Automation of assessment and feedback in IT teaching from the teaching staff perspective

Eerik Muuli
Institute of Computer Science
University of Tartu
Tartu, Estonia
eerik.muuli@ut.ee

Marina Lepp
Institute of Computer Science
University of Tartu
Tartu, Estonia
marina.lepp@ut.ee

Reimo Palm
Institute of Computer Science
University of Tartu
Tartu, Estonia
reimo.palm@ut.ee

Piret Luik
Institute of Computer Science
University of Tartu
Tartu, Estonia
piret.luik@ut.ee

***Abstract*—Information technology (IT) has been a popular specialty among high-school graduates for quite some time. As the number of entrants continues to grow year-by-year, the load on teaching staff also increases. Automation of teaching activities can be used to alleviate the high workload resulting from courses with hundreds of students. The aim of this study is to ascertain the processes that could be automated in the context of assessment and feedback. To gather ideas, eight one-hour mini-group interviews were conducted at the Institute of Computer Science, the University of Tartu, with 17 faculty members responsible for teaching IT courses. A qualitative approach was used for classifying the proposed ideas related to the automated assessment and automated feedback. Five main categories connected to automated assessment emerged from analyzing the codes generated based on the interviews using qualitative content analysis: use of automated assessment, instructor-related topics, negative effect on students, improvements of automated assessment, and technical concerns. Regarding automated feedback, three main categories emerged: feedback for teachers, feedback for students, and feedback for both teachers and students. The classification of ideas was used as an input for designing a schema displaying the current automated assessment system with all the potential additions suggested by the faculty members showing how, if and where exactly the proposed ideas would fit. We believe that the ideas and improvements related to automated assessment and feedback in this study, in conjunction with the detailed clarification of the process, can be a useful input for other institutes. The results will be used as a basis for implementing and improving the automation tools used hand in hand with teaching by the faculty members.**

Keywords—automated assessment, feedback, workplace, content analysis, interviews.

I. INTRODUCTION

The past two decades have seen ongoing growth of and interest in the Information Technology (IT) specialty. There is still an acute shortage of IT specialists on the market which puts them into a high demand and makes the specialty attractive for young people. It results in more and more enthusiasts from different backgrounds applying for a spot in IT courses. The University of Tartu accepts hundreds of IT students every year. Additionally, there is a noteworthy interest by students from other specialties that take part in IT courses. The combination of increasing interest in the specialty and more students taking the courses means a higher workload for the university staff as they need to perform different teaching activities in many courses.

Graded assignments are an important component of these courses and students usually solve a lot of such tasks [1]. Feedback and grades are a key reflection on students' learning and progress. Therefore, prompt and appropriate feedback is crucial for advancement in studies. There are many ways for students to receive help and feedback, such as meeting teachers face-to-face during the office hours, but this process in conclusion with feedbacking, grading and assessing can become repetitive and consume a lot of the teachers' time [2]. In order to scale these support mechanisms, an automated approach should be considered. Automated feedback generation can offer instant, zero-cost support for the students that are in need of the assistance [2].

Finding ways to automatically assess and feedback students' solutions of programming assignments has been researched for more than 50 years [3] but unfortunately, there still does not seem to be a uniform solution that all the teaching institutions could use. Researchers have built many internal and in-house automated assessment systems, but without much co-operation and standards [4]. When it comes to determining the requirements and needs of such systems, the expertise of teaching staff is usually of highest quality [5].

The purpose of this research was to identify the faculty's work processes related to automated assessment and feedback with opportunities for further improvement of automation. The research was based on the opinions of the teaching staff of the Chair of Programming Languages and Systems at the University of Tartu. The following research questions were addressed:

1. Which of the faculty work processes related to assessment could be automated according to the opinions of the teaching staff?

2. How does the faculty staff describe and characterize effective automated feedback?

II. LITERATURE REVIEW

A. Automated assessment

It can be useful to assess students based on their working (coding) habits along with assessing their coding solutions. Howles [6] is worried that many of the students do not think about the program's design before coding nor test their code with unit tests. She believes that incremental grading should be the way to go as it demands students to find, fix and document everything they do. Also, automation could be used there to manage different versions of the submissions for later analysis. Furthermore, instead of assessing the whole program at once and giving it either zero or maximum points, smaller components (such as functions or statements) could be assessed in order to

offer non-binary grading. For example, there have been multiple approaches [7, 8] to assess single functions and methods instead of the whole program by using reflection classes that enable invoking methods based on their signature. A system called ELP [9] even makes it possible to assess single calls and statements within a student's solution code. In addition to validating code compilation, function or statement calls, there are many other aspects of code that can be measured and assessed. Even when solutions give a correct answer as an output it does not mean that they are perfect and cannot be improved. For example, one way to measure the efficiency of a program is to measure how long the program runs [10]. Moreover, cyclomatic complexity can be measured on a student's solution code and afterwards compared with the instructor's one, and if the differences between the two are too large, there are most likely some problems with the student's submission [11]. Besides, some compilers are capable of providing information regarding language standards, features, implicit type conversions and unused variables. In order to automatically assess programming tasks, the task specifications and requirements must be well defined, but this often limits the potential of using the creative tasks. Muuli et al. [12] alleviated the problem and automatically assessed programming tasks with graphical output by creating a system that uses image recognition to detect images on the graphical output produced by the solution program.

