Teaching Computer Architectures through Automatically Corrected Projects: Preliminary Results

Mauricio Lima Pilla
Center for Technological Development
Federal University of Pelotas
Pelotas, RS, Brazil 96010-610
Email: pilla@inf.ufpel.edu.br

***Abstract*—In this paper, I report preliminary results of using GitHub and Travis CI as tools for assigning and grading projects in a course on Computer Architecture and Organization for undergraduate students in Computer Science and Engineering. Besides addressing the topics of the course itself, students are presented with development tools used in the industry with a hands-on experience. I present the workflow for assigning and grading students, some helper scripts, and results on submissions. Students tend to commit their code near the deadline, which is both because they procrastinate and they are still learning the philosophy behind version control, hence even with allegedly having coded part of the assignment, they leave commits for when the entire project is done.**

***Index Terms*—Computer Architectures, Memory Hierarchies, Teaching in Computer Science.**

I. Introduction

Automatic grading of assignments is not a novelty for a while. It is just recently that the availability of open, cloud-based versioning and continuous integration system started to make it easier to deploy solutions that are not ad-hoc in nature.

In this paper, I show a process for automatically distributing and collecting assignments using tools available free of charge in Computer Science and Engineering courses. A grading library in C and scripts for automation of grading tasks are provided.

My work is heavily influenced by the reports of Gennarelli in two case studies in GitHub: David J. Malan's CS50 classes in Harvard [1] and Omar Shaikh's C++ classes in San Francisco State University [2]. The former used GitHub and a grading script for instant correction of assignments, while the later included Travis CI into his workflow.

Besides the advantages of having an instantaneous feedback about assignments, students are also presented to tools widely used in the industry. Although some students already had contact with GitHub before this course, source versioning, continuous integration through Travis CI, and the case for test-driven development are not formally required by other undergraduate courses in my University to the best of my knowledge. Hence, alternatives such as the Virtual Programming Lab for Moodle (VPL) [3] for automatic grading code are interesting but they do not enforce the aspect of getting students to use industrial grade development tools. Other authors such as Lawrence et al. [4] agree that knowledge of version control is important for students majoring not only in Computer Science and Engineering.

This paper is structured as follows. First, I present an overview of the case test course in Section II. Afterwards, Section III describes the tools that I used. Section IV discusses the workflow from submitting a new assignment to its grading. Preliminary results obtained with the application of this workflow are shown in Section V. Finally, final remarks and future work are presented in Section VI.

II. Lay of the Land: the Course

The course "Computer Organization and Architecture II" (*Arquitetura e Organização de Computadores II* (AOC2) in the Portuguese original) is taught in both undergraduate courses in Computer Science (CS) and Computer Engineering (CE) in our school. It is preceded by "Computer Organization and Architecture I" and other related courses in digital systems and programming. In CS curricula, it is taught in the fourth semester, and for CE undergraduate students, in the fifth.

In this course, students learn about memory hierarchies, virtual memory, input and output, buses, and topics on advanced computer architectures (such as superscalar processors, simultaneous multi-threading, and multi-core architectures). AOC2 is pre-requisite for attending "Operating Systems" in both courses, and it is expected to provide the understanding of the underlying hardware and how it affects the design and implementation of operating systems.

It is important for the discussion of this paper that the most of the preceding programming courses in both CS and CE courses employ C as their language of choice, hence students should be reasonably acquainted with the usage of pointers at the point they course AOC2.

This is the first semester that I am lecturing this specific course in this University. Before that, I lectured undergraduate courses in Computer Architectures in another University, and I have also been lecturing graduate courses in this area since 2007, as my D.Sc. degree is in this field. I have also been lecturing other related courses such as Operating Systems for even more years.

Currently, I set up an evaluation comprised of three projects which sum up 60% of the grade, and an exam for the remaining 40%, for a total of 10 points. Students will pass if they either achieve at least a 7. If they do not have a pass grade but at least 3 points, they are allowed to try a final exam, where the average between their previous grade and the exam grade must be at least 5, accordingly to local regulations. Therefore, there is a strong motivation for doing the projects in a timely manner.

