is another script (Listing 3) that runs it multiple times for each student once.

Listing 2. Bash script for testing client application.

```
#!/bin/bash
### PART 1: PREPARING THE ENVIRONMENT ###
java="/usr/lib/jvm/java-17-openjdk/bin/java"
target_dir="target/libs/*:target/*"
username=$1
package="org.uni.course1.$username.homework_1.MainClient"
exec_command="$java -cp $target_dir $package"
clear;

port=8999
if [ -z "$2"]; then
        port=8999
else
        port=$2
fi

gotoStudentDir="cd ../submissions/$1_homework_1/$1_homework_1_app"
echo $gotoStudentDir
eval $gotoStudentDir

echo "###NEXT ARE COMMANDS WITH WRONG PARAMETERS!###"

### PART 2: INTENTIONALLY WRONG PARAMETERS SCENARIO ###
command="$exec_command"
echo "NO PARAMETERS: "
eval $command

command="$exec_command -k $1 -a localhost -v $port -t 1000 --meteo SENSOR-DHT11"
echo "MISSING PASSWORD: "
eval $command
...
```

Listing 3. Bash script for testing all submissions.

```
#!/bin/bash
input="students.txt"
while read username
do
        echo "TESTING STUDENT: $username";
        saveLog="mainClientTests/$username.txt"
        ./02_testMainClient.sh $username > $saveLog 2>&1
done < $input
```

4.3.2. Assessment Automation of the Server Applications

The server applications are a bit more difficult to test since there is more logic there. But by splitting this into multiple steps it is not difficult to accomplish. First, there are three servers in this particular case of assignments that simulate a microservice architecture. Two of the servers are standalone servers offering support to the third. To test this first, it is best to test the standalone servers. For each server a bash script (called startDinstanceServer.sh—Listing 4) was built to start the server (i.e., preparing the environment).

In contrast to the client application, this script is not run directly, but there is a custom Java-testing application based on the jUnit framework, which performs the scenario. Java unit tests were used since they have the possibility to assert correctness. A nice output was produced if the tests passed and if they supported parametric tests. *Parametric tests are important, since this enables them to treat student usernames as parameters and then run the same tests on all students and get reports per student.*

Listing 4. Bash script for starting a student server.

```
#!/bin/bash
java="/usr/lib/jvm/java-17-openjdk/bin/java"
target_dir="target/libs/*:target/*"
gotoStudentDir="cd /2022-2023/c1/hw1/submissions/$1_homework_1/$1_homework_1_app"
echo $gotoStudentDir
eval $gotoStudentDir

package="org.uni.course1.$1.homework_1.DistanceServer"
exec_command="$java -cp $target_dir $package"

command="$exec_command test_config_main_server.txt"
echo "RUN SERVER:"
eval $command

echo "SERVER STOPPED";
```

Before starting the actual testing server configuration files need to be prepared. Each server has a file with some configuration instructions, including which port to start on, how many threads to use, and so on. The first "test" basically just takes some prepared files and copies them to the student's folder. It is important to note that during server startup the configuration file is given as a parameter; in this way there is no need for a specific naming convention of such file. But there is a convention for the structure and type of file (xml, json, ...). This unit test was not evaluated, as it was just a preparation.

Next was a unit test that will check for students' config files. Students had to prepare config files in 4 different versions: json, xml, bin and yaml. Even though they were not used during testing, the students received credit for preparing these files. The problem here is that it is important to have a naming convention so that the existence of the files can be automatically checked. The test consists of logic to read a file and check if it contains the required parameters, but the values are not tested. This unit test is simple since it can be done without a running server.

The next step was to test the first standalone server. The unit test itself was simple. Its role was that for a given command a server was expected to answer in a specific way. In Listing 5 is one example of a correct and incorrect message unit test. The command tested was DISTANCE, which calculates the distance between two GPS locations on the surface of a sphere (Earth). The correct syntax is DISTANCE gps_latitude1 gps_longitude1 gps_latitude2 gps_longitude2, whereby the values for latitude and longitude are decimal numbers with 5 decimal places.

Listing 5. Example of jUnit tests for standalone student server.

```
@Test(timeout = 1500)
public void DISTANCE_EDDF() {
        String response = client.run("DISTANCE 46.30771 16.33808
        46.02419 15.90968");
        assertFalse(client.error);
        assertTrue(response.startsWith("OK"));
        assertTrue(response.startsWith("OK 45"));
}

@Test(timeout = 1500)
public void DISTANCE_WRONG_COMMAND_NAME() {
        String response = client.run("DIST EDDF LOWW");
        assertFalse(client.error);
        assertTrue(response.startsWith("ERROR 10"));
}
```

As can be seen, the unit tests are quite straight forward. The client used to send a message was built during lab sessions, so it was only necessary to define the request and the response. *The main issue however lies in how to start the server and shut it down.* The unit tests have good methods, setUp and tearDown, that were activated before each test. This was unnecessary, however, because the idea was to start the server for each student once since it takes time to do it before each individual test.

This issue was solved so that each time a setup method (Listing 6) was called, the studentUsername, which is the parameter in the parametric run, was stored in a global variable and checked to see if the username changed. If so, the server shut down and a new instance was run with the other username. Starting the server was simply done by the above shown bash script (Listing 4).

The pkill command was used to stop the server or, more accurately, to stop the script. The process object is stored while calling the bash script. In case the student server fails to be killed, the process object is checked. If the process object is not null, then there is no point to continue further with the assessment and the whole scenario stops. All this could be done just in Java, but this was the simplest way to do the job and have it work without issues. We tried to implement the whole process just in Java, but sometimes the server would not shut down, so the presented solution was used.