Fighting plagiarism is an ongoing issue and remains so until a perfect solution is found or until the submissions of the programming assignments are not formulated as text files [4]. Various systems have been built for comparing the submissions. Some of the more common plagiarism checking services, such as MOSS and JPLAG, have structural information included in their methods. Certain special tools for managing and assessing computer programming exercises like VPL (Virtual Programming Lab) also include the feature of checking for plagiarism among submissions for a task in a course and other sources, like submissions for the same task in previous semesters, or similar tasks from other courses, which are a probable source of plagiarism [13].

Using automated assessment only to save time and simplify the assessing and grading process is a dangerous endeavor [4]. Teachers and instructors must remember that the learning process is diverse and does not only consist of grades and automated feedback. Therefore, incorporating new ways of automated assessment and feedback into courses must be well reasoned and justified. Ignoring the possible consequences can lead to unwanted results [14]. Even the smallest mistakes in the marking definitions or ambiguities in the assignment description can have a negative effect on the students' learning process. On the positive note, building automated assessment tools and systems forces teachers to justify every assignment and test case [4]. As the creation of a well-rounded automated assessment system is a complex tax, more co-operation would be advisable because many automated assessment systems have already been built but, regrettably, most of them are built in-house in different institutes to cater mostly for their own needs [4]. This means that there are no standards and common interfaces, which could be used to build, for example, approaches, assignment and assessment banks. Significant advancements in the field could be made if teachers would use more concepts like open source and would cooperate with different institutions.

It is important for students not to get lazy using automated assessment. Chen [15] and Edwards [16] believe that such systems should be built in a way that forces students to think through all the edge cases and test their submission thoroughly before submitting instead of just relying on the feedback provided by automated assessment. This can be achieved by using a suitable assessment strategy, such as limiting the number of submissions, or by forcing students to also provide the set of test cases used on their solution. The same idea was proposed by Thiébaut [17] after using VPL in his courses. Interestingly, there are also opinions against using automated assessment. Ruehr and Orr [18] analyzed different assessment criteria and concluded that interactive demonstration was the best assessment method for programming assignments as the personal contact guarantees individual approach on the student and the task at hand. They also mention that it might not be the best solution for large classes and that the ideal approach might actually be something in between automated assessment and manual assessment.

B. Automated feedback

Providing automatically generated feedback in introductory programming courses is one of the key challenges that research institutes are battling with as the number of computer science students is growing. There are two different stages when automated feedback can be provided for the student: (a) when the program is having difficulties during the compilation or execution process, (b) when the program compiles and executes, but the output does not match the expected one [19]. There are multiple ways to provide automated feedback and some of them are described in the following paragraphs.

The errors that happen during the compilation or execution of a program usually also provide an error message, but such messages are often too complex for beginner programmers. The authors of GradeIT [1] identified the most common compilation errors and provided a simplified version of these errors, and GradeIT shows these when the error occurs together with valid and invalid code fragments that explain to the student how to fix the error(s). GradeIT is also capable of repairing common compilation errors using simple rewrite rules to fix the errors thus making it possible to give some points to the student. Similarly, HelpMeOut [20] provides hints on compilation errors by showing the fixed erroneous line to the student. The feedback is based on a database of errors encountered by other students. TRACER [21] is comparing the erroneous version of a submission with a later error free version. Based on these corrections, the fixes are learned and later offered automatically.

Furthermore, there is a popular way to generate automated feedback by clustering submissions with similar features. The clusters can be composed either manually or by using automation, such as machine learning [22] or program analysis [23]. For each cluster, a separate feedback is added manually for a representative program and for every other program in the cluster the feedback is automatically modified. One of such tools is TipsC [19] that uses rule-based clustering, but it uses linearized ASTs for efficient computation of distance between two programs.

Course instructors could also benefit from automated feedback regarding their students. One of such data sources that contain valuable information and that could be analyzed is logs data. Often, logs data is analyzed to generate automated feedback for students, but it can also be used to provide feedback for the instructors. In [24], logs data was collected, predictive models were built and the results were displayed on an instructor dashboard that shows a timeline of the class, a general summary of the class with estimates of students' progress, a visual representation of all students, and a detailed report for a selected student. Such a system makes it easier for instructors to get a general overview of how the class is doing and find the low-performing students that otherwise might not seek out help.

III. METHODOLOGY

A. Context of study

The Chair of Programming Languages and Systems (CPLS) at the University of Tartu is responsible for teaching different courses, like Computer Programming, Programming Languages, Object-oriented Programming, Databases, Algorithms and Data Structures, Automata, Languages and Compilers, etc. Automation has been used in most of the courses with the help of Virtual Programming Lab (VPL) plug-in [13]. VPL allows to formulate programming assignments and specify tests for giving automatic feedback on solutions. VPL was also used in programming MOOCs organized by the CPLS [25]. Furthermore, the system based on VPL for automated assessment of programming tasks with graphical output was developed at the CPLS [12].

B. Data collection and sample

The interviewees were chosen from the Chair of Programming Languages and Systems as many of them had previous automation experience in the courses they had taught. Table 1 describes the participants of each interview by mentioning their position, gender (M = male, F = female) and age (second number in the parentheses).