III. Tools

Many tools were employed in order to develop the workflow explored in this paper. In the following subsections, a short description of each one and its usefulness for teaching AOC2 are described. All those tools are currently gracefully provided free of charge.

A. GitHub Classroom

GitHub Classroom [5] is a GitHub initiative to reach out for the education public. It provides automated facilities to distribute starter code, give students feedback, provide automated tests and collect assignments [6]. These tasks were previously executed through a set of scripts called `teachers_pet` [7].

Repositories are hosted in GitHub and can be accessed by the same login and tools that students have for other projects. Teachers may apply for free educational organizations in GitHub, where there is no restriction in the number of private repositories.

B. Travis CI

Travis CI [8] is a Continuous Integration [9] service that integrates with GitHub. As modifications are pushed to GitHub's repositories, Travis CI's hooks clone the repository, then look for a `.travis.yml` file with a build recipe. Figure 1 shows an example of such a recipe. It asks Travis CI to load a Linux image, install GCC, clone the repository and then run "make" on it.

```
sudo: false
language: cpp
compiler:
  - gcc
os:
  - linux

addons:
  apt:
    sources:
    packages:
    - gcc

script:
  - make
```

Figure 1. Example of Travis' recipe file.

The results are presented in Travis CI's Dashboard and are available for all members of the project. Travis CI's website has two versions: https://travis-ci.org for public repositories (i.e. that everybody can clone and fork), and https://travis-ci.com for private repositories. I used the later as students are required to keep their repositories privates to avoid plagiarism.

C. Google Classroom

Google Classroom [10] is a virtual classroom environment that integrates well with the G Suite provided by Google. It provides facilities such as support for notes, assignments, sending email to students, sharing documents, and others.

For the purpose of this work, Google Classroom was used for sharing the link for the projects in GitHub, to answer general doubts about the specification, and to return grades. The later was done in order to simplify the collection of grades in the end of the semester. Another virtual classroom such as Moodle[1] would fulfill these needs as well.

D. Sherlock

The Sherlock Plagiarism Detector [11] is a C program aptly named after Conan Doyle's famous sleuth that identifies similarities among text files by first calculating signatures, and later comparing them. It can be configured for different levels of similarity, and the default setup is to warn if files are 20% or more similar to each other.

Sherlock is not accurate enough to detect all kinds of plagiarism and does not substitute manual inspection. If students change variable names and the ordering of functions in their code, Sherlock may not find it. It does not check against code available in the Internet too. However, Sherlock provides a first assessment of code similarity among students' projects which I found useful as the projects I design are specific enough to make it harder to find ready code elsewhere.

Currently, Sherlock is provided without a specific license.

E. Simpletest and Simplegrade

Simpletest[2] is a Unit Test library in C designed to be lightweight and simple to use. It was developed by my (then) Masters student Vitor Alano de Ataides to help me teach students how to design and apply case tests to their assignments.

It is a single header file with the functions used to test so the students do not need to worry about compiling and linking additional code to theirs. Four functions are designed to just output formatted information about what is expected for each test defined by the user (Figure 2).

```
void DESCRIBE(char* text);
void WHEN(char* text);
void IF(char* text);
void THEN(char* text);
```

Figure 2. Simpletest's formatted output functions. Source: [12].

Where `text` is the string to be displayed. Colors are different for each one of the functions. Simpletest is not structured, i.e., the programmer may use these functions in

[1]https://moodle.org

[2]https://github.com/bundz/simpletest

any order (or not use them at all) for each test, although this specific order is recommended.

The current provided tests are as listed in Figure 3:

```
void isNull(void* ptr);
void isNotNull(void * ptr);
void isGreaterThan(int num, int value);
void isEqual(int num, int value);
void isNotEqual(int num, int value);
void isLesserThan(int num, int value);
```

Figure 3. Simpletest's tests. Source: [12].