Listing 6. SetUp method for starting and stoping student server.

```
@Before
public void setUp() throws IOException, InterruptedException {
        System.out.println("END: "+previousUsername);
        System.out.println("START:"+global.studentUsername);
        client = new Client(8112);

        if (process != null && previousUsername.compareTo(global.studentUsername) != 0) {
                System.err.println(previousUsername + " = " +
                global.studentUsername);
                Runtime.getRuntime().exec("pkill -f " +
                this.previousUsername);
                Thread.sleep(250);
        }
        if (process == null) {
                File log = new File("distanceServerLogs/" +
                global.studentUsername + ".txt");
                if (!log.exists()) { log.createNewFile(); }
                String[] command = new String[]{"sh", "-c",
                        global.pathBashScripts + "startDinstanceServer.sh " +
                        global.studentUsername + " >> " + log + " 2>&1"};
                System.out.println(command[2]);
                process = Runtime.getRuntime().exec(command);
                System.err.println("SERVER PID:" + process.pid());
                Thread.sleep(1000);
        } else {
                System.exit();
        }
}
```

The second standalone server was done in the same manner as the first server. The third server was not standalone, but nonetheless the testing was done in a similar way. First, we tested the third server without starting the other servers to see if the server returned the correct error message. Next, we ran the server with other servers running, but in this case the student servers were not used; rather, two dummy servers were run that returned preconfigured answers depending on the command received. In this way, the unit tests could easily check if the correct answer was returned. This was done because sometimes the two standalone servers did not work correctly or return the data depending on the input, making it more difficult to check if the answer was correct.

One special test situation was to see if multithreading works. An example of a unit test is shown in Listing 7. Here, the important part was to have a timeout, since often students' servers block when multiple requests are sent at once. The first test will fail if the requests are not served in 1 second and an "OK" status is returned. In the second test, it is expected to get an "ERROR" status code since a maximum of 5 threads should be allowed.

Listing 7. Example of jUnit tests for multithread testing.

```
@Test(timeout = 1000)
public void TEST_MULTITHREADING_WORKS() throws InterruptedException{
        Client client = new Client(8112);
        String response = client.run("AIRPORT LIST");
        assertTrue(response.startsWith("OK"));

        for(int i = 0; i<4; i++){
                Thread d = new Thread("AIRPORT LIST");
                d.start();
        }
        response = client.run("AIRPORT LIST");
        assertTrue(response.startsWith("OK"));
}
@Test(timeout = 6000)
public void TEST_FOR_MAX_5_THREADS() throws InterruptedException{
        Client client = new Client(8112);
        String response = client.run("AIRPORT LIST");
        assertTrue(response.startsWith("OK"));

        for(int i = 0; i<10; i++){
                Thread d = new Thread("AIRPORT LIST");
                d.start();
        }

        response = client.run("AIRPORT LIST");
        assertTrue(response.startsWith("ERROR"));
}
```

Since the above test for multithreading takes some time, there was no point to test for students who did not implement it and wait for the test to timeout. In Java there are different methods to implement mutual exclusion, but in our course two possibilities were used to implement it: using synchronized keyword and/or concurrent package. A simple check using grep was used to find which students had implemented this logic. The commands are shown in Listing 8. The commands (Listing 8) give the usernames of students who have potentially implemented the logic, and then for those the above test is done.

Listing 8. Commands for searching implementations of mutual exclusion.

```
grep -iRl "concurrent" --include="*.java" ./
grep -iRl "synchronized" --include="*.java" ./ > synchronized.txt
```

Most of the logic can be tested as described in the above examples. Using unit tests is very easy, because it gives nice feedback and can be run for all students at once, and many variations can be tested. The first year building this setup was a bit time consuming, but in following years it is not since all that is required is just to change the commands that are sent and the responses that are expected. Sometimes some commands have to store something in a file and then there is a test at the end that checks the content of such files if they exist. In such a case, it is necessary that the file names where the data is stored are part of the configuration files. This is a minimal problem to check and change their names.

We want to emphasize that in the previous example some parts can be partially replaced with software like Apache jMeter. Firstly, one must define a test plan in its GUI mode. The test plan should be parameterized in order to get some data outside of the tool so it can be run in Command Line Interface (CLI) mode as part of the script that is very similar to the previous ones that handle each student, prepare the environment, and start all necessary servers. In this example, for testing a Distance Server (Figure 2) two consecutive thread groups were created, first with one thread for test commands and second with ten threads for test commands in multithreading mode. Both groups had the same content that consisted of two types of samplers: HTTP Raw Request and TCP Sampler. Both sampler types work similarly, and each of six samplers is dedicated to a particular command and its parameters, which can be correct or incorrect according to the defined protocol. The results of their work are based on Response Assertion, where for each sampler is expecting a specific response that is defined with a pattern. In a case of pattern matching, the sampler returns true, otherwise false with some failure message. When the server being tested has an error while processing a request/command, then JMeter gives information in the form of a response message (Figure 3).

To minimize the teacher's workload, it is very convenient to automate as many steps as possible. In this case, JMeter running in CLI mode will generate one file for each student. At the end of the assessment, the teacher should get some kind of joint overview of data of all students. The script in Listing 9 takes each student one by one, prepares the environment (excluded for clarity), starts the server (excluded for clarity), executes JMeter with the prepared test plan and other parameters, analyses the JMeter output file to get information about success and failure commands, determines the percentage of success commands, and writes information to the report file in csv format. In Figure 3 the content is shown in a spreadsheet form.