TABLE I. THE PEOPLE THAT TOOK PART IN THE INTERVIEWS AND CONTRIBUTED TO THE RESEARCH.

Interview no.	Participants
1.	Teaching Assistant (F1, 33), Associate Professor (F2, 68)
2.	Teaching Assistant (M1, 34), Lecturer (M2, 60), Teaching Assistant (F1, 42)
3.	Lecturer (M1, 67), Teaching Assistant (F1, 35), Research fellow (M2, 34)
4.	Professor (M1, 51), Lecturer (M2, 45)
5.	Lecturer (F1, 33), Associate Professor (M1, 70)
6.	Associate Professor (F1, 52), Associate Professor (F2, 39)
7.	Lecturer (M1, 43)
8.	Teaching Assistant (M1, 26), Associate Professor (M2, 39)

The research questions were used as the base for constructing the interview questions which could be split into five key types provided by Krueger in [26]: opening, introductory, key, transition and ending questions. The respondents were asked about previous experience with automation, the processes that should be automated in teachers' and students' work, an imaginary perfect study process, and the future directions of automation. A more detailed description of the interview questions is given in our previous study [27].

The first of the eight interviews was a pilot interview with two participants, which lasted for 78 minutes. It was used for validating the sequence of questions and their wording, as well as the format of the interviews. The results of this research also include the pilot interview. Two people conducted all the interviews, the first of which had a role of the leader – making the introductions, asking the questions and concluding the interview, whereas the other person had a more supporting role – contributing to the discussion wherever needed, taking notes and making sure that the interview followed the predefined structure. The format and purpose of the interview were presented in the introduction followed by asking for consent to record the interviews. It was stated that the anonymity of the interviewees was guaranteed and that suggesting ideas does not oblige the interviewees to implement or carry them out. The interviews were recorded and lasted on average for 71 minutes. A manual and verbatim transcription was conducted by one of the authors after all the interviews were carried out. The transcriptions contained 29,962 words in total, with the shortest transcription consisted of 2,941 words and the longest of 5,737 words.

C. Data analysis

In order to get a detailed insight into the qualitative data, inductive content analysis was used to code the transcriptions based on the eight mini-group interviews carried out as part of the research. Inductive content analysis is used if the phenomenon has not been previously studied or the previous studies are inconclusive [28], as in this case. Following the suggestion of Elo and Kyngäs [28], the transcripts were first read several times to immerse oneself in the data, followed by the analysis process, which included open coding, creating categories, and abstraction. Data were analyzed by research questions. According to Mayring [29], such an approach allows for in-depth analysis of the data, as transcripts are repeatedly analyzed from different starting points (research questions). For the coding and forming of categories, a web software called QCAmap (https://www.qcamap.org/) was used. The meaningful unit was the text that conveyed an important overall meaning in the context of the research question [28].

The initial codes were formulated during repeated coding by the first author. To ensure internal coding consistency, the data were coded a second time after the first coding three months later to detect possible discrepancies or to notice new nuances in the coding and coding process, thereby increasing the reliability of the study. In order to find consistency between the researchers (to increase the reliability of the study), the other author of this paper performed an independent coding of the interviews. Any differences in the coding comparisons were discussed with the co-coder and a common understanding of the definition and coding of the meaningful items was reached.

There emerged 38 codes related to automated assessment (RQ1) and 28 codes related to automated feedback (RQ2). Based on the Elo and Kyngäs [28] the next step in qualitative inductive content analysis was categorization where codes were grouped into subcategories and categories based on content

similarity. This process was carried on by three authors of this study (coder, co-coder, and the second author).

IV. Results and discussion

The results of the study are presented below, broken down by categories formed on the basis of the research questions. To illustrate the results, quotations from interview transcripts are included in the text. The parentheses after each quotation indicate the interview and the participant whose quote is presented.

A. Results regarding automated assessment

Five main categories emerged from analyzing the codes generated based on the interviews using qualitative content analysis: use of automated assessment, instructor-related topics, negative effect on students, improvements of automated assessment, and technical concerns. The first mentioned category also includes two subcategories.

1) Use of automated assessment

a) Cases for use

It was suggested that automated assessment should be used only in the easier topics and for beginner courses. It was recommended that automation should not be used with more complex topics that require deep focus. One of the viewpoints was that automated assessment should not even be used at all at universities. Ruehr and Orr [18] agree that the best approach is face-to-face interactive demonstration but do not go as far as saying that automation should not be used at all.

> *But there should be no automated checks in the courses taught at a university. The emphasis at a university should be for students to reflect more on their actions. (7.M1)*

The interviewees recommended using automated assessment in MOOCs that have a lot of participants as automated assessment facilitates scalability. A similar aspect has been pointed out previously [18]. Furthermore, the consensus seemed to be somewhere in the middle based on our interviews, meaning that there should be both automated assessment and also some human involvement. For example, it was recommended to use automated assessment for individual tasks that students perform at home for instant feedback. Interviewees suggested that the key is finding the right balance between automation and human efforts.

