Tools to assist large scale introductory programming courses

Roope Luukkainen[1], Rami Saarivuori[1], Jesse Peltola[1], Uolevi Nikula[1] and Jussi Kasurinen[1]

[1]*LUT University, Yliopistonkatu 34, 53850 Lappeenranta, Finland*

Abstract

The first programming course (CS1) at LUT University passed the 1000 student mark in the fall 2022. The continuously increasing number of students attending the course introduces new challenges for the course arrangements. The core of our course is a course specific programming guide, which is updated regularly to keep up with the course contents. For the past few years, we have been developing a style guide that defines the programming style to use in the course project, and in fall 2022 a new autograder was taken in use in all the LUT programming courses. We have also concluded that if we want to keep improving the course and how it is run, we need to develop ourselves small tools to support the course activities. In this paper we summarize three tools that we have developed over the past few years. The first tool is ASPA, which is used to detect style guide violations. Second, we have GradeTool to help in grading student projects and exams. And third, we have a tool, Mímir, to help manage the increasing number of programming assignments and their variations, and to generate assignment documents automatically. All these tools have the same basic goals to reduce the amount of manual work and time required by the tasks, at the same time they increase quality by producing more consistent outputs in a systematic way. All these tools are still in the development phase, but we have started their empirical testing. The initial feedback from both students and teaching assistants using the tools or their outputs suggests that these tools can speed up doing the tasks and improve the quality of the end products. Thus, we plan to continue their development and explore expanding their adoption.

Keywords

CS1, Programming education, Interoperable technology, Static analysis, AST, Abstact syntax tree, Generated feedback, Assignment management, Grading, Instructions generation, Tool assisted

1. Introduction

The first programming course (CS1) at LUT University hit the low point in 2010 when only 146 students registered for the course. A few years earlier in 2004 the course had 441 registered students, but after university level administrative changes the interest in the course dropped. Even after a major revision of the course in 2006, the interest in the course continued to fall. Only in 2011 the interest increased for the first time in seven years and the registration count reached 206 students. The interest has continued to increase thereafter, and in fall 2022 the course was offered for 1114 students both in Finnish and in English. The course changes between 2005 and 2009 have been reported in detail in [1], and the improvement work has continued ever since. In this paper we look at the main ongoing course development topics.

The main LUT CS1 course is given in Finnish and it is based on the Python programming language [2]. The course covers basic programming topics like variables, I/O, branching, loops, file I/O, lists, classes, and error handling. The course includes 3-5 weekly programming assignments, in total 60 assignments, a course project, and an exam that is done in an electronic exam environment using the same tools as all other programming assignments. The study materials include lecture slides, a programming guide developed for the course [3], programming videos, and students are also offered exercise sessions both in computer classrooms and online.

With increasing interest in CS1, we developed two smaller variants of it. First, an online-variant of the course was developed as a part of FITech ICT -project, which aimed at offering university level education for all the Finns [4]. This was done by dropping the course project and exam, and focusing on the 60 programming assignments. The outcome was a 3 ects course that can be completed fully online, and since we are using an autograder for assessing the assignments, the effort required to manage this course is feasible even if circa 200 students complete it annually. Second, an English variant of the course was introduced in the fall 2021 to serve new degree programs for English speaking students [2]. In the fall 2022 the English course had 393 registered students while the Finnish one had 721.

The increasing number of students taking the course introduces new challenges for the course arrangements. Fundamentally the continuously growing number of students suggests that it is possible to study and learn the course contents with the existing materials and arrangements. However, not all the students are computer science majors, but the number of different backgrounds is also increasing rapidly. For example, the English degree programs are open to students from different countries and cultures, and the online course is open to any Finn who can complete the strong identification required by

TKTP '23: Annual Symposium for Computer Science 2023, June 13–14, 2023, Oulu, Finland

roope.luukkainen@lut.fi (R. Luukkainen); rami.saarivuori@lut.fi (R. Saarivuori); jesse.peltola@lut.fi (J. Peltola); uolevi.nikula@lut.fi (U. Nikula); jussi.kasurinen@lut.fi (J. Kasurinen)

the university. Thus the need to support different kind of learners increases as students and their different backgrounds increase, and we also need to be able to handle large numbers of all kinds of study related events and activities.

In this paper we describe our ongoing tool development actions on the CS1 course. These actions started in the Finnish CS1 context, but all the tools can be adapted to suit for different courses and, for example, GradeTool and Mímir were used in our C-programming course in spring 2023. In particular, the tools have been designed so that they can be adapted to other programming courses independent of the programming language used, and serve courses offered both in English and in Finnish. We are currently developing 3 tools:

1. **ASPA**. ASPA is a static analyzer that can be used to analyze the student programs and especially programming style problems in them.
2. **GradeTool**. Evaluating a large number of projects is one thing, and another thing is to provide students feedback that is consistently presented for all the students and detailed enough so that students can understand their errors and fix them. The exams need to be graded, too, even if now students do not fix the errors but should learn how to avoid them in the next exam. The key requirements for the tool are to speed up the grading, and provide more systematic and helpful feedback to students.
3. **Mímir**. Mímir is a tool to manage weekly assignments and their variants. Using the same weekly assignments multiple times increases the likelihood that students return a solution from a friend without doing it him/herself. Thus we are developing a tool to support management of multiple variants of the assignments, and automatic generation of systematic weekly assignment documents with minimal effort.

In the rest of this paper we shall describe each of tools in more detail, and look at the following three aspects:

- research questions and motivation
- tool developed
- empirical observations and data.

The paper is closed with a summary of the current and future interoperability of these development actions.

2. ASPA - Tool to provide feedback on program structure

In the fall 2022 students submitted over 92 000 programs for evaluation in our Finnish CS1 course. Consequently we do not evaluate all these programs manually but use an autograder to execute the programs and check that they work as instructed. As of now most student programs pass the autograder tests, but in manual review their structure is not always understandable, which makes both grading and code maintenance difficult. Thus we have moved our focus from a working program to understandable code, which we did by introducing a style guide. Since many software companies are using style guides today, we decided to use it in our course to show students how to write understandable code. One key reason to introduce the style guide was the use of global variables that is syntactically acceptable in typical programming languages, but from the good programming style point of view it has been deemed not acceptable since 1970's. That is, even if a compiler/interpreter accepts global variables, in our CS1 it is forbidden to use them by the course style guide.