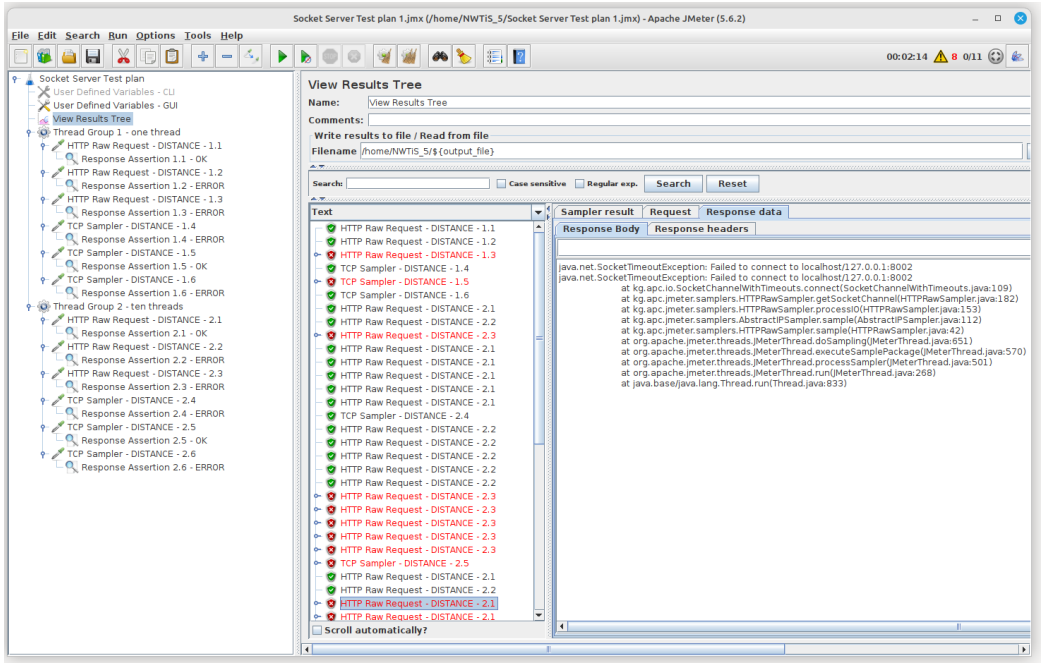

Figure 2. Apache jMeter testing Distance Server over GUI.

	A	B	C	D	E	F	G	H	I
1	elapsed	label	responseCode	responseMessage	threadName	success	failureMessage	grpThreads	allThreads
2	3	HTTP Raw Request - DISTANCE - 1.1	200		Thread Group 1 - one thread 1-1	true		1	1
3	2	HTTP Raw Request - DISTANCE - 1.2	200		Thread Group 1 - one thread 1-1	true		1	1
4	502	HTTP Raw Request - DISTANCE - 1.3	500	java.net.SocketTimeoutException: Timeout exceeded while reading from socket	Thread Group 1 - one thread 1-1	false	Test failed: text expected to contain /ERROR 10/	1	1
5	2	TCP Sampler - DISTANCE - 1.4	200	OK	Thread Group 1 - one thread 1-1	true		1	1
6	501	TCP Sampler - DISTANCE - 1.5	500	org.apache.jmeter.protocol.tcp.sampler.ReadException: Error reading from server, bytes read: 0	Thread Group 1 - one thread 1-1	false	Response was null	1	1
7	4	TCP Sampler - DISTANCE - 1.6	200	OK	Thread Group 1 - one thread 1-1	true		1	1
8	3	HTTP Raw Request - DISTANCE - 2.1	200		Thread Group 2 - ten threads 2-1	true		1	1
9	1	HTTP Raw Request - DISTANCE - 2.2	200		Thread Group 2 - ten threads 2-1	true		1	1
10	501	HTTP Raw Request - DISTANCE - 2.3	500	java.net.SocketTimeoutException: Timeout exceeded while reading from socket	Thread Group 2 - ten threads 2-1	false	Test failed: text expected to contain /ERROR 10/	6	6
11	404	HTTP Raw Request - DISTANCE - 2.1	200		Thread Group 2 - ten threads 2-2	true		6	6
12	304	HTTP Raw Request - DISTANCE - 2.1	200		Thread Group 2 - ten threads 2-3	true		6	6
13	212	HTTP Raw Request - DISTANCE - 2.1	200		Thread Group 2 - ten threads 2-4	true		6	6
14	115	HTTP Raw Request - DISTANCE - 2.1	200		Thread Group 2 - ten threads 2-5	true		6	6
15	16	HTTP Raw Request - DISTANCE - 2.1	200		Thread Group 2 - ten threads 2-6	true		6	6
16	12	TCP Sampler - DISTANCE - 2.4	200	OK	Thread Group 2 - ten threads 2-1	true		6	6
17	11	HTTP Raw Request - DISTANCE - 2.2	200		Thread Group 2 - ten threads 2-2	true		6	6
18	10	HTTP Raw Request - DISTANCE - 2.2	200		Thread Group 2 - ten threads 2-3	true		6	6
19	11	HTTP Raw Request - DISTANCE - 2.2	200		Thread Group 2 - ten threads 2-4	true		6	6
20	11	HTTP Raw Request - DISTANCE - 2.2	200		Thread Group 2 - ten threads 2-5	true		6	6
21	11	HTTP Raw Request - DISTANCE - 2.2	200		Thread Group 2 - ten threads 2-6	true		6	6
22	501	TCP Sampler - DISTANCE - 2.5	500	org.apache.jmeter.protocol.tcp.sampler.ReadException: Error reading from server, bytes read: 0	Thread Group 2 - ten threads 2-1	false	Response was null	10	10
23	500	HTTP Raw Request - DISTANCE - 2.3	500	java.net.SocketTimeoutException: Timeout exceeded while reading from socket	Thread Group 2 - ten threads 2-2	false	Test failed: text expected to contain /ERROR 10/	10	10
24	501	HTTP Raw Request - DISTANCE - 2.3	500	java.net.SocketTimeoutException: Timeout exceeded while reading from socket	Thread Group 2 - ten threads 2-3	false	Test failed: text expected to contain /ERROR 10/	10	10
25	501	HTTP Raw Request - DISTANCE - 2.3	500	java.net.SocketTimeoutException: Timeout exceeded while reading from socket	Thread Group 2 - ten threads 2-4	false	Test failed: text expected to contain /ERROR 10/	10	10
26	501	HTTP Raw Request - DISTANCE - 2.3	500	java.net.SocketTimeoutException: Timeout exceeded while reading from socket	Thread Group 2 - ten threads 2-5	false	Test failed: text expected to contain /ERROR 10/	10	10
27	501	HTTP Raw Request - DISTANCE - 2.3	500	java.net.SocketTimeoutException: Timeout exceeded while reading from socket	Thread Group 2 - ten threads 2-6	false	Test failed: text expected to contain /ERROR 10/	10	10
28	430	HTTP Raw Request - DISTANCE - 2.1	200		Thread Group 2 - ten threads 2-7	true		10	10
29	10	TCP Sampler - DISTANCE - 2.4	200	OK	Thread Group 2 - ten threads 2-3	true		10	10
30	9	TCP Sampler - DISTANCE - 2.4	200	OK	Thread Group 2 - ten threads 2-4	true		10	10
31	10	TCP Sampler - DISTANCE - 2.4	200	OK	Thread Group 2 - ten threads 2-5	true		10	10
32	10	TCP Sampler - DISTANCE - 2.4	200	OK	Thread Group 2 - ten threads 2-6	true		10	10
33	8	HTTP Raw Request - DISTANCE - 2.2	200		Thread Group 2 - ten threads 2-7	true		10	10
34	100	TCP Sampler - DISTANCE - 2.6	500	java.net.SocketTimeoutException: Connect timed out	Thread Group 2 - ten threads 2-1	false	Response was null	10	10
35	100	TCP Sampler - DISTANCE - 2.4	500	java.net.SocketTimeoutException: Connect timed out	Thread Group 2 - ten threads 2-2	false	Response was null	10	10
36	504	HTTP Raw Request - DISTANCE - 2.1	500	java.net.SocketTimeoutException: Failed to connect to localhost/127.0.0.1:8002	Thread Group 2 - ten threads 2-8	false	Test failed: text expected to contain /OK 46.0/	9	9
37	504	HTTP Raw Request - DISTANCE - 2.1	500	java.net.SocketTimeoutException: Failed to connect to localhost/127.0.0.1:8002	Thread Group 2 - ten threads 2-9	false	Test failed: text expected to contain /OK 46.0/	9	9
38	506	HTTP Raw Request - DISTANCE - 2.1	500	java.net.SocketTimeoutException: Failed to connect to localhost/127.0.0.1:8002	Thread Group 2 - ten threads 2-10	false	Test failed: text expected to contain /OK 46.0/	9	9