> *This automated check does help, but the human dimension is crucial. But the truth is likely somewhere in between. (2.M1)*

b) Task-related topics

According to the interviewees, it is a complicated challenge to automate practical tasks, such as automatically assessing students' homework, lab exercises or tests in which each task depends on a previous task. Another viewpoint was that we can automatically assess only fact-based knowledge, but not the problem-solving process itself and this is exactly the thing that we need to assess. Previously, Howles [6] has recommended grading solutions incrementally in order to capture all the steps that students perform, including designing, testing, and documenting.

> *When it comes to databases then each task depends on the earlier state of the database. The student's database needs to be launched in each instance to see whether the state of the database is as it should be after the correct solution of the task. This makes it very complicated to provide feedback on what actually happened there. (3.M1)*

Automatically assessing creative tasks also seems to be a difficult problem in the opinion of the interviewees, but still possible as for example Muuli et al. [12] provided a way to automatically assess programming tasks with graphical output using image recognition. The interviewees said that tasks such as creating screen recordings, writing essays or texts that consist of discussion are necessary, and students also like to write. Unfortunately, all tasks of this kind take a considerable amount of time to assess manually, but automation does not seem to provide a solution, either.

> *Furthermore, we have a task where screen videos need to be created. While each of them lasts only a couple of minutes, it takes a lot of time to review all of them in detail. If this process could be automated somehow, it would certainly be convenient. (6.F1)*

It was quite interesting that some interviewees thought that a specific task's solution could be submitted only once for automated assessment as it would encourage students to delve deeper into the assignment. The same has also been proposed previously by Thiébaut [17]. On the other hand, others thought that there should definitely be multiple tries allowed in order to improve the solution based on the feedback.

2) Instructor-related topics

Unfortunately, using automated assessment currently comes with a price. More specifically, interviews revealed that creating and maintaining automated assessments is currently very difficult, takes a lot of time and only a few people within our Chair are capable of doing it. The current system is fragile and not documented. Many interviewees believe that creating an interface or simplifying the process of creating a new automated assessment is of critical importance for the Chair. The issue has been documented in other institutions as well and there are many competent researchers building their automated assessment systems but unfortunately, most of them are meant for internal use, which results in a lack of standards and co-operation [4].

> *In addition, my thinking is that creation of automated checks could be simplified or perhaps improved somehow. So that it would be easy to create them and that I would not need to contact someone else whenever I need to change them. (2.F1)*

Some students tend to submit their solution dozens of times and teachers believe it could be a warning sign that the student needs help. An interesting suggestion was that whenever a student submits a solution for a specific task more than X times, the teacher would receive a notification regarding the task and student. From there on the teacher could decide whether to assist the single student or go over the topic once again in the lecture or lab. Furthermore, currently there seems to be no place for teachers to get a general overview on how each student is doing. Such an automation was suggested that would brief teachers

regarding tasks and activities that the students have or have not performed.

> *There could be automation showing which students have not completed which tasks; then I could send an email, say, a couple of weeks after the start of the course, asking them how they are doing, what are they currently working on, do they want to stay on the course or withdraw, and is there anything I could help them with? (1.F1)*

It seems to be a consensus among interviewees that using automated assessment saves teachers time in general and therefore frees up more time to improve other aspects of teaching. Although there are still some trust issues among teachers regarding automated assessment and from time to time they make manually random checks on random assignments. And in case a student's solution receives a negative result, some interviewees said that they still often manually go over such submissions in order to make sure that those students do not indeed deserve any points for the specific task in hand. Ala-Mutka et al. [14] have also recognized that using automated assessment forces teachers to analyze and reason every assignment and test case.

> *Even when the test shows that everything is ok, I personally tend to make some random double checks. Did it really work how it was supposed to, did the student perhaps not invent something, which the automated check was unable to detect? (3.F1)*

3) Negative effect on students

It was mentioned in the interviews that it is important to think about the possible negative aspects of automated assessments while constructing them. Losing human contact seems to be an issue because sooner or later students will have questions that automated assessment cannot answer. Also, the interviewees said that students sometimes prefer to send solutions directly to the teacher instead of using automated assessment, because they know their code is incomplete, and the automated assessment system would not accept it. In addition, if students do not get a chance to have contact with teachers, their communication skills will not improve and might even diminish.

> *In programming, it is also not ideal if there is no contact at all. It is good that they can solve assignments at home, but there are also practical training labs where they can reach out and ask questions. (2.F1)*

The downside of the automated assessment, based on the opinion of the interviewees, is that students might start focusing too much on the feedback given by automated assessment and therefore, their knowledge regarding the topic can remain superficial. For example, the interviewees said that automated assessment can present students all the missing edge cases in a way that students would not even have to think about them. It can also make students more impatient and focus on passing the automated tests rather than trying to understand the key concepts in the current assignment. Chen [15] and Edwards [16] have suggested tackling this issue by using a suitable assessment strategy that would force students to thoroughly focus on the assignment, for example, by making them upload the set of test cases used to validate the submission.

> *However, a potential risk here is that if we give too many hints, the student will not need to think at all. If we tell them outright that they are missing x and y, they may simply implement those conditions without giving it any thought. Consequently, they may lose the ability of independent analysis. (6.F1)*

It turned out from the interviews that it also depends on the student whether automated assessment helps to improve the submitted solution or not. Some students are capable of analyzing the feedback provided by automated assessment and making amendments to their solution. On the other hand, some students get seriously disturbed by automated assessment, for example, when they solve the assignment in a different way than automated assessment expects or when they have minor mistakes in their solution, but they know that automated assessment would give them zero points for that.

