Automatic Analysis and Evaluation of Student Source Codes

Ádám Pintér[1] and Sándor Szénási[2,3]

[1]Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, Budapest, Hungary
[2]John von Neumann Faculty of Informatics, Óbuda University, Budapest, Hungary
[3]Faculty of Economics, J. Selye University

IEEE 20th International Symposium on Computational Intelligence and Informatics
November 5-7, 2020, Budapest, Hungary

Introduction

Evaluation is an essential part of education and is useful both for students who receive feedback on the progress of their studies and for educators who can assess whether or not students have achieved their goal. In this approach, it has been shown that most students choose the easier path to achieve the best result during evaluation, instead of acquiring comprehensive knowledge, which would also benefit for companies. It follows that continuous assessment of the course can be used to guide and improve students' learning process. As manual evaluation requires a significant effort even in the case of small groups, it is advisable to try to find ways to automate some or all of the work. Automation is further justified by the fact that it is almost impossible to take all aspects into account in the evaluation, and it is also very rare for two instructors to perform the assessment based on the same criteria and arrive at the same result. Of course, the grades developed in this way will depend heavily on the instructor conducting the assessment, which may unfairly affect many students.

Materials and method

The source code evaluation model consists of the following main steps (Fig. 1.):
- Download the list of students in the course.
- Collect the templates and student solutions for the, then create a skeleton file that contains the templates and the unit tests needed to check the tasks at runtime, as well as the syntax rules for the task.
- Transforming students' source code into an abstract syntax tree (AST), pre-processing the solution based on AST, and deeper syntactic analysis.
- Transform the skeleton file into AST and then merge it with the student solutions to get the student standalone solution for a given homework.
- Compile standalone source code and run unit tests.
- Preparation of a detailed html report and a summary report in excel per student.

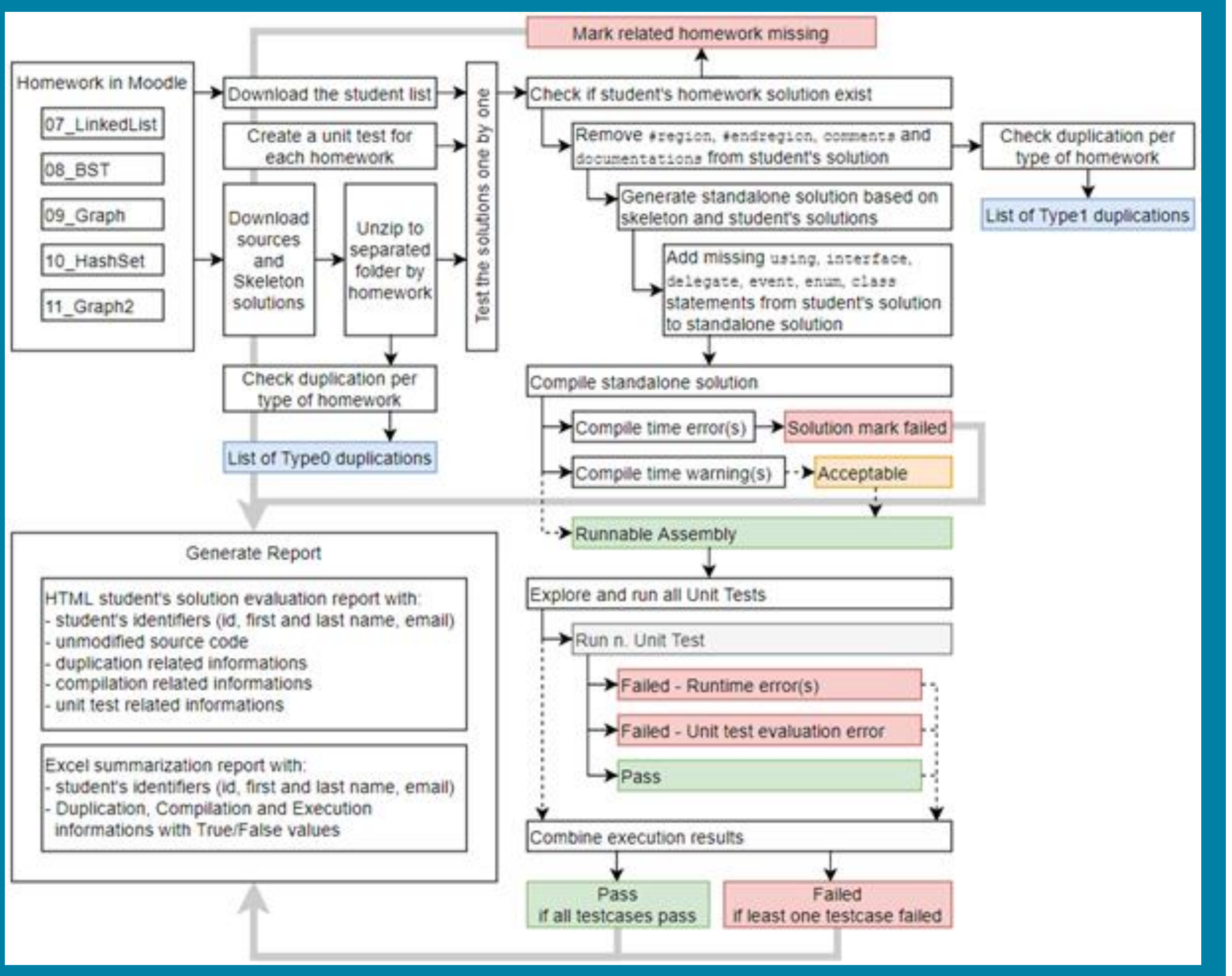
Mark related homework missing
Homework in Moodle
07_LinkedList
08_BST
09_Graph
10_HashSet
11_Graph2
Download the student list
Create a unit test for each homework
Download sources and Skeleton solutions
Unzip to separated folder by homework
Test the solutions one by one
Check if student's homework solution exist
Remove #region, #endregion, comments and documentations from student's solution
Check duplication per type of homework
List of Type1 duplications
Generate standalone solution based on skeleton and student's solutions
Add missing using, interface, delegate, event, enum, class statements from student's solution to standalone solution
Check duplication per type of homework
List of Type0 duplications
Compile standalone solution
Compile time error(s)
Solution mark failed
Compile time warning(s)
Acceptable
Runnable Assembly
Generate Report
HTML student's solution evaluation report with:
- student's identifiers (id, first and last name, email)
- unmodified source code
- duplication related informations
- compilation related informations
- unit test related informations
Excel summarization report with:
- student's identifiers (id, first and last name, email)
- Duplication, Compilation and Execution informations with True/False values
Explore and run all Unit Tests
Run n. Unit Test
Failed - Runtime error(s)
Failed - Unit test evaluation error
Pass
Combine execution results
Pass if all testcases pass
Failed if least one testcase failed