Figure 3. Apache jMeter tesing results.

Listing 9. Apache jMeter testing distance server over CLI.

```
#!/bin/bash
input="students.txt"
logFile="report.txt"
echo "username;success;fail;total;%" > $logFile
server=localhost
port=8002
jmeter=/opt/apache-jmeter-5.6.2/bin/jmeter

while read username
do
# prepare environment
# start student x distance server
        rm $username"_test.csv"
        $jmeter -n -t Socket_Server_Test_plan_1.jmx \
                -Jserver_name=$server -Jserver_port=$port -Joutput_file=$username"_test.csv"
        result_success=$(awk 'BEGIN { FS = ";" } ; \
                { if($6 == "true") print NR, $6 }' $username"_test.csv" | grep -c "true")
        result_fail=$(awk 'BEGIN { FS = ";" } ; \
                { if($6 == "false") print NR, $6 }' $username"_test.csv" | grep -c "false")
        total=$((result_success + result_fail))
        result=$((result_success * 100 / total))
        printf "$username;$result_success;$result_fail;$total;$result\n" >> $logFile

# shutdown student x distance server
done < $input
```

Assessment criteria might expect different types of information than are what is presented above. It is possible to extend the script to get more detailed information for the report based on the JMeter output file (i.e., grouping by sampler or sampler type).

4.4. Automated Assessment—Case 2

The same principles are used as in other courses, but there are some differences. For example, in course 3, instead of Java the programming language is JavaScript on a Node.js environment. Here, the students had to implement (among other things) a web interface and a REST service. The part that can be automated is the REST service. This could be done as in case 1, but here it was done using only bash scripts and the curl command.

First the server had to be started. The script that started the server was exactly the same as in case 1 (Listing 4), except that Node.js was used. The run command is given in Listing 10. Once the server was running, the scenario implementation was as shown in Listing 11.

One important thing is the status code. The curl command first just made the request and checked the HTTP status code. The function for asserting was implemented with the same idea as how a unit test works: green if it is ok, red if it is not, and then write what is received and what was expected. The implementation is given in Listing 12.