> *In general, students are very apprehensive about the use of automated checks in tests or exams, because they often make some minor mistakes and they know that the instructor would not give them zero points for that but automated assessment might. (6.F2)*

4) Improvements of automated assessment

Despite the fact that some systems can perform partial grading of solutions [7, 8, 9], the automated assessment system used in our Chair can currently assess the submitted solutions as either failed or succeeded for most of the cases according to the interviewees. This means that the assessment process is strict and binary. Many interviewees recommended assessing different parts of a task in isolation and in multiple steps. This would allow non-binary grading and would also make students' performance more transparent.

> *Another helpful option could be, perhaps, splitting assignments into smaller parts. This would give us timely information not only on a single programming assignment but also on a specific part of the assignment. (5.M1)*

The interviewees stated that students tend to use, from time to time, elements of code that are not taught in the specific subject. This often leads teachers to think about possible resources that may have enabled students to come up with such solutions and, of course, there is the fear that students took the solution from the internet or it was made by someone else. The interviewees conjectured that a tool that would automatically notify teachers when a submitted solution is using elements of code taught within the current subject would make their lives easier and would help discover fraud, but also give ideas on how to improve instruction in the subject. Similarly, the interviewees thought that it would be of great help if submissions would be automatically compared against each other to spot plagiarism and a report would be generated and made available for teachers regarding specific tasks. For example, [4] recommends using common tools. such as MOSS and JPLAG for fighting plagiarism. In addition, the interviewees mentioned that it would be interesting to automatically detect which of the learning outcomes did the student accomplish and which not.

> *That they would compare the solutions to those they have learned from before. Our students currently use a lot of code that they have learned earlier or have found on the internet.*

It could be useful to highlight the spots where the code that was used has not been taught in our course. (2.M2)

A more dynamic way of creating assignments was also suggested by the interviewees. Specifically, every time a student opens the task at hand, its input parameters would be automatically generated and would therefore be unique for every student. The assessment would not become more complex, but the tasks themselves would become more dynamic.

It is important to design tests in a way that a test with a new input is generated every time, to enable them to take it again without always having the same correct answer to the same question. (7.M1)

Logs of student solutions can provide a lot of valuable information in the opinion of the interviewees. Such logs could be used, for example, to automatically detect whether the submitted solution matches the solving log provided by the student or not. In addition, logs provide valuable information regarding how many times and what did a student copy and paste, and where exactly the students got stuck in each assignment. A previous study [24] has also used logs to provide insights for the instructors.

Speaking of logs, there could be a possibility of an automated verification whether the appended log is actually related to this particular solution or whether there is a major case of copying to be detected. (2.F1)

Using automated assessment efficiently requires efficient integration with different systems. The interviewees reckoned that there are currently several systems that are not integrated with any other study system and therefore are isolated and more difficult to use. Integration between different systems would allow for more efficient automation and would create opportunities for further developments, such as automated log synchronization and plagiarism check.

In my opinion, Thonny's integration with Moodle. For instance, for automated synchronization of logs from there, an automated analysis of the logs and a plagiarism check. (2.M1)

5) Technical concerns

The interviewees were worried that automated assessment is often tightly connected with its corresponding assignment. Sometimes, this makes the assignment depend on automated assessment and imposes some limitations on the task. Also, in this case, when we want to make only slight modifications to an assignment, it means that automated assessment needs an update as well. Furthermore, there are cases when an assignment can have multiple correct solutions, but automated assessment oftentimes accepts only one specific solution. For example, a programming task could expect a dictionary as an answer, but the order of keys should not be vital. Also, in some cases, automated assessment requires files with specific names and gives the submission zero points if there are any deviations.

It would be nice if automated checks could accept a range of different solutions, because students tend to have different visions and ideas about how a particular task should be solved. At the moment, there is, in principle, only one course of solution envisaged for each task. (6.F2)

One of the worst cases mentioned by the interviewees is, of course, when automated assessment accepts a faulty submission or when students somehow manage to trick the system by submitting wrong solutions and still receive points for it.

With automated checks, it sometimes happens that they need some kind of fix, and a resourceful student is able to make a type of mistake that is not expected in automated assessment. (3.M1)

B. Results regarding automated feedback

Regarding automated feedback, three main categories emerged from analyzing the codes generated based on the interviews using qualitative content analysis: feedback for teachers, feedback for students, and feedback for both teachers and students. The category of feedback for students also included three subcategories.

1) Feedback for teachers

Teachers can have dozens or even hundreds of students in a semester. The interviews revealed that it is difficult to keep an eye on the progress of each and every one of them, and to see if they are struggling and submit solutions several times. Therefore, if students themselves do not seek help they can fall behind and could even fail the course in the end. The interviewees suggested that some sort of automated feedback could be provided for instructors regarding the progress that students are making during the semester and also regarding detected plagiarism issues. From the point of view of lecturers in charge, the issue is similar as they do not have any overview of how all the lab groups and instructors are doing.

