Unit testing frameworks in the classroom

David Oniani, Roman Yasinovskyy
Luther College, Summer 2018

Abstract

This project explores the feasibility of using unit testing in a classroom environment in order to help students get immediate feedback from the test suite rather than delayed feedback from an instructor. The goal is not to replace an instructor whose feedback and guidance are usually more detailed and helpful, but to allow students to assess their progress and understanding of the content sooner. Feedback provision for the courses CS 140 and CS 253 was fully automated. The new refresher course was designed with the purpose of helping students refresh their python skills where feedback is also fully automated. As a side project, Linux's Ubuntu distribution was updated for all the the lab computers which now support the new LTS (long term support version of Ubuntu). Some scripts were also written to help instructor in assessing CS 253 assignments.

CS 140

CS 140 is the the data modeling and querying course where students explore the world of data and databases. It covers SQL and relational algebra as well as some basic concepts about the data. The assignments are given in the form of Jupyter notebooks where students have to complete the exercises in the notebooks and submit their solutions when finished. Prior to this research project, assignments were checked automatically yet, the checker was flawed which would usually result in the wrong and misleading feedback. As the part of the project, the checker has been completely

redesigned and now gives a meaningful feedback to the students as well as some hints on what they could've done better. Here is the part of the code used in the checker:

```
def check(result, expected, database='', points=10):
    if database:
        try:
            result = DataFrame(result, columns=(result and result.keys) or [])
        except ValueError:
            print("Something went wrong. Please, review your code")
            return

        result = result.convert_objects(convert_numeric=True)

        if type(expected) != list:
            expected = [expected]

        expected = read_sql_query(*expected, create_engine(database))

    if result.equals(expected):
        print("\33[32mCorrect!\n")
        print("You got {} out of 10 points.".format(points))
    else:
        return giveHint(result, expected)

    return points
```

As the name "check" implies, the function above tests whether the student's answer matches the original one. If these two answers match, the student has successfully completed the task. Otherwise, the function moves to the *giveHint* part where the student will be provided with some feedback on what he did wrong and what are the reasons his answer is not correct. Hence, now, instead of just checking whether the student's solution is correct or not, the checker also give hints if the student's answer is wrong. Such approach is efficient in several ways. Firstly, the student will be getting immediate feedback in lieu of delayed one from the instructor. Secondly, instructor will have more time dedicated to providing detailed, meaningful feedback. At last, since the checker provides an automated feedback, it will be easier for students to revise already covered material whether it's for the midterm, the final or something else. In addition, the maintenance of the checker and the Jupyter notebooks got a lot easier since the number and size of Python files are much less resulting in much cleaner environment to maintain. Overall, the project made a significant progress in CS 140 which now not only supports the idea of unit testing, but also is much easier to maintain.

CS 253

CS 253 is the introductory course which covers object oriented principles in the C++ programming language. The course is designed for the students who are eager to delve into and explore new programming language. Since the course is introductory, it is vital to make the process of learning as quicker and as efficient as possible so that students are able to have more time to grasp key ideas and concepts of the language. Unit testing comes handy in such situations. In the course, most of the tasks were provided with accompanying unit tests yet, some of them were left without any tests. Within the scope of the project, tasks without unit tests were rewritten in the way that now they support unit testing. Additional tasks and exercises were designed with corresponding unit tests. In total, three additional exercises were designed with the labels: easy, intermediate, and hard. These exercises will be used to either as a bonus credit or just the refreshing tools for some other classes where students might need to have a good grasp of C++ programming language. It should also be noted that the bash script was written in order to help an instructor with grading the exercises. The script allows the instructor to go thorough assignment directories, where instructors usually have students' solutions and/or answers to the problems, student by student and run the tests to see how many of the tasks were completed successfully by the student. Yet again, this is done with the premise that the instructor will still look into the students' work to give a more meaningful and detailed feedback.

Python Refresher

When students jump into the new Python course, it is always a good idea to refresh their knowledge before getting into the new ideas and concepts right away. This is what Python Refresher is designed for. Simply put, it is a bunch of problems stacked together to help students refresh their knowledge of Python programming language. The tasks vary in difficulty as well as in their purpose. Some of the main topics and ideas of Python such as strings, lists, dictionaries, sets, and file input/output are covered in these exercises. All the tasks have corresponding unit tests. Most of the

tasks are tested very strictly meaning that virtually all the edge cases are considered. All that is left for the students is to complete the tasks and run either of these three commands:

- `pytest example_test.py`
- `pytest -vv example_test.py`
- `pytest example_test.py::test_task1`

The first command lets students get the information about how many tests have been successfully passed while the second command, if run, will provide more detailed description on how many and which test cases were successfully passed. The third command, however, lets students test the individual tasks that is if there are more than one task in the assignment and student wishes to test the first one and then move forward (proceed incrementally), this is the command which the student would want to use.

Ubuntu 18.04 LTS

Once in two years, new LTS (long term support) version of the well-known Ubuntu operating system comes out. As the educational institution, one of Luther's primary objectives is to be up to date on new technology. This was the reason we took the time to work on the brand new Ubuntu 18.04 and make sure all the lab machines as well as the students are equipped with it. It is of the utmost importance for the current and incoming students to embrace the new ideas as well as the changes in the technological world. New Ubuntu has a lot of new features and changes since the previous LTS such as featuring Python3 as the system-wide Python version instead of old Python2 and new desktop environment as a whole. Software that the students will need is also preinstalled which makes it possible for them to jump straight into the course and get started right away without spending time on installations and configurations of the prerequisite software.