For each test, the programmer provides an input which is compared to an expected result. The first two functions do not require the expected result, as they just check for null pointers. The output is printed in green, if the test PASSED, or in red, if NOT PASSED. For the cases where a second parameter is provided, the output also includes the expected value if the test fails. The source code in Figure 4 presents an example for `isEqual()`, which compares two integer values. `KGRN` changes the output color to green, `KRED` to red, and `KNRM` back to normal.

```
void isEqual(int num, int value) {
  if (num == value) {
    printf("%s␣PASSED!\n%s", KGRN, KNRM);
  } else {
    printf("%s␣NOT␣PASSED!\n␣got:␣%d␣==␣%d␣%s\n",
     KRED, num, value, KNRM);
  }
}
```

Figure 4. Simpletest's source code for isEqual(). Source: [12].

Simpletest provides no side effects from tests, just colored output in the standard output. An important hallmark of Simpletest's design is its simplicity. It should not made the process of compiling and linking programs, and should be IDE-agnostic, as to avoid dispersing students from the core of the assignment itself.

Simplegrade [3] is a fork from Simpletest in which I added some extra functions to help grade projects. Each test function gets an extra parameter that is used to calculate a final grade. The global variable `grade` accumulates the grades from passed tests, while `currmaxgrade` stores the maximum grade as if all tests so far were passed. Three functions were added to help with the final grading (Figure 5):

```
int GETGRADE();
int GETMAXGRADE();
void GRADEME();
```

Figure 5. Simplegrade's grading functions.

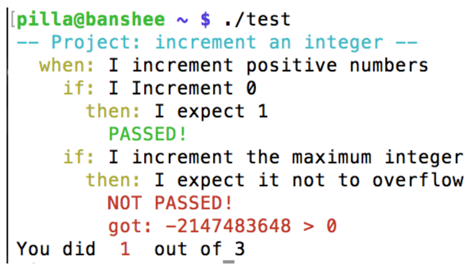
These functions are intended to *(*i) get the current absolute grade, *(*ii) get the current maximum grade as if all tests passed, and *(*iii) grade the tests. The last function prints the grade and the maximum possible grade for the tests in green if it is more than 70% of the maximum grade, or red otherwise. It can be easily changed by modifying one line in Simplegrade. A usage example of Simplegrade is provided in Figure 6.

[3]https://github.com/pilla/simplegrade

```
#include "simplegrade.h"
#include <limits.h>

/** A function that the student should implement,
 *  preferentially in another source file
 */
int increment(int a){
        return (a+1);
}

int main(){
        DESCRIBE("Project:␣increment␣an␣integer");

        WHEN("I␣increment␣positive␣numbers");

        IF("I␣Increment␣0");
        THEN("I␣expect␣1");
        isEqual(increment(0),1,1);

        IF("I␣increment␣the␣maximum␣integer");
        THEN("I␣expect␣it␣not␣to␣overflow");
        isGreaterThan(increment(INT_MAX),0,2);

        GRADEME();

        return GETGRADE();
}
```

Figure 6. Example of grading with Simplegrade.

Figure 7 shows the result of the execution of the code presented in Figure 6. The first test will pass, but the second one will not as there is a untreated overflow.



```
[pilla@banshee ~ $ ./test
-- Project: increment an integer --
  when: I increment positive numbers
    if: I Increment 0
      then: I expect 1
        PASSED!
    if: I increment the maximum integer
      then: I expect it not to overflow
        NOT PASSED!
        got: -2147483648 > 0
You did  1  out of 3
```

Figure 7. Result of the execution of a Simplegrade example.

IV. Workflow

As I expected that some students would not have acquaintance with Git versioning, and most would not been used to Travis CI, the first assignment was a tutorial whose grade was not considered. This first tutorial was a simple factorial calculator. The `README.md` provided presented instructions that included how to setup accounts on both GitHub and Travis, how to get the link for the assignment that would generate a private repository for each student, and what should be done to push modifications to the implementation to GitHub (and Travis). The large majority of students did not have much difficulties to finish the tutorial without further help.