Introduction of a style guide in our CS1 course caused problems since many students aimed at a working program and once it passed the autograder, no second thought was given to it. Thus requiring that programs follow a style guide raised a lot of objection, and to reduce this we introduced a tool to find style guide violations. First we developed an abstract syntax tree (AST) analyzer in 2020 and called it ASPA. As of now the staff uses ASPA to check to student projects and exams, and we also give it to the students so they can check their programs with it, too. Second, we adopted a new autograder in fall 2022 and we are currently using CodeGrade [5] which has a semgrep-feature supporting structural style guide checks. Including the style guide checks as a part of autograder and rejecting submissions that do not pass them makes it easier for students to understand that the style guide needs to be followed. In this paper we describe our ASPA-tool in short. The tool was developed for the first programming course with Python starting with the following research questions:

RQ1: Can static analyzer be used to assist students with programming assignments?

RQ2: Does a usage of static analyzer correlate with grades?

2.1. Literature review

AST-based static analysers are interesting from the programming education point of view, and many such systems have been developed. For example, PyTA is a wrapper module for Pylint [6], PEDAL is a feedback system for Python 3 programs [7], and AutoStyle is a style tutor providing hints for student [8, 9]. On the other hand, Semgrep is an open-source based extended AST solution that detects patterns in source code. It supports over 30 programming languages, as well as generic option for any text-based search [10]. And finally, there are education platforms such as ViLLE [11] and A+ [12] which can do style checks in the submission phase.

2.2. Developed solution

ASPA uses AST for structural pattern checks to detect, for example, whether variables, classes, and functions are defined within functions or globally. This part of the solution is very similar to PyTA solution, but we created own checks without wrapping Pylint in between. In practice we have a style guide describing the patterns, which are then implemented as rule based checks in ASPA. When ASPA is run, a structural analysis for program is conducted, as demonstrated in Figure 1.

```
1  data_list = []
2  def main():
3      data_list.append("abc")
4      print(data_list)
5      return None
6  main()
```

ASPA feedback for the program above:

```
Functions, detected:
Line 1: Global variable 'data_list'.
```

Figure 1: Detection of a global variable.

Program shown in Figure 1 is evaluated against three rules: First, the file must contain a function called main. Second, the main level code should contain only one function call, which calls the main-function. Third, variables should not be defined in global namespace. Notice that the third rule focuses on global variables, and global constants are acceptable. Since Python does not support constants, ASPA needs to differentiate global variables and constants by checking how often the identifier is set – if it is set once, it is considered constant, but if it is set more often, then it is considered a variable. In Figure 1 `data_list` is defined in the global namespace and is modified after definition, so it is a global variable and violates the style guide. While comparing to other solutions, such as PEDAL and AutoSyle which provide hint for solution, ASPA points the violation and style guide is student's is source for correct style. This way student can also see correct solution before and after usage of the tool.

ASPA was first introduced in our CS1 in 2020, and currently it has 41 checks. Over time the rule base evolves as Python develops and the students get more creative. For example, Figure 1 demonstrates the basic global variable, but we have also seen students using classes and functions as global variables. Namely, classes are defined in the global namespace and if an object is not created but a class attribute is set directly, it can be used as a global variable. In the same vein functions are defined in the global namespace and since it is possible to add an attribute to a function, this can be used as a global variable. In general this kind of solutions cannot be considered good programming style, and thus their use is forbidden by our style guide. In short the idea is to document the principles in the style guide, e.g. "global variables are forbidden", and then check all the different ways to implement them by a rule based tool. This systematic process becomes important when both the number of student programs and rules increase to assure that all the student submissions are evaluated in the same way and in a reasonable time.

Since ASPA was developed to be used in the first programming course with freshmen students, a key design goal was simplicity. Figure 2 shows the ASPA graphical user interface (GUI), which offers some features both for the students and the staff. The students get access to ASPA after lecture 5 in which the functions and namespaces are explained. The following lectures address file handling, data structures – lists, classes, objects – self-made libraries, and exception handling. Thus students can select, on the left, with checkboxes the relevant analyses; **Basic commands** – e.g., checks for loops and unreachable code, **Functions** – checks for functions, parameters, and return values, **File handling** – checks for file opening, reading, writing, and closing, **Data structures** – checks for lists, classes, and objects, **Library usage** – checks for imports and library function calls, **Exception handling** – checks for proper exception handling in file operations. On the top right it is possible to select a single file for analysis, or a folder. The folder option is for staff so that a desired group of programs can be analyzed at once. The actual analysis is started with the button on the bottom, and results are shown in a separate window.

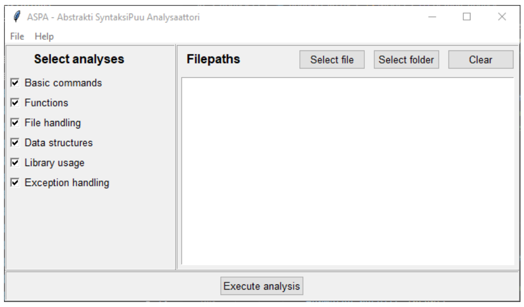


Figure 2: ASPA graphical user interface.

Table 1
Correlation values between usage of ASPA in programming assignments and performance in course tasks. Used correlation methods: ^{P} = Pearson's r, ^{K} = Kendall's Tau B. Significance values: *p-value < 0.05, **p-value < 0.01, ***p-value < 0.001.

