Automatic Grading of Introductory Programming Labs

JD Kilgallin

Questions

- What do UA CS 1 students have the most difficulty with during lab?
- At what rate do students learn & improve?
- Can we automatically detect an error in an ungraded lab?
- What areas of the lab grading process would be easiest to automate?
- Can grading the labs be fully automated?

Background

- Grading of Computer Science I labs has traditionally consisted of manual inspection of program code, output, and version control history, and comparison against a rubric with several items.
- This is tedious, time-consuming, and prone to missing errors.
- 13 labs are collected from over a hundred students and many rubric items are repeated across labs; these common items account for 25-60% of the total points in each lab.

Approach

- To alleviate these issues with lab grading, a promising solution involves automatic grading of lab submissions.
- Existing course infrastructure is built on an SVN repository containing student code and results. It is much simpler to examine student code and results directly from the file structure than to use a database.

Implementation

- Python was selected for the implementation of an autograder due to the ease of performing file and string operations, and the easy mapping between Python objects and JSON.
- Program takes as input the number or name of the lab to grade, and the id of the student to be graded. “all” can be specified for the user to grade all students in turn.
- A configuration file in JSON containing information about the lab requirements is loaded.

Implementation

- A text file containing the lab report is loaded and parsed into three sections:
    - Header, including results from compilation and execution
    - Program code
    - Version control history
- The commit history is checked first; if no commits have been made since the starter code a grade of 0 is assigned and grading of that lab ends. If an insufficient number have been made a point is deducted.
- If the program does not compile, 5 points are deducted.

Implementation

- The output of the code is then examined from the lab header. If there is a single error a point is deducted, while 2 points are deducted for multiple errors.
- The code is then checked for the required inclusion of the student’s name and email in a header comment.
- The code is then checked for a sufficient number of comments and wrapping of lines over 80 chars.
- Comments including the points lost and the reasons are printed to a text file in the student’s evaluation folder.
- The Lab Instructor then only need examine the autograder results and program code in most cases to complete grading

Results

- An initial version of the program was tested on a random sample of 40 submissions of Labs 1-4 in section 010, which had already been graded. 13 uncaught errors were detected for an average grade lower by 0.4 points out of 20, or 2%.
- The script was then used to assist in grading all submissions for Labs 6-9. The following errors were caught:

Lab	Lack of name/email	Lines over 80 chars	Too few comments	Too few commits
6	0.3	0.1	0.2	0.2
7	0.1	0	0.6	0.1
8	0.2	0	0.3	0.2
9	0.1	0.4	0.8	0.2

Limitations

- The script cannot detect all errors due to undecidability or infeasible program complexity.
- Many rubric components are not yet examined, such as proper indentation.
- There is some risk of incorrectly deducting points; for example, a student who puts his name in the header comments but does not delete the pre-existing line “Author: Your Name” will be penalized.
- There is also risk of incorrectly awarding points; for example, if a student comments out a section of code, it will still be counted as though it provided code documentation.

Conclusion

- Automatic grading of programs reduced lab grading times from over 4 hours to less than 2.
- In some labs, the majority of the grading by point value can be done automatically.
- Not all parts of a lab can be automatically graded, but many parts are easy to check.
- Automatic grading is an appropriate and useful tool for grading Computer Science I labs in complement with manual grading.

Future Work

- Checking of additional lab requirements such as proper indentation and required commit messages.
- Fuzzy string matching to accept small variations from the required strings.
- A Domain-specific language for specifying lab requirements with more flexibility than the JSON config file.
- Machine-learning based classification of labs to statically provide further information about program correctness.