The workflow for each assignment followed these steps:

1) I created a seed repository with the assignment in `README.md`, a C header with required structures

and function signatures, C sources to build tests, a `Makefile` to be used to build the tests, and a `travis.yml` for Travis CI. This repository is then pushed to the previously generated "ufpelaoc2" organization in GitHub.

2) Then, I created a new assignment in GitHub Classroom for individual, private repositories.
3) Afterwards, I shared the link for the assignment in Google Classroom with the required deadline.
4) Some classes were reserved for discussion of the project.
5) After the deadline, I cloned all the repositories and started tests.
6) The first test was Sherlock's for all student files to check for plagiarism.
7) The second test was to copy over my own Makefile and test sources for each repository, and then run `make`.
8) The resulting log from the previous step was sent to all students.
9) Optionally, extra time for corrections was provided for the second assignment.
10) I did a manual correction of the final version of each assignment.
11) The final grade was established.

Travis CI provides an interesting dashboard that can be used to follow students' progress.

V. Results

In this Section, I discuss the results of two of three assignments presented to students. The first assignment was an average latency calculator for memory hierarchies that were being studied in the beginning of the course. A configuration would imply in none to three levels of cache and their latencies, and the main memory's latency. It was a fairly simple assignment that required about 20 to 30 lines of C code. Deadline was set to about 10 days after its initial availability.

The second assignment was a simplified memory hierarchy simulator. It could also include up to three levels of cache, but configurations also included other information such as associativity, block size, and capacity. All levels should use the same writeback policy, all accesses were aligned, and the replacement policy to be used was LRU (Least Recently Used). For this project, students received about 5 weeks.

The class was comprised of 37 students, but only 36 proceeded to register themselves in Google Classroom. In the following discussions, only the 36 students that accessed Google Classroom are considered.

All grades are out of a maximum of 100 points.

A. Participation

The first assignment was returned by 31 out of 36 students, an 86.1% delivery rate. The average grade was 93.58 out of 100 not counting projects that were not delivered.

In the second assignment, 5 out of the 37 students already had missed more than 25% of the classes, thus making it impossible for them to get approved in the class according to local regulations. Only 36.1% of students, 13, delivered code that would compile and run to a grade greater than zero. The main reasons for that seem to be related to the increased difficulty when compared to the first assignment.

An interesting trend is that no student that failed returning the first assignment returned the second one. As the first assignment was very simple and there was more than enough time for its development, I suppose that the second, much more demanding second assignment was probably too hard or not worth the development time for these students.

Although I do not have historic data about return rates in this specific course, the trend is similar to other courses I lecture (such as Operating Systems) for students in Computer Science and Engineering in our University.

B. Grades

Both assignments presented very good grades for the students that returned them. The average grade for the first assignment was 93.58 with a standard deviation of 20.80 points, while the second assignment had an average grade of 97.08 and a standard deviation of 10.12 points.

If we take a look at all grades, 37 out of 44 assignments (or 84%) received the maximum grade. Only 3 assignments were graded less than 70 (8, 23 and 62, respectively).

Currently, I do not have historic data about practical assignments in this class, hence they cannot be directly with previously seen performance.

C. Commits

Figures 8 and 9 show the number of commits by date for both assignments. In the first assignment, as it was simpler and a smaller timeframe was provided for students, commits are well distributed with a peak in the middle of the period between assignment and deadline.

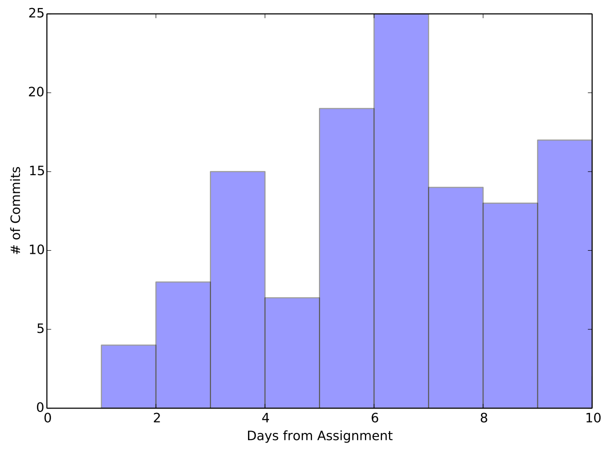


Figure 8. Commits by date for the first assignment.