Exactly! Because at the moment there is no overview. We cannot see their progress anywhere. As there are so many of them, it would be entirely impossible to do it one by one. Where there are 20-30 in a course, it would be possible, but it cannot be done with 300 students. It is only at the end of semester that you'll find out how they were doing. (1.F2)

While some previous studies (e.g., [24]) have already dealt with identifying the students that need extra attention, automated student classification was still brought up in many interviews. Finding automatically the stronger and weaker students would help to make specific groups for them. Also, it would help teachers to pay more attention to these students and potentially give specific tasks to help them get on track. It would also be useful to detect the exact task or topic in which the students fall behind.

Yes, if it would be possible, for instance, to group the students – who are weaker and who are stronger. Who should be given which tasks? (7.M1)

As suggested by the interviews, an automated assessment system could potentially group similar submissions or snippets of code and there could be a database which consists of the feedback given on specific submissions. Using such a solution, the system could provide automated feedback on students' solutions. Some systems [19] are already capable of doing so.

The system would, for instance, group similar presentations or pieces of code. And there would be a database of feedback we have provided. Telling with

machine learning methods: someone has said this about such a solution/presentation. (8.M2)

2) Feedback for students

a) Evaluation

The feedback regarding points given for a solution is often provided in a binary format – correct or incorrect. Based on our study, such feedback is not very useful for students because a solution can be partially correct and would receive at least some points if a teacher would grade it. Many interviewees suggested that non-binary automated feedback would be preferable, with partially correct solutions receiving points for the parts of the assignment that were correct.

And also, that it would not simply pass or fail. That there would be points in between, for instance, 0.5 or 0.7. (6.F2)

For example, in a Java course, students automatically receive zero points for their solution if one of the files does not compile, although the content (at least some of it) in this and other files can be correct. A similar problem has also been discussed and solved in connection with the TRACER system [21]. The interviewees also noted that the current system rewards students with zero points in case of faulty file or function names or incorrect capitalization. Furthermore, the interviews revealed that automated assessment used in our university usually accepts one specific solution, although at times the exercise could be solved in multiple ways and this can cause issues. such as misleading feedback given to a student.

b) Human vs. machine feedback

Teachers usually tend to give feedback on more aspects than the answer alone. It was suggested in the interviews that automated feedback could also be given on 'correct' solutions as there are many more things that could be analyzed, such as code complexity, coding style, unnecessary lines, variable names, etc. Code complexity, for example, could be analyzed by measuring cyclomatic complexity [11] and code efficiency based on program execution time [10]. The interviewees said that, in such a way, students could also improve other parts of their programming skill.

In addition, the automated check currently only checks the output, but it could also check the content. Because, when we as instructors do the manual checks, we are looking at many other aspects as well. (6.F2)

It was suggested by the interviewees that, if feasible, feedback should also be as similar as possible to the natural language as teachers would provide it. For example, the level of detail in error messages should be maximized to provide students with the most detailed information possible to enable them to improve their solution and understand what and why is wrong with their solution. The interviewees said that, in many cases, students do not even understand why automated assessment failed and what was wrong with their solution. The previously developed system GradeIT [1] makes it as easy as possible for students to understand the error by providing examples of faulty and correct code snippets of the given syntax error with a modified description.

But the more intelligent the feedback systems we created, the better the feedback provided. When we look at the thousands of solutions and the feedback given in response to them, we can hopefully find a way to use them to synthesize feedback, based on machine learning. (4.M1)

c) Improved feedback

Instant feedback is crucial when doing programming exercises, especially when students are working alone, e.g., when they are doing their homework. The interviewees mentioned that the error messages themselves can sometimes be misleading for students and can even put them off performing the tasks. Therefore, in the opinion of the interviewed staff, the goal of automated feedback should be to help students to find the errors in their solutions and solve their issues. Many interviewees believe that it would be of great help if automated feedback could pinpoint the issue in their submission and could also provide hints on what to try next or how to fix the errors in the solution, while not solving the task for students. The recently introduced system GradeIT [1] pinpoints students to their error while also mapping the syntax error into a student-friendly message with examples on how to fix it. Based on the interviews, the worst-case scenario is when automated assessment runs into an internal error and, for example, just hangs. In this case, the student does not get any feedback whatsoever, irrespective of whether their solution is correct or incorrect, causing even more confusion for the student. An interesting idea suggested was for students to see how much time they spend on solving any exercises in comparison to others, but it might be better to use this approach with Master's level students.

Ideally, feedback would include hints that would make them think and improve on their solution, if not too much is spelled out. (6.F1)

3) Feedback for students and teachers

Both students and teachers would benefit from the feedback generated by the automated assessment system also showing the exact line and character of the fault location. It was even suggested by the interviewees that the system could paint these spots with different colors so that teachers could more easily give feedback and that students could have easier measures to improve their code. A previous study [20] has dealt with a similar problem.

Additionally, the output of automated checks could highlight certain spots of error in red, because sometimes the answer needs to have an uppercase letter, while students write in lowercase, and it is difficult to spot the mistake. But if the automated check would highlight it in red, for instance, they could perhaps find it faster. (6.F2)

Many interviewees mentioned the fact that it is very difficult to find cumulative feedback given to a specific student. Such information would be useful for finding if and how a student has used or acknowledged the feedback given previously. This kind of feedback should also be available to the student, so that they could follow along the progress made.