ASPA usage (rows) compared to performance (columns)	Weekly assignments	Course project	Examination	Final grade
Amount of ASPA usage in weekly assignments ^{P}	0.407^{***}	0.371^{***}	0.360^{***}	0.392^{***}
Amount of ASPA usage in course project ^{K}	0.222^{***}	0.206^{***}	0.171^{**}	0.212^{***}

2.3. Results

ASPA was introduced in our CS1 course in the fall 2020 and it is now a standard part of the course. Our standard course feedback survey in fall 2022 was answered by 156 or 26.1 % of the students and 92 % of respondents had used ASPA while doing the course project. The most common reason for not using ASPA, 3 % of the respondents, was that they did not want to use an extra tool. In general with Likert scale of 1-5; 1 worst and 5 best, ASPA was found useful for the project with an average 4.4, selecting the checks was easy 4.5, selecting the files was easy, 4.6, and the results were understandable 4.5.

A more comprehensive analysis of ASPA and its usefulness was done for 2020 course [13]. Also then students found ASPA useful as seen in Figure 3.

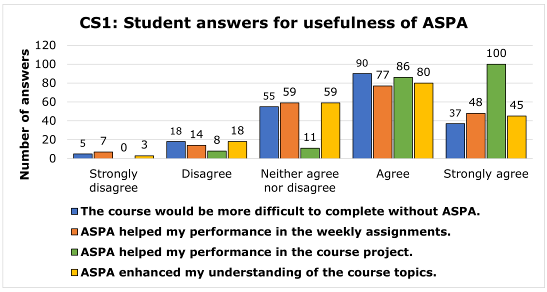


Figure 3: Usefulness of ASPA in CS1 course.

At the end of 2020 course, 205 students reported their ASPA usage via survey. Weekly assignment data is an integer number representing student's answer to a question *In how many weekly assignment you used ASPA?* and course project data is an answer to a question *Did you use ASPA while doing the course project?*, with options “No”, “Yes, once”, “Yes, 2-3 times”, or “Yes, multiple times”. Pearson's correlation is used to analyse correlation between weekly assignments and performance on programming assignments, while for the course project usage, which is ordinal data, Kendall's Tau B is used instead. In both there is a clear positive correlation between ASPA use and grades, presented in Table 1. For correlation calculations JASP program [14] was used.

2.4. Summary

As of now we are using ASPA in our CS1 with Python, but we have also considered extending it to other programming courses. Current ASPA is implemented with the Python 3 abstract syntax tree, but since syntax trees are available for other languages, ASPA can be extended for other languages. One key benefit of the AST approach is that it allows measuring student learning based on their submissions. We have done studied learning earlier [15], and initially ASPA seems to fit this task well. ASPA was initially developed with Tkinter to avoid need to any installations before it can be used. However, as seen in our more recent tool development efforts, other user interface frameworks could help improve the usability and make the tool even easier to use than now. A key consideration with ASPA development is synchronizing it with the autograder style checker. As of now a simple standalone tool seems to help students follow the style guide rules, but since our current autograder has a built-in style checker, it is faster and more powerful in enforcing the rules, and we need to study this balance closer.

3. GradeTool - Tool to assist grading process

In LUT programming courses various tools, including ASPA mentioned in Section 2, are assisting teaching assistants (TAs) in assessment process of programming assignments. The current aim in introductory courses is to synchronize assessment process between TAs and help them to provide consistent feedback for such cases where automated grading is limited or is not yet supported at all. Generation of feedback based on selected violations would both save time of TA, due to the decreased amount of writing required, and standardize majority of given feedback. However, there are cases where general feedback is not enough, and content should be personalized and therefore possibility to modify the feedback is required. In addition, the usability of grading tool should such simple that assessment process is easier and faster compared to more primitive methods such as spreadsheets and text editors.

We decided to develop a tool, called GradeTool, for this purpose. A development process started in summer 2022 and currently the second version of the tool is in use. Course staff uses GradeTool to keep track of programming errors and style violations in assessed student submission, and based on marked errors the tool calculates grade, generates categorical feedback and skeleton for open feedback. Open feedback can be modified by evaluator, if needed. In this study we describe our GradeTool-tool in short. The tool was developed for the first programming course with Python starting with the following research questions:

The research questions are as follows:

RQ3: Can a grading tool make evaluation process of programming assignments easier and faster for teaching assistants?

RQ4: What kind of programming violations are commonly done by students?

3.1. Literature review

Various solutions to assist grading have been developed, such as Labtool, a website to given and share feedback from staff to students [16], ALOHA, a grading rubric based online tool to generate feedback based on selected errors [17, 18], GradeIT, a grading and program repair system for TAs [19].

In addition, automated grading systems, which enable feedback generation, have also been utilized in education. For example, an automatic grading system, which utilized Bash scripting, grep and regular expression [20], gdb and valgrind based spectral error localization solution to generate feedback report [21], and education platforms ViLLE [11] and A+[12] which allow both automated tests and manual feedback.

3.2. Developed solution

GradeTool has been iteratively developed since summer 2022, and in addition to functionality, usability of the tool has been continuously improved by removing unnecessary steps and creating GUI responsive with flexible layout to allow grader to modify to it if desired. GradeTool is developed with Python and selected GUI module is Dear PyGui [22], which enables variety of layout flexibility out of the box. To store all possible violations and all marked violations, grades, feedback etc. metainformation for each graded submission, JSON files are utilized.

A grader can mark all the errors and violations via GUI, shown in Figure 4, as well as modify open feedback. In Figure 4, list of all categorized violations can be seen on the left side of the GUI, with category `Parameters and return values` being open. Inside the open category a programming violation *global variable* is marked to exist twice in the selected students' submission. The graded students can be selected on the right side of the GUI, while below this student list, level of submission, current grade, and current amount of errorpoints are shown. At the bottom right the generated skeleton for open feedback is shown, currently there are three different violations selected, aforementioned global variable violation, file which is left open, and close command without parenthesis. Finally, at the top right there is `WRITE TO FILES`-button which saves all the assessments files.