Since the response is in JSON format, the jq command was used to parse information out and print it. An example is given in Listing 13. This assignment did not have to specify the whole structure of the response data, so one could print the response data to the screen and then just check manually: if the data is in the correct type (eg. json, xml), if it contains some expected data, etc. Sometimes it is enough to count the number of rows returned, which can be automated, for example, when there is pagination and it is expected to get 3 rows, as shown in Listing 13. In this example, some other bash commands/programs were used, such as: wc, grep, or similar. The assert function was the same as for the status code since it basically checked if two values were the same.

Listing 10. Example of Node.js run command.

```
node --experimental-fetch restServer.js ../../service-conf-ok.csv
```

Listing 11. Examples of testing REST service using bash.

```
echo "###--GET---api/users-####"
url="$server/api/users?$auth"
code=$(curl -s -o /dev/null -w "%{http_code}" $url)
assertEqual 200 $code
curl -X GET $url
echo -e "\n\n";

echo "###--GET---api/users/user1-####"
url="$server/api/users/user1?$auth"
code=$(curl -s -o /dev/null -w "%{http_code}" $url)
assertEqual 200 $code
passwordUser=$(curl -s $url | jq '.password')
assertEqual "123456" $passwordUser
curl -X GET $url
echo -e "\n\n";
```

Listing 12. Implementation of assertEquals method in bash.

```
assertEqual (){
        RED='\033[0;31m'
        GREEN='\033[0;32m'
        BLACK='\033[0m'
        expected=$1
        recieved=$2
        if [ $expected -eq $recieved ]; then
        echo -e "$GREEN OK $BLACK";
        else
        echo -e "$RED Expected: $expected = $recieved :Recieved $BLACK"
        fi
}
```

Listing 13. Example for testing REST service using external commands.

```
query="page=1&perPage=3&category=5488998"
echo "###--GET---api/filmovi $query-####################"
url="$server/api/movies?$query"
code=$(curl -s -o /dev/null -w "%{http_code}" $url)
assertEqual 204 $code
curl -X GET $url
echo ""
numberOfMovies=$(curl -s -X GET $url | jq '.[] | .id' | wc -l)
assertEqual 3 $numberOfMovies
echo -e "\n\n";
```

The web part of the assignment can not be automated, since the user experience and design are graded together with the logic. It is not worth wasting time on automation rather than simply clicking through the application. What can be automated is starting the server. If there is an input configuration file, it can be checked if it is correct. During the startup if there are any log files or databases required that are standard, they can be cleared out or filled with dummy data so that during testing one can look for this data, such as a search.

4.5. Automated Assessment—Case 3

The presented techniques in case 1 and case 2 work well, but there are some assignments that use external servers like Apache Tomcat, Payara, or GlassFish. In addition, some assignments require a working database like MySQL, Apache Derby, HSQLDB, or similar. In such assignments, there is the need to deploy applications to the corresponding server and reset the database before each student can be tested. Over the years there have been different ways for accomplishing this. Regarding the deployment, because we used Maven (or Apache Ant), this was not an issue. There is a pom.xml or build.xml where it can be configured to make the deploy right after compilation. For example, we used the mavan cargo plugin.

The database reset was a bit more complicated. Here, a simple Java program was written that connects to the database and that clears a table of any data and then inserts new data. When writing these assignments, the database structure was fixed. Fixing the database structure is important since otherwise it is not possible to do this kind of reset. Sometimes the applications also have configuration files that contained some information used at runtime. Since these files have exact naming conventions, we were able to change the values of the parameters. The parameter names in the configuration files were fixed to make this possible.

In the last year, we introduced docker to the course. These servers are each in a separate docker container. This might seem like it would complicate testing, but actually it makes it easier. The reason is the database can be a separate docker container that has specific data. Now, before each test this container can be easily reset. It is necessary to have the containers in the same network. This is a small problem, but simply running a bash script (Listing 14) can stop a container and delete the network and then start it again. The networks are always different for each student, so the script has to take the username as a prefix. There is a difference in names, but it is important to have a naming convention so that it can be automated. Here is the first part that restarts/stops/removes the docker containers from the previous students. It uses just awk and grep.

Listing 14. Example for testing REST service using external commands.

```
#!/bin/bash
homeDir="/2022-2023/course3/hw2"
cd $homeDir/test

dockerUser=$(docker network ls | grep network_1 | awk '{print $2}')
previousUsername=$(echo $dockerUser | awk -F '_' '{print $1}')
echo $previousUsername
docker stop $previousUsername"_payara_micro"
docker rm $previousUsername"_payara_micro"
docker stop $previousUsername"_payara_micro"
```

4.6. Automated Assessment Verification and Manual Assessment

Once the automated part is done, there are still some criteria that can not or are not worth automating. In this phase, then, the teacher manually examines the assignments.

When looking back at case 1, what is left is to check the documentation, the unit tests written by students, Java doc, and the implementation of multi thread logic if it works. Also, teachers should take a quick look through the code to see if the code is clean and how

easy it is to navigate. Overall, most of the hard work is automated. It still takes time to look at the test results and check it in Excel form, but the evaluation is mostly trivial and can be done rather quickly.

On average, it took 10 min per student to go through one submission with the automation scripts in place. More time was required for students who had some issues with compilation. There was an attempt to fix the errors to test the logic, but that did not take more than 5 min of the teacher's time. If it can not be fixed in 5 minutes, it was 0 points for this part of the assignment. Additionally, if the teacher fixes it, depending on the error type, the student lost some points.

With so many assessments being automated and checked, the teacher can focus on the user experience and the overall look and feel of the application and code and instead worry about if something "works". The only problem is if something does not work but then there is no possibility to check the design and the student gets 0 points for this part. Even with manual assessment, establishing the assessment environment is necessary (compile, start) and it is fully automated.

