Exploring the use of unit tests, linters, and format checkers to enhance computer-programming instruction

Paul Salvador Inventado
pinventado@fullerton.edu
California State University Fullerton
Fullerton, California

ABSTRACT

It takes a lot of practice to master a complex skill like programming. Computer Science instructors provide as much programming exercises as they can to students, but struggle with the time and effort required for creating problems, checking, and providing feedback on students' solutions. Many instructors have used unit tests, which are programs that compare the output of student programs against expected results, to check for code correctness then manually check coding style and design. We are currently building a framework to explore the use of linters and format checkers in addition to unit tests to check the design and style of code written in C++. These tools catch common issues that significantly speeds up checking. We used the framework to create a programming-problem repository that instructors can use to assign exercises in class. Informal interviews with instructors indicated that the addition of linters and format checkers helped encourage their students to use coding best practices. We have also begun to store log data from the automated checkers. We plan to use these data for analyzing student progress in solving programming problems and gathering insights that can help inform instructor feedback.

CCS CONCEPTS

• **Social and professional topics** → **CS1**; • **Software and its engineering** → **Software testing and debugging**.

KEYWORDS

programming exercises, programming problem repository, unit testing, linter, coding style

ACM Reference Format:
Paul Salvador Inventado. 2019. Exploring the use of unit tests, linters, and format checkers to enhance computer-programming instruction. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Programming is a complex skill that involves several processes. First, it requires a good understanding of several interrelated concepts to design an algorithm that solves a given problem. Second, programmers need to implement their algorithms using a programming language. Third, they need to decipher error messages when they compile or run their code. Finally, they need to modify their code to fix syntactical and logical errors that will then satisfy the requirements of the programming problem.

Students need to practice all these processes to refine their programming skills. Instructors can help students learn by supporting them in each step, but this can be challenging in a classroom setting where many students request help at the same time.

We have been developing an open programming problem repository which contains problems that instructors can use in class and automated tests that can help instructors check and give feedback to students. We are also starting to collect data from students' interaction with the system so that we can investigate how it contributes to student learning and find new ways to support them.

2 AUTOMATED CHECKING AND FEEDBACK

There are several tools that programmers use to help them verify the correctness and design of their code. We investigate the viability of three such tools to support student learning: unit tests, linters, and code formatters.

2.1 Unit tests

Unit tests are used to validate code behavior. Such tests often compare the output of code that is tested against expected output given specific inputs. Figure 1 shows the results of a unit test that checks the behavior of a user-defined *power* function.

```
======================
Running unit test
======================

[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from Power
[ RUN      ] Power.OutputFormat
[       OK ] Power.OutputFormat (2 ms)
[ RUN      ] Power.PositiveBase
[       OK ] Power.PositiveBase (1 ms)
[ RUN      ] Power.NegativeBase
[       OK ] Power.NegativeBase (0 ms)
[----------] 3 tests from Power (4 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (4 ms total)
[  PASSED  ] 3 tests.

======================
Unit test complete
======================
```

Figure 1: The results of a googletest-based unit test that checks the behavior of a user-defined power function against three test cases.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Instructors can use unit tests to check student code by designing test cases that validate important aspects of their code. They can assign points to every test case that passes and zero or partial points to those that fail.

Students can also run unit tests to check their programs against the test cases defined by their instructor. Test results do not only provide students with immediate feedback but also frees up the instructor's time to answer other students' questions.

2.2 Linter

A linter is a tool that diagnoses common programming errors, style violations, or bugs. Figure 2 shows the results of a linter that highlights two potential bugs in the code.

```
======================
Running style checker
======================

6 warnings generated.
6 warnings generated.
7957 warnings generated.
/home/student/Desktop/temp/power-function/problem/power.cpp:5:38:
 warning: statement should be inside braces [google-readability-b
races-around-statements]
for (int a = 1; a < power_input; a++)
                                     ^
                                     {
/home/student/Desktop/temp/power-function/problem/power.cpp:6:8:
warning: The left expression of the compound assignment is an uni
nitialized value. The computed value will also be garbage [clang-
analyzer-core.uninitialized.Assign]
result *= base_input;
       ^
/home/student/Desktop/temp/power-function/problem/power.cpp:4:1:
note: 'result' declared without an initial value
int result;
^
```

Figure 2: Results generated by clang-tidy, a C++ linter, on students' code. Specifically it suggests using curly braces to enclose any code block regardless of length and suggests initializing variables.

Unlike unit tests, linters focus less on coding errors and more on best practices which also allow it to evaluate the design of students' code. Interestingly, they are quite useful for uncovering and fixing logical errors as shown in the example.

2.3 Code formatter

Code formatters are usually used to fix the spacing, indentation, and line length to make code easier to read. Many text editors and IDEs provide this functionality already. Figure 3 shows the differences between a students' code and a preferred code format using a *diff* program.

Code formatters are not problem-specific, but they help make the code more readable and easier to debug. Novice programmers can benefit a lot from this tool because they often forget to indent their code. In many cases, students immediately figure out their errors after fixing code formatting. Instructors can also use this checker as a measure of code readability.

3 OPEN PROGRAMMING PROBLEM REPOSITORY

We have been developing an online repository of programming problems that are distributed with unit tests, linters, and code formatters. We plan to share this repository with other instructors to help them provide more practice problems to their students. More importantly, we hope to reduce the effort required for checking students' solutions and free up instructors' time so they can focus on answering higher-level questions while the tools provide students with low-level feedback.

```
======================
Running format checker
======================

[Checking main.cpp]

[Checking power.cpp]
--- /dev/fd/63  2019-02-27 01:30:24.143714682 -0800
+++ /dev/fd/62  2019-02-27 01:30:24.143714682 -0800
@@ -3,6 +3,6 @@
 int power(int base_input, int power_input) {
-int result;
-for (int a = 1; a < power_input; a++)
-result *= base_input;
-return result;
+  int result;
+  for (int a = 1; a < power_input; a++)
+    result *= base_input;
+  return result;
 }

[Checking power.hpp]
======================
Format checking complete
======================
```

Figure 3: The results of a *diff* program comparing students' code and the preferred way of formatting code.

The repository is hosted on GitHub[1] and contains around 145 programming problems mainly designed for C++. It uses the *googletest*[2] library for unit tests, *clang-tidy*[3] as the linter, and *clang-format*[4] for code formatting.

4 INSIGHTS AND FUTURE WORK

Informal interviews with instructors revealed that they appreciated the amount of time the repository saved them from creating and checking problems on their own. However, they felt less control over the problems unless they modified it on their own. Advanced students liked answering problems at their own pace, but felt the tests limited their coding style. Novice students felt they learned better from additional practice, but sometimes got overwhelmed by the number of problems they had to answer.

We just started collecting data from students' interactions with the three tools. Specifically, we take snapshots of the students' code at the time they run a tool and also keep a log of the results it generates. We plan to use the data to answer the following questions:

- How much time do unit tests, linters, and code formatters save from answering programming problems?
- Are the tests useful or do they disrupt student learning?
- How much time and effort do the tools save for checking and grading student solutions?

[1] https://github.com/ilxl-ppr
[2] https://github.com/google/googletest
[3] https://clang.llvm.org/extra/clang-tidy
[4] https://clang.llvm.org/docs/ClangFormat.html