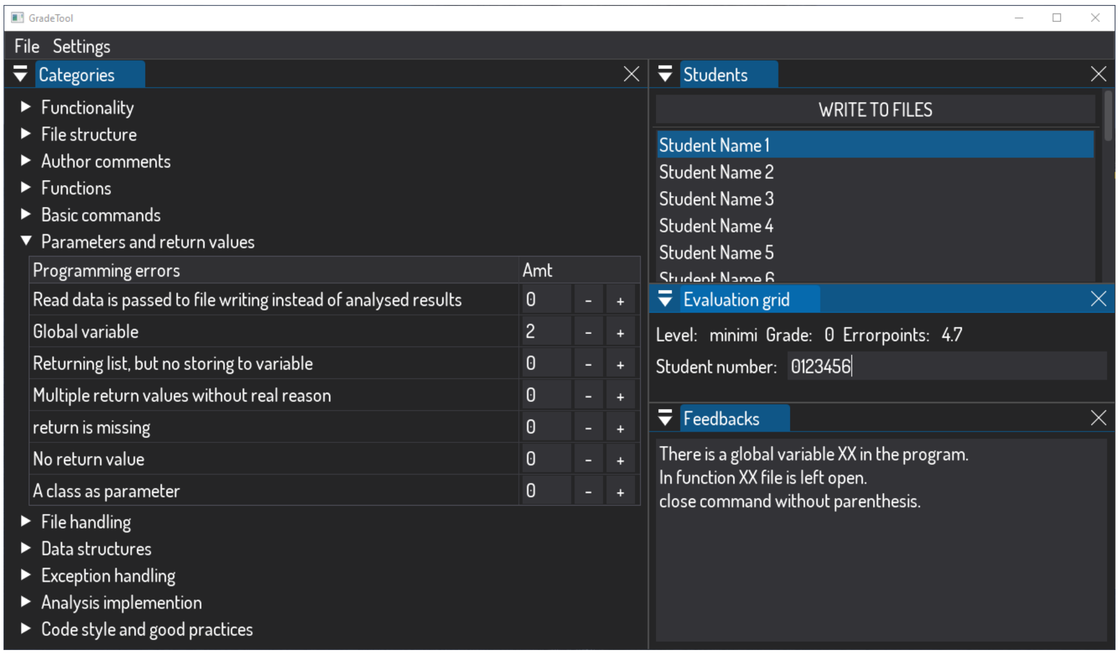
Open feedback in `Feedbacks` section can be edited by grader if needed, similarly, as in ALOHA tool [17]. In some cases, it is also intended to give students more specific and personalized feedback, e.g., in simple case giving name of global variable or in more complicated case explain why implemented analysis option is not suitable but is against the project instructions or style guide.

To enable selection of common violations, these violations are related to a certain assignment and all related information, i.e., identifier, name visible for grader, category, errorpoints depending on the error amount, and generated feedback phrase, are stored to a JSON file before starting an assessment process. However, violation set can be updated and modified during the evaluation process, if needed and just by reopening GradeTool with updated violation file updated errors can be selected. A minimal example of violation set used is shown in Figure 5.

ID is unique identifier of a violation, text is label shown in GUI, feedback is generated skeleton phrase and category is heading under which the violation belongs to in GUI and in feedback. As same violation can be done multiple times in same submission there is possibility to scale grading based on how often certain violations occur, e.g., in Figure 5 *close command without parenthesis* has three error values, for occurrences 1, 3 and All. One time is considered as a careless mistake and therefore only 0.7 error points are given, while 3 times doing same violation is not anymore just a careless mistake and therefore more error points, 1 in this case, are given. The label All is used when student has never done this specific part correctly, and it leads to the most error points, in this case 2. In the GUI this can be marked with value of -1.

On the other end, *Global variable* is considered as a fatal error and therefore already a single occurrence leads to 2 error points, in this case all option is not very relevant but is stored to allow similar behaviour for every violation.

In addition to traditional grading, GradeTool enables grading of multiple levels of same assignment, meaning that the grade calculation is based on marked violations and the level of submitted programming project. In LUT CS1 course project three levels were used successfully. When assessment is done, marked violations



- Functionality
- File structure
- Author comments
- Functions
- Basic commands
- Parameters and return values

Programming errors	Amt		
Read data is passed to file writing instead of analysed results	0	-	+
Global variable	2	-	+
Returning list, but no storing to variable	0	-	+
Multiple return values without real reason	0	-	+
return is missing	0	-	+
No return value	0	-	+
A class as parameter	0	-	+

- File handling
- Data structures
- Exception handling
- Analysis implemention
- Code style and good practices

Student Name 1
Student Name 2
Student Name 3
Student Name 4
Student Name 5
Student Name 6

Level: minimi Grade: 0 Errorpoints: 4.7
Student number: 0123456

There is a global variable XX in the program.
In function XX file is left open.
close command without parenthesis.

Figure 4: GradeTool graphical user interface.

can be saved to a JSON file, which allows returning to assessment process and continuing later from exactly same assessment state, if needed. In addition, generated feedback also saved to a separate file which is used while importing assessments to learning management system (LMS). The entire generated feedback contains three sections in following order. First general information about what was graded and overall pass/fail grading for that submission. Second, category feedback for each category, in CS1 course 11 categories were used and they can be seen in Figure 4, and third, open feedback, which is the only part intended to be modified by graders.

3.3. Results

GradeTool was used in LUT CS1 course during fall 2022 and CS2 course during spring 2023. In the latest use during CS2 course with four graders and 169 graded course projects, assessment process with GradeTool resulted over 17 000 words and 2860 lines of feedback, while excluding empty lines, which is almost 17 lines of feedback per student. Empty lines were used to distinct mentioned sections from each other and to make feedback more readable within sections. While the amount is large, also consistency of the feedback has, at least so far, been consistent enough as there have not been any complains about varying grading between similar violations, however, few feedback comments were perceived too general and did not help students to locate their actual problem.

From graders perspective, three TAs, with experience from previous years courses, estimated the time used to give these feedbacks being 33 % - 50 % less than with previous methods they were using. However, these values are estimates and are not statistically significant. On the other hand, assessment results were published to students 10 days earlier than estimated publication date. The estimation was based on previous assessment process.

Assessment of student submissions with GradeTool generates also quantitative data about violation occurrences. In fall 2022 Finnish version of CS1 course with circa 600 students, the course project assessment resulted 1418 violations for 76 different violations. As an example, subset of this dataset with only 10 violations is shown in Table 2. The colouring is used to visually highlight higher values, used coloring scale is from green, 0 occurrences, to red, the highest occurrence, in this example case 13.