One might ask why Selenium and similar software were not used for automation. The reason is simply that the user interface specification is not firmly prescribed, and it was left for students to express their creativity.

4.7. Grade Publishing

In this phase, the teacher has to sum up all the points. In our case, this means using simple Excel and just entering points for each criterion. The sum is then calculated automatically. Sometimes the teacher does not have to enter the points but can enter a number of errors, and based on that calculate the number of points for some category. Finally, when the number of points have been entered, the students are given the results. Feedback is partially generated automatically based on the defined criteria. If the student has all points for a criterion, then the message is positive; if not, then they will get a message that they did not get all points for this criterion. This might be improved by calculating the percentage for how well a criterion is done, as shown in Listing 9. Some feedback is written manually by the teacher. This is then published to the students.

After the students have had time to review their scores, they have a place to come ask questions about the points. Here, having automated scripts is also useful since it enables the teacher to quickly rerun some tests and show the students exactly what the errors were.

5. Discussion

Based on the use cases, one can see that automating complex assignments doesn't always have to be demanding work. There are prebuilt tools, but building custom scripts is not very complicated and they can be used to automate many things in the area of assessments. When one starts from scratch, it does take a little more time to create and test the scripts before putting them into use. But, in subsequent years those scripts can be reused and optimized. Since they are custom, the teacher can use any language that they are familiar with. In our case, we used Bash, Java, and Maven. But they can be translated to any other language.

Pre-built tools might have the benefit of a clearly defined interface, but they have the restriction of a programming language and expectations of how and what to test. The biggest problem with this study was that the existing pre-built tools did not have all the elements we needed, so a semi-automated option was the best approach. As stated in [8] "They typically struggle, however, to grade design quality aspects and to identify student misunderstandings". For an in-depth comparison of pre-built tools, we recommend looking at Table 1 in [3] where 30 tools were compared. In addition, limitations of tools can be found in articles like [27].

In this paper we demonstrated how automation can be done, that it can very easily be customized, and there are no reasons not to utilize automated assessments. If a teacher teaches programming, it should be easy for them to build such automation scripts.

These cases also demonstrated that knowing the command line tools like grep, awk, sed, wc goes a long way in making these scripts. Most logic was done using bash scripting, unit testing, and using Maven (Ant). It was also shown that in certain cases it is possible to use special software like JMeter (run in CLI mode and is easily configured for the required task through its GUI mode) as a part of the automated assessment process.

There are some special challenges when it comes to application servers, docker containers, databases, etc. At first, using docker seemed like it would only complicate the assessments, but in the end it made things easier.

In the methodology section there were several questions that can now be answered.

- Which parts can be automated?—Almost all functionalities can be automated, and the most important thing is to make a good assignment description. Some non-functional requirements can not be automated, such as source-code quality, user interface look and feel, how the code is implemented, design patterns selection, and implementation. In our case, we first ran the scripts to test correctness and later we manually went through the web application interface to test for usability to see how the design of the web page and overall user-experience turned out. When talking about design patterns there are certain patterns that were done in lectures, and it was expected that the students use those and not some other design patterns. When talking about source-code quality we looked for how well the variables, function, and classes were named; how easily we could find our way in the code; and how well the documentation (comments) was written.
- What methods can be used for which part?—Simple bash scripting and unit tests are very useful for automation, and both can be used, but unit tests are designed to assert outcomes so they might be a better choice. Bash scripts have the benefit that one can easily utilize various command line tools and automate stuff like compilation, starting servers, etc.
- Are custom scripts difficult to implement?—As seen from the cases presented, building custom scripts is relatively easy, and, while it takes some time to build them from scratch, the following year they just have to be used again. Building custom scripts helps the teacher to know the inside out of the implementation. One can easily modify it when needed. Teachers that teach coding already know how to code, so why not use this knowledge?

  While implementing scripts is not difficult, there is one big question each year: "Does the script work correctly?" To ensure that the script works, at first it needs to be tested. For that, we used the submissions of 5 students. To choose the 5 students we had a self-evaluation survey where students had to estimate the completeness of their assignment. The students who give themselves 90% or more were considered. In addition, usually there are more than 5, so based on the teachers' knowledge from exams or activity during class, the most promising 5 were chosen. These students were tested first with the scripts, and basically their solutions were test cases for the script. If some of the script tests failed, they were checked manually along with the students' submission source-code. If there was an issue, the script was corrected; if not, it was tested with the next student. Usually with the first three students all issues were resolved. The issues might be that there was an unintentional mistake in the assignment description, or some point was not clear enough, or students misunderstood something, but based on the solution we think this should also be accepted as the correct solution. In rare cases it happened that a mistake was found while grading some other student. In this case, the test case was corrected and rerun for all students. While all this takes up more time it ensures that there are no mistakes in grading. At the end after the grades were published, the students got feedback and they had the opportunity to attend a consultation if they thought that some parts were not graded correctly. At this stage, we never had a situation where the student did not get some points because of a failure in the test script. This confirms to us that the semi-automated approach was done well.