For the second assignment (Figure 9, there is a clear tendency for commits to be near the deadline, with a large peak during the deadline date (represented by a red line). There were just a few commits before the first month after the assignment was released. There were two distinct hypothesis that could explain it: *(i)* students procrastinated; and *(ii)* students started working without committing their first codes. I am strongly inclined to believe the first hypothesis is correct for most cases, as I provided class time for doubts about the assignment during the entire period and there was almost none in the first month. Notice that some commits happen after the deadline, as the students received an extra week to fix problems in their submissions.

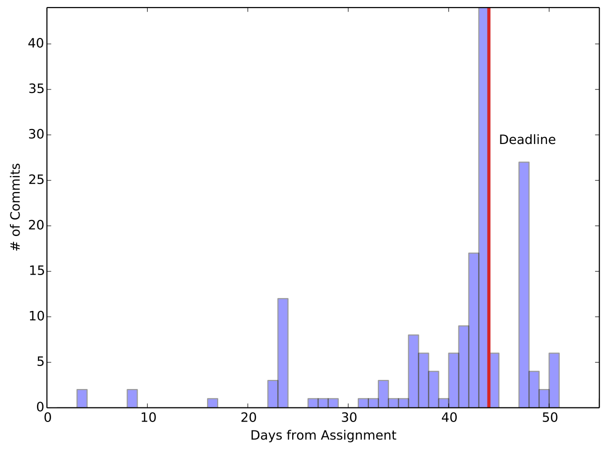


Figure 9. Commits by date for the second assignment.

Figure 10 shows the trend for commits in assignment #2 to be left for the deadline, with the cumulative percentage of commits by date. A conclusion from these data is that I provided too much time for this specific assignment. For future assignments, it may be interesting either to shorten these or require partial submissions. However, the later option would increase my work if I was to check for meaningful submissions.

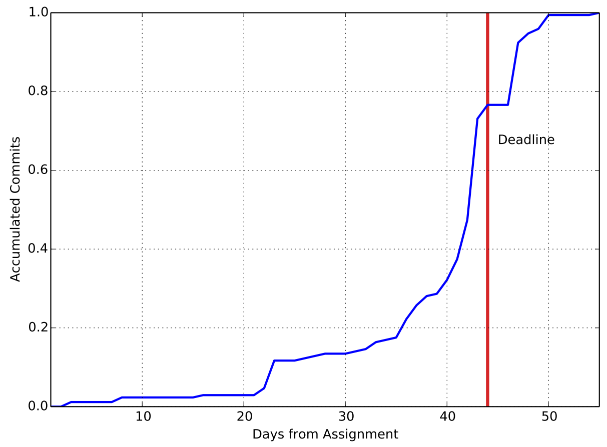


Figure 10. Cumulative commits by date for the second assignment.

Figure 11 presents the number of inserted (green) and removed (red) lines for each commit. The vertical axis represents the students. The largest green circle is a 34K lines debug file added by one of the students to the respective repository, and not actual code. The majority of insertions were around 3K lines, and most times there were less than 100 lines being deleted (with a maximum of 3K lines in the largest red circle). There is only one student that seems to be distributing commits in time, and also finishing before the deadline. Most students concentrated their commits in a few dates in the last two weeks before the deadline. Commits after the deadline were very small and intended to fix bugs that were made clear by my tests. Students did not proceed to overhaul their source codes after the deadline, although there was probably time for more of them to get their assignments to partially work considering that most students seemed to finish the assignment in two weeks.