In the complete set, the most common violations were programming practice violations which violated style guide. The highest value of 196 occurrences was with *Clearing data structures at the end of the program*, the second highest value of 96 occurrences was *initialization of values in analysis was not done according to guide.* The style guide guides to initialize value with the first element of analysed list to ensure that analysis will always work regardless of dataset. However, many students ini-

Table 2
Number of violations marked in CS1 course project grading by each teaching assistant. As a demo only 10 out of all 76 given violations are shown and TA columns from 4 to 10 are compressed.

Violation name	TA 1	TA 2	TA 3	...	TA 11	TA 12	TOTAL
Multiple similar variable instead of a list	7	4	10		11	10	69
Entire analysis done with multiple `if` branches	7	1	7		10	9	58
`except` without an exception type	3	1	0		3	13	53
File is read in other than readfile selection	1	4	9		0	0	41
Global variable	3	0	1	...	8	0	15
File is left open	0	1	1		0	0	13
`close` command without parenthesis	1	0	0		0	1	8
No main function	1	0	0		0	0	3
Class is nested inside a function	1	1	0		1	0	3
Hardcoded results	0	1	0		0	0	1
TOTAL	24	13	28	...	33	33	264

```
{
        "violations": [
        {
            "ID": "TV0601",
            "text": "Global variable",
            "feedback": "There is a global
 ↪  variable XX in the program.",
            "category":"Parameters and return
 ↪  values",
            "error_values": {
                "1": 2,
                "All": 2
            }
        },
        {
            "ID": "TK0704",
            "text": "close command without
 ↪  parenthesis",
            "feedback": "close command without
 ↪  parenthesis.",
            "category":"File handling",
            "error_values": {
                "1": 0.7,
                "3": 1,
                "All": 2
            }
        }
    ]
}
```

Figure 5: As en example of two violations from JSON file storing programming violations.

tialize values with real numbers, e.g., for minimum and maximum search they use "big value" and zero (0), or zero for both. In cases with negative values in dataset zero works also for minimum, but often that is not the case.

3.4. Summary

GradeTool has successfully been used in assessment process of CS1 and CS2 courses with hundreds of students. With four graders, the produced 2860 lines of feedback, excluding empty spacer lines, is such high amount of text that it is easy to argue that GradeTool have saved huge effort of writing all that. In addition, the feedback generated by GradeTool is consistent enough, but feedback phrasing should be focused on more in the future to avoid unnecessary misunderstandings and confusion about feedback.

The data about time used by TAs is not statistically significant and it is based on TA estimates, not measured values. However, being able to publish course project assessments 10 days earlier in a mass course like LUT CS2, is a great success. Moreover, gradings can be also directly imported to LMS, which objectively saves time and removes one possibility for human errors. In addition, estimated difference is high and amount of generated feedback is high compared to fully manual feedback, therefore benefit of tool should be studied more to thoroughly answer RQ3 about faster and easier evaluation process. However, it is important to mention that amount itself is not our target but quality of the given feedback. As an answer to RQ4 the most common violations for CS1 course are *clearing data structures at the end of the program*, and *initialization of values in analysis was not done according to guide.* which are style guide violations. As exact occurrences for each marked violation are now available and in the future, these can be used to focus teaching more on topics which seem to create the most problems. Moreover, detection of clearly distinguishable assessments between TAs is also possible, which could be utilized to standardize grading and marking of violations even more to ensure fair and standardized assessment for every student.

4. Mímir - Tool to manage course assignments

While detection of plagiarism in normal academic writing is fairly straightforward, the detection of plagiarism in source code can be difficult [23, 24]. In addition, simple plagiarism detection tools can be easily fooled by changing variable names, by changing the structure of the program, or by doing other types of obfuscations [25]. Moreover, when similar programming assignments, with no major updates to them, are utilized multiple years in a row students start to do plagiarism even more [26]. To mitigate these two plagiarism problems, i.e., lack of advanced plagiarism detector and similarity of assignments over the course implementations, in our programming courses an autograder was changed and all the programming assignments in LUT CS1 and CS2 courses were completely updated in fall 2022 and in spring 2023, respectively. Current autograder used in LUT programming courses, CodeGrade, has integrated plagiarism detector called JPlag, which have detected many obfuscated student submissions already [5, 27]. An enormous effort of updating assignment included designing new assignments, programming example solutions, rewriting assignment instructions and setting up autograder tests for circa 100 assignments.

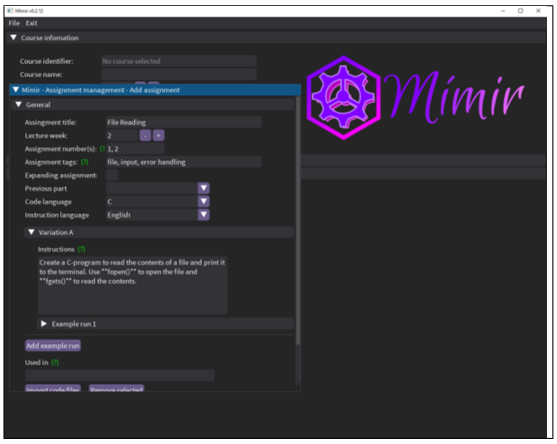
Since the effort for recreating assignment base was time-consuming, we decided to develop a tool to manage assignments and their variations for upcoming course implementations. Tool needed to be able to generate assignment instruction papers based on information inputted by user. The generated document would naturally need all the assignment related information which were available in handmade instructions, such as assignment instructions, example outputs, and possible example data from file(s). In addition, for course TAs an example solution is needed to be attached to the document. The developed tool is called Mímir, and in this study we describe our Mímir-tool in short. The tool was developed for the CS1 and CS2 courses with Python and C, respectively, starting with the following research questions:

RQ5: How to manage multiple sets of programming assignments?

RQ6: Are generated assignment instructions clear and understandable enough for students?