It would be nice to have this feedback at least within one subject course; for instance, if a student received such a feedback on their first assignment and if I then checked their response to a tenth assignment, I would like to see some progress or regress. It would help me understand if I need to do some

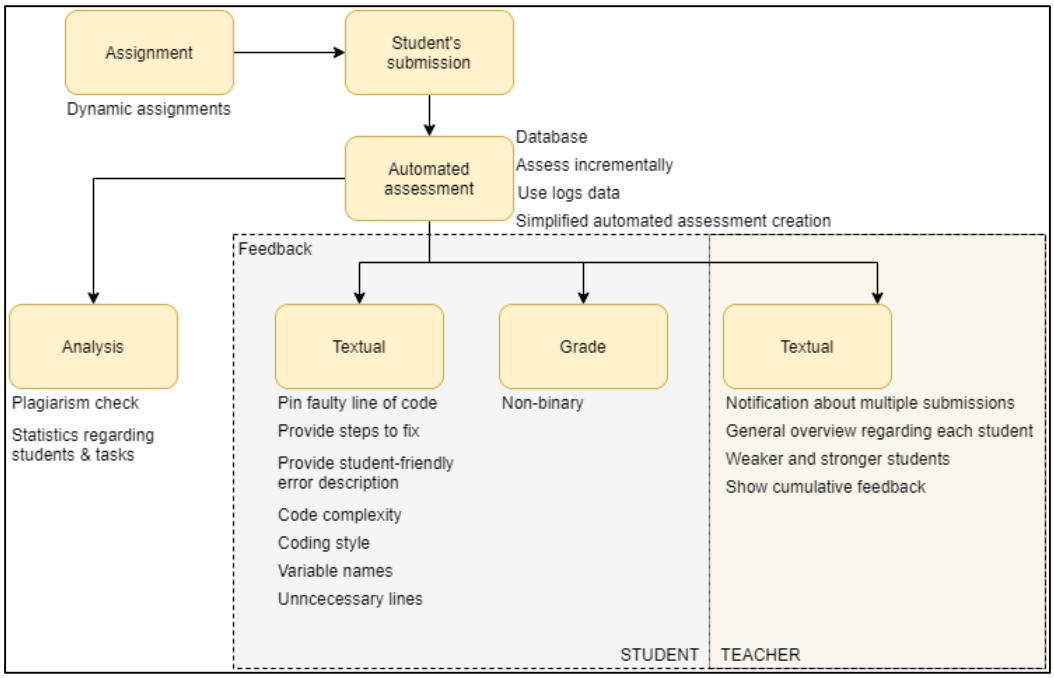

Fig. 1. The current automated assessment schema with its possible additions and improvements.

additional work with the student. It would be nice to have a history somewhere close to the task. (3.F1)

The interviewees said that the automated assessment and feedback result is often received only after submitting the solution. Constant feedback is a key concept for ensuring a smooth learning process.

As mentioned by the interviewees, in an ideal world, there would be one lab instructor per student who would constantly follow their progress, but that is not realistic. One way to alleviate the circumstances is to analyze, and potentially provide automatic feedback during, the solving process. This would reduce the number of instances when students are getting stuck, while enabling teachers to detect the tasks that students are struggling with in real time. An idea proposed earlier was to use logs data for that [24]. Also, it was mentioned in the interviews that students and teachers could potentially see different feedback and one way to implement it would be to automatically execute one set of automated tests after a student submits the solution and another set of tests if the task deadline has passed.

> *It would also be interesting to see the current mistakes that students make while they are solving the tasks. At the moment, when they submit their solution to automated assessment, it is presumably a corrected code without errors. Perhaps it would be possible to create an environment where students can solve the tasks in Moodle or on another platform that can provide instant feedback whenever requested. (2.M1)*

C. Recommendations for improving the current system

The aforementioned ideas were gathered and a schema was drawn (see Fig. 1.) with the current automated assessment system with all the possible improvements and additions provided by the teaching staff during the interviews (the improvements are added next to the rectangle boxes).

V. Conclusion

The findings of this study contribute to identifying the faculty work processes that are potentially suitable for automated assessment and feedback in a more efficient and superior way. Eight mini-group interviews were conducted to capture the teaching staff's opinions on the topic. The ideas that emerged from the interviews were set out on a schema that displays the previous automated assessment system together with the ideas proposed.

It is difficult to generalize the results as the research was carried out in a single department chair using a relatively small sample of participants. Despite that, we believe that the results of this research are still of great value for other research institutions and could be used as a basis for further research on the proposed and discussed topics in conjunction with the detailed descriptions of the processes used in our study. In order to diversify and enrich the results, the opinion of students could also be taken into account and researched further. The interviews revealed that there are many great ideas and many of them have already been implemented by different research institutions, but unfortunately most of them have been built in-house and not enough collaboration takes place between universities. The way forward should be to implement new systems in collaboration and to use open standards determined in concert.

This research will be used as a starting point for conducting further research on the discussed topics and building the suggested automation tools. The further research and implementation requires resources and people. Improving and simplifying the creation and maintenance of automated assessment was one of the major issues and further research will be conducted on the aforementioned topic.

REFERENCES

[1] S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare, and A. Bhattacharya, "Automatic grading and feedback using program repair for introductory programming courses," In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, 2017, pp. 92-97.