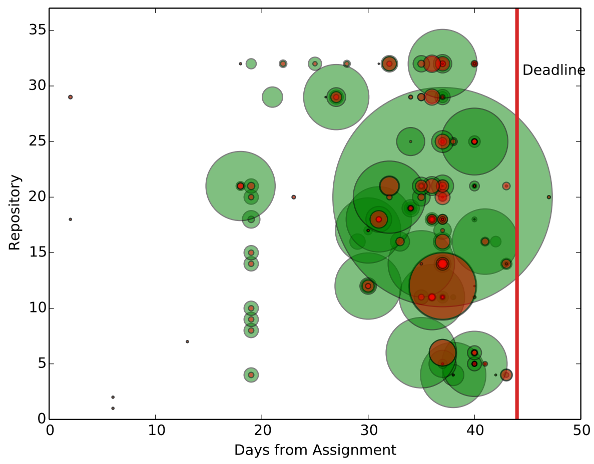


Figure 11. Lines committed by date and student for the second assignment.

VI. Conclusions

In this paper, I discussed preliminary results of teaching Computer Organization and Architecture using GitHub and Travis CI. My experience differs from those reported by Genarelli [1], [2] as I try to provide both automatic testing and a simple, IDE-agnostic environment for students.

The Simplegrade library is distributed with a MIT license and can be found in https://github.com/pilla/simplegrade. The scripts used for management of students' repositories can be found in https://gist.github.com/pilla/.

In future work, I intend to make Simplegrade thread-safe as a safeguard for its application in more complex cases in other courses such as Operating Systems. I also intend to add limits to time and memory used by tests.

Simplegrade is vulnerable to attacks to the global variables that keep grades, but scripts to search for references to it from students' code could easily address it.

I also intend to add results with Operating System classes and more statistics extracted from git logs.

Acknowledgments

I would like to acknowledge the efforts of Github, Travis, and Google to provide their storage and computing resources to educational use free of charge. I would also like to thank Rob Pike and Loki for Sherlock, and my student Vitor Alano de Ataides for the Simpletest library. This work was partially supported by CAPES/Brasil (Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior)

References

[1] V. Gennarelli, "How CS50 at Harvard uses GitHub to teach computer science," 2017, available at https://github.com/blog/2322-how-cs50-at-harvard-uses-github-to-teach-computer-science. Accessed in: July 2017.

[2] ——, "Real-time feedback for students using continuous integration tools," 2017, available at https://github.com/blog/2324-real-time-feedback-for-students-using-continuous-integration-tools. Accessed in: July 2017.

[3] J. C. R. del Pino, "The Virtual Programming Lab for Moodle," 2017, available at: http://vpl.dis.ulpgc.es.

[4] J. Lawrance, S. Jung, and C. Wiseman, "Git on the cloud in the classroom," in *Proc. of the 44th ACM Tech. Symp. on Computer Science Education*, ser. SIGCSE '13. New York, USA: ACM, 2013, pp. 639–644.

[5] GitHub, "Github classroom," 2016, available at: https://classroom.github.com. Accessed in: July 2017.

[6] ——, "Github education," 2016, available at: https://education.github.com. Accessed in: July 2017.

[7] A. Feldman, "teachers_pet," 2013, available at: https://github.com/education/teachers_pet. Accessed in: July 2017.

[8] Travis CI GmbH, "Travis continuous integration," Berlin, 2016, available at: https://travis-ci.com. Accessed in: July 2017.

[9] M. Fowler, "Continuous integration," 2010, available at: https://martinfowler.com/articles/originalContinuousIntegration.html. Accessed in: July, 2017.=.

[10] Google, "Google classroom for higher education," 2016, available at: https://edu.google.com/higher-education/. Accessed in: July 2017.

[11] R. Pike and Loki, "The Sherlock plagiarism detector," 2016, available at: http://www.cs.usyd.edu.au/~scilect/sherlock/. Accessed in: July 2017.

[12] V. A. Ataides, "Simpletest: the simplest C test framework ever," 2016, available at: https://github.com/bundz/simplest. Accessed in: July 2017.