4.1. Developed solution

Since, according to our knowledge, there is no publicly available solution, which would solve our assigment management problem, we created our own proof-of-concept solution for such tool. The tool, Mímir, helps a programming course instructor to combat plagiarism by automating selection of course assignments from a pool of similar assignments, moreover, automates generation of assignment instruction document provided to student, which ensures they are always done with consistent formatting and layout. Mímir is developed with Python and selected GUI module is Dear PyGui [22], in addition, instruction document generation utilizes LaTeX, which features a robust and programmatically easy way to compile uniform and clear documents from different types of data, and JavaScript Object Notation (JSON) files are used to store assignment data. It is intended to be easy to use and to help in compiling the assignment instructions from the selected assignments, while keeping the visual style and readability consistent across each lecture week during one course implementation, as well as across multiple course implementations. This aims to help students focus on the task itself and helps the course instructor to focus on assignment creation and other duties rather than trying to make the visuals of the assignments readable enough. In addition, Mímir enables the instructor an easy method of editing the assignments based on previous feedback or changes in teaching. This way the edits are easy to implement and do not require fiddling simultaneously with text editors and IDE. User can input all information needed, e.g., title, lecture number, instructions, and example files, to generate assignment instructions via GUI, shown in Figure 6, or by utilizing existing files, e.g., source code files of an example solution.



Course identifier: No course selected
Course name:
Assingment title: File Reading
Lecture week: 2
Assignment number(s): 1, 2
Assignment tags: file, input, error handling
Expanding assignment:
Previous part
Code language: C
Instruction language: English
Instructions:
Create a C-program to read the contents of a file and print it to the terminal. Use **fopen()** to open the file and **fgets()** to read the contents.
Used in

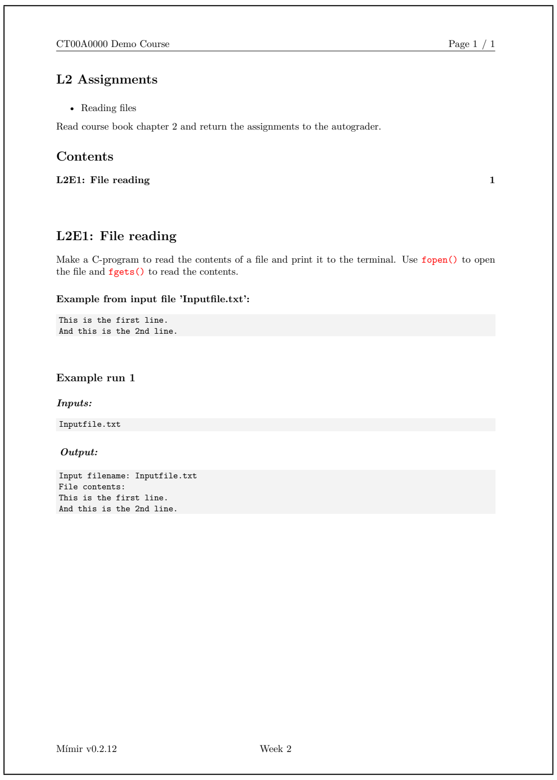
Figure 6: Mímir GUI with assignment pop-up open.

Course metadata is inserted in a main window, seen in the background in Figure 6, and for the new assignment a pop-window is opened, seen in foreground in Figure 6. In this demo example, the assignment is simple, with only single variation called “variation A”, and instructions part is extremely short. After insertion data is stored to a JSON file, which structure can be seen from Figure 7.

```
{
    "course_id" : "CT00A0000",
    "course_name" : "Demo Course",
    "assignment_id" : "XYZ",
    "exp_lecture" : 2,
    "exp_assignment_no" : [1, 2],
    "tags" : ["file", "input", "error handling"],
    "title" : "File reading",
    "next, last" : [],
    "code_language" : "C",
    "instruction_language" : "ENG",
    "variations" : [
        {
            "variation_id" : "A",
            "instructions" : "Create a C-program
 ↪  to read the contents of a file and print it
 ↪  to the terminal. Use \\texttt{\\textcolor{red}
 ↪  {fopen()}} to open the file and \\texttt{
 ↪  \\textcolor{red}{fgets()}} to read the
 ↪  contents.",
            "example_runs" : [
                {
                    "generate" : true,
                    "inputs" : ["Inputfile.txt"],
                    "cmd_inputs" : [],
                    "output": "Input filename:
 ↪  Inputfile.txt\nFile contents: \nThis is the
 ↪  first line.\nAnd this is the 2nd line.\n",
                    "outputfiles" : []
                }
            ],
            "codefiles" : ["ExampleCode.c"],
            "datafiles" : ["Inputfile.txt"],
            "used_in" : []
        }
    ]
}
```

Figure 7: An example of the JSON file used to generate assignment with Mímir.

The main level value pairs in JSON object contain the general assignment information that are not specific to a variation, or a single example run. As assignments are not fixed to single week or order per week and therefore also these metainformation values are editable. The variations of the same assignment are then listed as a list of assignment objects, each having their own versions of instructions and paths to the files. Example runs, generated to the instruction paper, are tied to a variation and there can be multiple example runs per variation, therefore, they are stored inside a list that is inside the variation. In addition, an example run object must store inputs and outputs for that specific run as they may differ. Finally, each variation also stores implementations that it has been used in, such as ”Spring 2023”. Generated assignment instructions for this demo assignment are shown in Figure 8. In normal course setting there would be 3-5 assignments per week, but now only one assignment was added to demo week 2.


CT00A0000 Demo Course — Page 1 / 1

L2 Assignments

- Reading files

Read course book chapter 2 and return the assignments to the autograder.

Contents

L2E1: File reading — **1**

L2E1: File reading

Make a C-program to read the contents of a file and print it to the terminal. Use `fopen()` to open the file and `fgets()` to read the contents.

Example from input file ’Inputfile.txt’:

```
This is the first line.
And this is the 2nd line.
```

Example run 1

Inputs:

```
Inputfile.txt
```

Output:

```
Input filename: Inputfile.txt
File contents:
This is the first line.
And this is the 2nd line.
```

Mímir v0.2.12 — Week 2

Figure 8: Demo assignment generated by Mímir.