- How much time is required for a custom implementation script?—Building a script for compilation requires about 1 h of work. Building a bash script from scratch that tests a rest service was done in 4 h. Unit tests have taken longer, but mostly the part of starting and stopping the server took about 5 h time. But once done, now it is only a minute per unit test to write/change a test. In Tables 1 and 2 column *Script creation* gives the exact number how much time was spent on building a custom script for an assignment in the particular academic year.
- How to integrate some software in mainly script based automation?—To integrate external software is not complicated, and we have done it many times. In our case the integration was simple since we chose software that has a CLI. CLI makes things much easier for integration into scripts than any GUI. With a CLI, any external program is as simple as using any other shell command.
- What is the time benefit of using a semi-automated process over a fully manual process?—Automation or semi-automation enables testing more thoroughly than would be possible manually. Having 50 test cases for one assignment is possible with automation or semi-automation while doing it manually it is not. When the number of students is low, the benefits are less visible, but even with 10 students with automation we can test the assignments more thoroughly than is possible manually. Semi-automation in comparison to manual assessment in our experience saved at least 50% of time.
  To be more concrete, in Table 1 we present the data on correction times for 5 academic years. The data was acquired by using the pomodoro technique [28] to measure time spent on activities. This technique has been used by the author continuously over 7 years for all activities. In the first two years for which we have the data 2018/2019 and 2019/2020 the grading was done manually without any scripts. From 2020/2021 we started to build automated scripts. In column *Correcting* is the number of hours spent on correcting all assignments. The number of submitted assignments is given in column *No. of submissions*. Divide the correction time by the number of submissions is presented in column *per submission*, which is the time spent for correcting one submission. From this one can see that we went from 35 min per submission to 12 min. Still, we have to include the time spent on building the scripts, so the total time (presented in column *Total*) decreased from 35 h to around 17 h for all submissions. This is confirmed also in Table 2 where the data are presented for the case 2 assignment. In this case, only two academic years are presented, since the course has existed for only two years. For this year, after correcting all assignments, an estimation was made. Three submissions were graded manually; an average was calculated and multiplied by the number of submissions to get an estimated amount of time that would be required to manually grade all submissions.
- What is the benefit at undergraduate level courses in comparison to graduate level courses?—In our experience, the biggest benefit comes with more complex assignments that have many different parts but a well-defined interface. If the assignment is simple, it can be easily and quickly checked manually. In such case, setting up scripts for automation takes more time than manual assessment. Also, if the interface is badly defined, it is a problem to test it. According to our experience, on both levels the assignments were complex enough to justify the invested effort to implement automation. On the other hand, in some courses where the basics of HTML and CSS are parts of the course content, it is faster to manually look through all HTML pages since the design has to be checked manually anyway. Automation can be useful here to some extent if something specific had to be used to determine if the students used that specific piece.

Table 1. Correcting times for Case 1 Assignment.

Ac. Year	No. of Submissions	Type	Scripts Creation (h)	Correcting (h)	Per Submission [1] (min)	Total [2] (h)
2018/2019	55	manual		34	37.09	34
2019/2020	61	manual		36	35.41	36
2020/2021	78	automated	6.5	15	11.54	21.5
2021/2022	76	automated	2	13	10.26	15
2022/2023	74	automated	4	15	12.16	19

[1] Correcting time devided by number of submissions. [2] Correcting time plus script creation time.

Table 2. Correcting times for Case 2 Assignment.

Ac. Year	No. of Submissions	Type	Scripts Creation (h)	Correcting (h)	Per Submission [1] (min)	Total [2] (h)
2022–2023	65	automation	4	20	18.46	24
2023–2024	68	automation	3	17	15.00	20
2023–2024	68	manual estimated [3]			33	37.4

[1] Correcting time devided by number of submissions. [2] Correcting time plus script creation time. [3] Three submissions were corrected manualy. Average time per submission was caculated and multiplied by number of submissions.

6. Conclusions

Assessment automation is a useful way of performing a task, especially for teachers with larger classrooms. In this work the focus was on complex programming assignments, which have quite a different assessment structure than simple introductory programming assignments. In complex assignments there are more things that need to be tested, and the testing cases are not as predictable as in simple assignments.

A lot of papers [13–15] have focused on introductory programming courses, but there is an open space in the domain of more complex assignments. Also, researchers often focus on building different tools (as seen from [8]) rather than focusing on the process that goes into it. This paper fills this gap. Pre-built tools are useful, but they have their limits, especially in grading complex assignments due to a complex environment that has to be established.

In this paper a process was described for automating assessments, which shows that assessment automation is a much wider topic than just building a tool. In our experience, it is a process that starts with a good description of an assignment and its assessment criteria.

It was demonstrated that using pre-built tools is not necessary and that by utilizing simple programming techniques one can go a long way towards assessment automation. The described process does not exclude external tools; rather, it encourages teachers to integrate them when needed. One premise when doing our automation was that by using this process the assessment quality would increase (or at least not decrease) while the time required to do the assessment would obviously be reduced.

In addition, this process emphasized that preparation is as important to assessment automation as is the implementation. Even more, without a good assignment description it is practically impossible to automate assessments. One needs a good defined interface in order to achieve efficient assessment automation. Defining an interface is not only useful for automation but it also gives students an overview of what it is like to work in a corporate environment. In a corporate environment, requirements exist that a software has to fulfil, for example interoperability, and a developer cannot be an isolated individual person who does not respect the established rules.