Fig. 1. A model for evaluating student homework

Results

The model was implemented in C#. The data were given by the homework of OE-NIK SzTF2 course, which were collected from the e-learning system of Óbuda University. The course was attended by a total of 364 students who had to complete 4 homework assignments during the semester, an overview of this is given in Table 1. The 1247 source codes uploaded during the semester could be verified by automatic evaluation in just 3 hours, to which is added the time for syntactic rules and unit tests. Overall, a solution can be evaluated in approximately 9 seconds. In contrast, instructors took an average of 30 minutes to complete a task, in addition to which several (not perfectly working) solutions were marked as correct.

Homework	Number of			Criteria			Unit Tests					Duplications	
	Solutions	CTW	CTE	C1 pass	C2 pass	C3 pass	Pass all	Failed 1	Failed 2	Failed 3+	Failed all	L0	L2
LinkedList	315 (87%)	30 (8%)	22 (6%)	279	233	287	106 (36%)	30	49	94	14 (5%)	2	2
BST	317 (87%)	6 (2%)	1 (1%)	274	175	306	199 (63%)	26	34	57	0 (0%)	4	17
Graph	309 (85%)	40 (11%)	26 (7%)	245	230	259	38 (13%)	95	12	116	22 (8%)	6	8
HashSet	306 (85%)	39 (11%)	15 (4%)	265	270	233	182 (63%)	0	42	49	18 (6%)	104	124

Table 1. Students' solutions and a summary of the results of the evaluation.

Conclusion

Overall, automatic assessment was found to be useful and forward-looking by both the students and the faculty. The biggest benefit for the instructors was that the review process was speed up (~99% time savings) and, thanks to the detailed report, students received more feedback on their submitted work, thus avoiding more general questions. In addition to quick and detailed feedback, independent and unbiased evaluation was the most important thing for the students, thanks to which they could master and complete the subject material more successfully.
For more complex accounting, the model has not been used as an automated evaluation system, because in such cases it is no longer necessary to run only an executable file, but even an underlying database or user interface (GUI), which, although not impossible, is more difficult to verify. It would make it more time consuming to build the evaluation logic than manual checking.
Our further development plans include expanding the rules of syntactic tests, automatically generating a large number and full unit tests for the tasks, which can reveal the faulty code in more detail, and adapting the evaluability of other programming languages taught at the University (Java, PHP, Assembly) to the model.

CINTI 2020