While example in Figure 8 is done with C, Mímir is language independent, so that it can be used on multiple courses that use different programming languages. However, as the tool is designed to be used on programming courses specifically, there are no features or options for other type of assignments. The major external dependencies which are not included in the program itself are `pdflatex` program and programming language’s compiler or runtime. `pdflatex` is needed to compile the LaTeX documents into Portable Document Format (PDF) files, and compiler or runtime are used to generate example runs by executing an example solution. However, the example solution compiling and running is voluntary, so if user chooses not to use the feature, external compilers or runtimes are not needed.

4.2. Results

A survey about generated assignment instructions was conducted during CS2 course in spring 2023, in total 65 out of 240 students, i.e., 27.1 %, answered. Students were asked how they perceive visual clearness of generated assignment instructions, as well as, how the generated documents compared to handmade documents, used in the first four weeks of the course. In addition, there were four questions about how layout and formatting highlights help students while doing their assignments. Answer distributions for these question sets are shown in Figures 9 and 10, respectively.

As seen in Figure 9, with the scale of 1-5; 1 worst and 5 best, documents generated by Mímir were considered slightly clearer than handmade ones, with an average 3.2, colored highlight of keyword was perceived better than only bolding the keyword, with an average 3.9. Visual clearness and readability as well as distinction of separated instruction sections were considered good with averages 3.9 and 4.2, respectively. As seen in Figure 10, formatting and layout helping students with assignment were perceived very good for listing functions in bullet points, separating inputs as a distinct section, colouring keywords, and background highlighting, with averages 4.1, 4.4, 4.1 and 4.1, respectively.

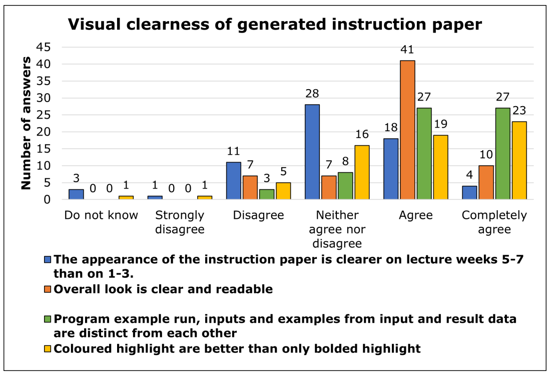


Figure 9: Visual clearness of instruction paper generated by Mímir.

4.3. Summary

Mímir is a tool, developed with Python 3, to manage sets of programming assignments, with option to automate selection of programming assignments for new course implementations. It stores assignment data to JSON files and utilizes LaTeX to generate assignment instructions as PDF files. The tool is designed to be used by course staff either before the course implementation to generate dataset for the entire course at once, or when needed before each assignment set is released. While Mímir is not for students, they are a stakeholder as they the target group for generated assignment instruction papers, and therefore the survey was conducted to gather feedback

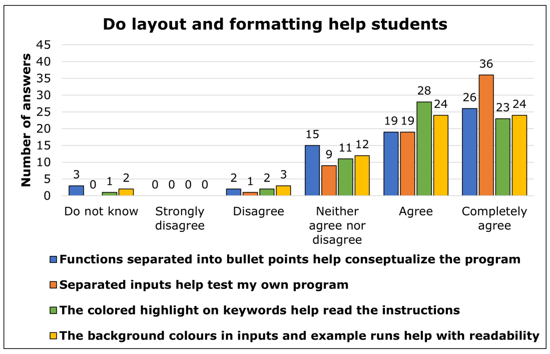


Figure 10: Do generated layout and formatting help students understand the assignment instructions.

about generated documents. Based on the survey students were satisfied with the generated content, which motivates us to develop the tool more and utilize it in upcoming courses. As Mímir is not limited to generate assignments for CS2 course and C programming language, it is planned to be used for fall 2023 CS1 course lectured with Python 3.

For the future development Mímir could be utilized to automatically generate autograder test cases, which could then be imported to autograder. However, this development option requires closer cooperation with the used autograder.

5. Conclusion

The LUT CS1 course has continued to grow over a decade now. To improve the quality to the student materials as well as to reduce the time and effort required to complete all the course tasks, we are currently developing three new tools to support the course arrangements. In this paper we described three tools we are developing - ASPA, GradeTool and Mímir - and presented initial data that indicate the usefulness of these tools.

ASPA is an AST-based static analyzer tool to detect common programming style violations, and saves time both for students and teaching assistants by providing feedback anytime needed. Thus TA can focus on less common problems which require human to be detected or to explain reasons for required changes to student. Empirical data shows that students find the tool useful, and numerical data shows a statistically significant positive correlation between ASPA usage and student grades.

GradeTool is a tool to assist programming assignment grader in assessment process by providing convenient way to mark detected programming violations. Empirical data shows that the feedback generated by GradeTool is consistent enough, but feedback phrasing should be focused on more in the future, and numerical data shows that with help of the tool, graders can generate a lot of feedback for students without need to write everything. Furthermore, while using GradeTool TAs generate numerical data about violation occurrences which can used to improve teaching and assessment processes even more.

Mímir is a programming assignment manager to handle assignment variations and generate assignment instruction documents for selected assignment variations. Empirical survey data shows that generated assignment instruction documents are clear and consistent enough, even a little bit better than handmade ones, which were used earlier. In addition, students perceived that, on the average, used layout and formatting highlights helped them to focus on the actual programming task and its implementation instead of assignment instructions.

Each of these tools has already been successfully used in programming courses and they suit for CS1 context, both in Finnish and in English, with Python programming language but are not limited to it, which is why GradeTool and Mímir are already used also in CS2 course lecture with C programming language. Due to the success, development is continued, and aim is moved for integrating these tools together more closely not just use their outputs in same course. For example, violations detected by ASPA are aimed to be importable directly to GradeTool, which would even more fasten the assessment process when grader would not need to even mark already found violations. On the other hand, the future development of Mímir could be automatic generation of autograder test cases, which could then be imported to autograder, which would then save time on assignment setup phase.