[2] P. M. Phothilimthana, and S. Sridhara, "High-coverage hint generation for massive courses: Do automated hints help CS1 students?" In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, 2017, pp. 182-187.

[3] J. Hollingsworth, "Automatic graders for programming classes," Communications of the ACM, vol. 3, no. 10, pp. 528-529, 1960.

[4] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," Computer Science Education, vol. 15, no. 2, pp. 83-102, 2005.

[5] M. Jamlan, "Faculty opinions towards introducing e-learning at the University of Bahrain," The International Review of Research in Open and Distributed Learning, vol. 5, no. 2, pp. 1-14, 2004.

[6] T. Howles, "Fostering the growth of a software quality culture," ACM SIGCSE Bulletin, vol. 35, no. 2, pp. 45-47, 2003.

[7] C. C. Ellsworth, J. B. Fenwick Jr, and B. L. Kurtz, "The quiver system," In Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, 2004, pp. 205-209.

[8] L. Bettini, P. Crescenzi, G. Innocenti, and M. Loreti, "An environment for self-assessing Java programming skills in first programming courses," In IEEE International Conference on Advanced Learning Technologies, 2004, pp. 161-165.

[9] N. Truong, P. Bancroft, and P. Roe, "A web based environment for learning to program," In Proceedings of the 26th Australasian Computer Science Conference - Volume 16, 2003, pp. 255-264.

[10] B. Cheang, A. Kurnia, A. Lim, and W. C. Oon, "On automated grading of programming assignments in an academic institution," Computers & Education, vol. 41, no. 2, pp. 121-131, 2003.

[11] D. Jackson, and M. Usher, "Grading student programs using ASSYST," In Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education, 1997, pp. 335-339.

[12] E. Muuli, E. Tõnisson, M. Lepp, P. Luik, T. Palts, R. Suviste, K. Papli, and M. Säde, "Using image recognition to automatically assess programming tasks with graphical output," Education and Information Technologies, vol. 25, pp. 5185-5203, 2020.

[13] J. C. Rodríguez-del-Pino, E. Rubio-Royo, and Z. J. Hernández-Figueroa, "A virtual programming lab for Moodle with automatic assessment and anti-plagiarism features," In Proceedings of the 2012 International Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government, 2012.

[14] K. Ala-Mutka, T. Uimonen, and H. M. Järvinen, "Supporting students in C++ programming courses with automatic program style assessment," Journal of Information Technology Education: Research, vol. 3, no. 1, pp. 245-262, 2004.

[15] P. M. Chen, "An automated feedback system for computer organization projects," IEEE Transactions on Education, vol. 47, no. 2, pp. 232-240, 2004.

[16] S. H. Edwards, "Improving student performance by evaluating how well students test their own programs," Journal on Educational Resources in Computing (JERIC), vol. 3, no. 3, pp. 1-24, 2003.

[17] D. Thiébaut, "Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in," Journal of Computing Sciences in Colleges, vol. 30, no. 6, pp. 145-151, 2015.

[18] F. Ruehr, and G. Orr, "Interactive program demonstration as a form of student program assessment," Journal of Computing Sciences in Colleges, vol. 18, no. 2, pp. 65-78, 2002.

[19] S. Sharma, P. Agarwal, P. Mor, and A. Karkare, "TipsC: Tips and corrections for programming MOOCs," In International Conference on Artificial Intelligence in Education, 2018, pp. 322-326.

[20] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do: suggesting solutions to error messages," In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2010, pp. 1019-1028.

[21] U. Z. Ahmed, P. Kumar, A. Karkare, P. Kar, and S. Gulwani, "Compilation error repair: for the student programs, from the student programs," In Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, 2018, pp. 78-87.

[22] C. Piech, J. Huang, A. Nguyen, M. Phulsuksombati, M. Sahami, and L. Guibas, "Learning program embeddings to propagate feedback on student code," In International Conference on Machine Learning, 2015, pp. 1093-1102.

[23] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni, and B. Hartmann, "Writing reusable code feedback at scale with mixed-initiative program synthesis," In Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale, 2017, pp. 89-98.

[24] N. Diana, M. Eagle, J. Stamper, S. Grover, M. Bienkowski, and S. Basu, "An instructor dashboard for real-time analytics in interactive programming assignments," In Proceedings of the Seventh International Learning Analytics & Knowledge Conference, 2017, pp. 272-279.

[25] M. Lepp, P. Luik, T. Palts, K. Papli, R. Suviste, M. Säde, and E. Tõnisson, "MOOC in programming: A success story," In Proceedings of the International Conference on e-Learning, 2017, pp. 138-147.

[26] R. Krueger, Focus Groups: A Practical Guide for Applied Research (2nd ed.). Thousand Oaks, CA: Sage, 1994.

[27] E. Muuli, E. Tõnisson, M. Lepp, R. Palm, and P. Luik, "Automation of IT faculty work from the teaching staff perspective," In Proceedings of the 15th International Technology, Education and Development Conference, 2021, pp. 2839-2848.

[28] S. Elo, and H. Kyngäs, "The qualitative content analysis process," Journal of Advanced Nursing, vol. 62, no. 1, pp. 107-115, 2008.

[29] P. Mayring, "Qualitative content analysis," A Companion to Qualitative Research, vol. 1, no. 2, pp. 159-176, 2004.