Author Contributions: Methodology, M.N. and D.K.; Software, M.N. and D.K.; Validation, M.N. and D.K.; Investigation, M.N. and D.K.; Writing—original draft, M.N.; Writing—review & editing, D.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, X.; Wang, S.; Wang, P.; Wu, D. Automatic Grading of Programming Assignments: An Approach Based on Formal Semantics. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), Montreal, QC, Canada, 25–31 May 2019; pp. 126–137. [CrossRef]
2. Pieterse, V. Automated Assessment of Programming Assignments. In Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research, Heerlen, The Netherlands, 4–5 April 2013; pp. 45–56.
3. Souza, D.M.; Felizardo, K.R.; Barbosa, E.F. A Systematic Literature Review of Assessment Tools for Programming Assignments. In Proceedings of the 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET), Dallas, TX, USA, 6–8 April 2016; pp. 147–156. [CrossRef]
4. Marin, V.J.; Pereira, T.; Sridharan, S.; Rivero, C.R. Automated Personalized Feedback in Introductory Java Programming MOOCs. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 1259–1270. [CrossRef]
5. Keuning, H.; Jeuring, J.; Heeren, B. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, New York, NY, USA, 11–13 July 2016; pp. 41–46. [CrossRef]
6. Ihantola, P.; Ahoniemi, T.; Karavirta, V.; Seppälä, O. Review of Recent Systems for Automatic Assessment of Programming Assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research, New York, NY, USA, 28–31 October 2010; pp. 86–93. [CrossRef]
7. Caiza, J.; Del Alamo, J. Programming assignments automatic grading: Review of tools and implementations. In Proceedings of the 7th International Technology, Education and Development Conference, Valencia, Spain, 4–5 March 2013; pp. 5691–5700.
8. Messer, M.; Brown, N.C.C.; Kölling, M.; Shi, M. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Trans. Comput. Educ.* **2023**. [CrossRef]
9. Delgado-Pérez, P.; Medina-Bulo, I. Customizable and scalable automated assessment of C/C++ programming assignments. *Comput. Appl. Eng. Educ.* **2020**, *28*, 1449–1466. [CrossRef]
10. Fonte, D.; da Cruz, D.; Gançarski, A.L.; Henriques, P.R. A Flexible Dynamic System for Automatic Grading of Programming Exercises. In Proceedings of the 2nd Symposium on Languages, Applications and Technologies, Porto, Portugal, 20–21 June 2013; Leal, J.P., Rocha, R., Simões, A., Eds.; OpenAccess Series in Informatics (OASIcs); Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2013; Volume 29, pp. 129–144. [CrossRef]
11. Wang, T.; Santoso, D.B.; Wang, K.; Su, X.; Lv, Z. Automatic Grading for Complex Multifile Programs. *Complex* **2020**, 1–15. [CrossRef]
12. Ala-Mutka, K.M. A Survey of Automated Assessment Approaches for Programming Assignments. *Comput. Sci. Educ.* **2005**, *15*, 83–102. [CrossRef]
13. Singh, R.; Gulwani, S.; Solar-Lezama, A. Automated Feedback Generation for Introductory Programming Assignments. *SIGPLAN Not.* **2013**, *48*, 15–26. [CrossRef]
14. Staubitz, T.; Klement, H.; Renz, J.; Teusner, R.; Meinel, C. Towards practical programming exercises and automated assessment in Massive Open Online Courses. In Proceedings of the 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), Zuhai, China, 10–12 December 2015; pp. 23–30. [CrossRef]
15. Parihar, S.; Dadachanji, Z.; Singh, P.K.; Das, R.; Karkare, A.; Bhattacharya, A. Automatic Grading and Feedback Using Program Repair for Introductory Programming Courses. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, Bologna, Italy, 3–5 July 2017; pp. 92–97. [CrossRef]
16. Zhang, H.; Liu, J. Research Review of Design Pattern Mining. In Proceedings of the 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 16–18 October 2020; pp. 339–342. [CrossRef]
17. Zanoni, M.; Arcelli Fontana, F.; Stella, F. On applying machine learning techniques for design pattern detection. *J. Syst. Softw.* **2015**, *103*, 102–117. [CrossRef]
18. Dwivedi, A.K.; Tirkey, A.; Rath, S.K. Software design pattern mining using classification-based techniques. *Front. Comput. Sci.* **2018**, *12*, 908–922. [CrossRef]
19. Joy, M.; Luck, M. Plagiarism in programming assignments. *IEEE Trans. Educ.* **1999**, *42*, 129–133. [CrossRef]
20. Prechelt, L.; Malpohl, G.; Philippsen, M. Finding plagiarisms among a set of programs with JPlag. *J. Univers. Comput. Sci.* **2002**, *8*, 1016–1038. [CrossRef]
21. Schleimer, S.; Wilkerson, D.S.; Aiken, A. Winnowing: Local algorithms for document fingerprinting. In Proceedings of the 2003 ACM SIGMOD International Conference on on Management of Data, San Diego, CA, USA, 10–12 June 2003; pp. 76–85. [CrossRef]

22. Novak, M. *Effect of Source-Code Preprocessing Techniques on Plagiarism Detection Accuracy in Student Programming Assignments*; University of Zagreb Faculty of Organization and Informatics: Zagreb, Croatia, 2020.
23. Martins, V.T.; Henriques, P.R.; da Cruz, D. An AST-based Tool, Spector, for Plagiarism Detection: The Approach, Functionality, and Implementation. *Commun. Comput. Inf. Sci.* **2015**, *563*, 153–159. [CrossRef]
24. Kermek, D.; Novak, M. Process Model Improvement for Source Code Plagiarism Detection in Student Programming Assignments. *Inform. Educ.* **2016**, *15*, 103–126. [CrossRef]
25. Đurić, Z.; Gašević, D. A Source Code Similarity System for Plagiarism Detection. *Comput. J.* **2013**, *56*, 70–86. [CrossRef]
26. Novak, M.; Joy, M.; Kermek, D. Source-code similarity detection and detection tools used in academia: A systematic review. *ACM Trans. Comput. Educ.* **2019**, *19*, 1–37. [CrossRef]
27. English, J.; Rosenthal, T. Evaluating Students' Programs Using Automated Assessment: A Case Study. *SIGCSE Bull.* **2009**, *41*, 371. [CrossRef]
28. Cirillo, F. *The Pomodoro Technique*; Vrgin Books: London, UK, 2018.