Next iterations for these development actions are planned to be done for the fall 2023 CS1 course and expand range to other programming courses as well. In the future, we also could expand our analysis to cover topics related to the support services of code generation, such as use and role of artificial intelligence, or taking an indepth look into the discussions and common problems the teaching assistants encounter on the online support platforms.

References

[1] U. Nikula, O. Gotel, J. Kasurinen, A Motivation Guided Holistic Rehabilitation of the First Programming Course, ACM Transactions on Computing Education 11 (2011) 1–38. doi:10.1145/2048931.2048935.

[2] U. Nikula, J. Järvinen, Ohjelmoinnin perusteet (6 op) / Introduction to Programming (6 cr), 2023. URL: https://sisu.lut.fi/student/courseunit/otm-d92aa788-3b57-4580-bb82-60b836198ee9/brochure.

[3] E. Vanhala, U. Nikula, Python 3 – ohjelmointiopas versio 1.2.1, LUT University, 2020. URL: https://urn.fi/URN:ISBN:978-952-335-622-1.

[4] The Finnish Institute of Technology, Fitech: Apply to summer courses from 4 april, 2023. URL: https://fitech.io/en/.

[5] CodeGrade, Streamline code learning and grading., 2021. URL: https://www.codegrade.com/.

[6] D. Liu, A. Petersen, Static Analyses in Python Programming Courses, in: Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19, ACM Press, Minneapolis, MN, USA, 2019, pp. 666–671. doi:10.1145/3287324.3287503.

[7] L. Gusukuma, A. C. Bart, D. Kafura, Pedal: An Infrastructure for Automated Feedback Systems, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, ACM, Portland OR USA, 2020, pp. 1061–1067. doi:10.1145/3328778.3366913.

[8] R. R. Choudhury, H. Yin, A. Fox, Scale-Driven Automatic Hint Generation for Coding Style, in: Proceedings of the 13th International Conference on Intelligent Tutoring Systems - Volume 9684, ITS 2016, Springer-Verlag, Zagreb, Croatia, 2016, pp. 122–132. doi:10.1007/978-3-319-39583-8_12.

[9] E. S. Wiese, M. Yen, A. Chen, L. A. Santos, A. Fox, Teaching Students to Recognize and Implement Good Coding Style, in: Conference on Learning @ Scale - L@S '17, 2017, pp. 41–50.

[10] r2c, Semgrep: Code scanning at ludicrous speed., 2023. URL: https://semgrep.dev/docs/.

[11] E. Kaila, M.-J. Laakso, T. Rajala, E. Kurvinen, A model for gamifying programming education: University-level programming course quantified, in: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE, Opatija, 2018, pp. 0689–0694. doi:10.23919/MIPRO.2018.8400129.

[12] V. Karavirta, P. Ihantola, T. Koskinen, Service-Oriented Approach to Improve Interoperability of E-Learning Systems, in: 2013 IEEE 13th International Conference on Advanced Learning Technologies, 2013, pp. 341–345. doi:10.1109/ICALT.2013.105, iSSN: 2161-377X.

[13] R. Luukkainen, J. Kasurinen, U. Nikula, V. Lenarduzzi, ASPA: A static analyser to support learning and continuous feedback on programming courses. an empirical validation, in: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET '22, Association for Computing Machinery, New York, NY, USA, 2022, p.

29–39. doi:10.1145/3510456.3514149.

[14] JASP Team, JASP (Version 0.14.1)[Computer software], 2020. URL: https://jasp-stats.org/.

[15] J. Kasurinen, U. Nikula, Estimating programming knowledge with Bayesian knowledge tracing, ACM SIGCSE Bulletin 41 (2009) 313–317. doi:10.1145/1595496.1562972.

[16] Labtool-2019 group, Welcome to the labtool wiki!, 2022. URL: https://github.com/UniversityOfHelsinkiCS/labtool/wiki.

[17] T. Ahoniemi, T. Reinikainen, Aloha - a grading tool for semi-automatic assessment of mass programming courses, in: Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006, Baltic Sea '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 139–140. doi:10.1145/1315803.1315830.

[18] T. Ahoniemi, E. Lahtinen, T. Reinikainen, Improving pedagogical feedback and objective grading, in: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 72–76. doi:10.1145/1352135.1352162.

[19] S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare, 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, ITiCSE '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 92–97. doi:10.1145/3059009.3059026.

[20] E. Hegarty-Kelly, D. A. Mooney, Analysis of an automatic grading system within first year computer science programming modules, in: Proceedings of 5th Conference on Computing Education Practice, CEP '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 17–20. doi:10.1145/3437914.3437973.

[21] J.-Y. Kuo, H.-C. Lin, P.-F. Wang, Z.-G. Nie, A feedback system supporting students approaching a high-level programming course, Applied Sciences 12 (2022). doi:10.3390/app12147064.

[22] J. Hoffstadt, P. Cothren, A modern, fast and powerful gui framework for python., 2023. URL: https://github.com/hoffstadt/DearPyGui/wiki.

[23] H. T. Jankowitz, Detecting plagiarism in student pascale programs, The Computer Journal 31 (1988) 1–8. doi:10.1093/comjnl/31.1.1.

[24] S. Mann, Z. Frew, Similarity and originality in code: Plagiarism and normal variation in student assignments, in: Proceedings of the 8th Australasian Conference on Computing Education-Volume 52, volume 52, 2006, pp. 143–150. doi:10.5555/1151869.1151888.

[25] L. Nichols, K. Dewey, M. Emre, S. Chen, B. Hardekopf, Syntax-based improvements to plagiarism detectors and their evaluations, in: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 555–561. doi:10.1145/3304221.3319789.

[26] T. Hynninen, A. Knutas, J. Kasurinen, Plagiarism networks: finding instances of copied answers in an online introductory programming environment, in: Proceedings of the 17th Koli Calling International Conference on Computing Education Research, Koli Calling '17, Association for Computing Machinery, 2017, pp. 187–188. doi:10.1145/3141880.3141906.

[27] L. Prechelt, G. Malpohl, M. Philippsen, Finding plagiarisms among a set of programs with JPlag, Journal of Universal Computer Science 8 (2002) 1016–1038.