Automated Evaluation of Programming Assignments – An Experience Report

Sujit Kumar Chakrabarti, Shilpi Banerjee

Abstract

We present the motivation, the important design insights and our experience in implementing a method for automated evaluation of assignments in an introductory programming course being taught in our institute. Our method is simpler to use compared to several existing automation systems for evaluation. Effort of installation is virtually non-existent. Also, it presents some fundamental methodological advantages w.r.t. to existing systems. This facilitates designing more effective test cases. Finally, the approach can be applied to classes of arbitrarily large sizes, and presents no scalability issues when compared to existing systems.

1 Introduction

Programming is a critical skill for almost all technical professions in general and for computer science and information technology in particular. Therefore, most universities take all their fresh batches through an introductory programming course. As an obvious consequence, programming classes tend to be large (100+ students). Teaching programming to such large classes present their very unique problems ranging from instruction, tutoring, course administration and organisation etc. Many of these problems centre around assessment – both formative as well as summative. One of the key elements of making such programming courses effective has been to administer a large number of assessments, in the form of assignments, quizzes, projects and examinations, so that the students get a fine-grained view of their progress through the course, and the instructors can make course corrections if found necessary.

With assessment comes the load of evaluation. Marking answer sheets of large classes is a huge burden, even for small quizzes. Quality parameters of objectivity, fairness and reproducibility have to be maintained on the one hand, and timeliness of feedback has to be adhered to on the other, for the assessment process to be effective. This often puts unrealistic demands on the intructor's time. Having armies of teaching assistants may help, but may not be as straightforward. First of all, getting good TAs in enough numbers is not easy. Further, training TAs about the naunces of evaluation, and tracking the progress and quality of evaluation done by TAs is itself a huge task. In short, administering frequent assessments

to programmming classes may not scale if evaluation is manually done. Automation becomes the only viable alternative.

Automated evaluation is already fairly mainstream in programming courses. A number of online assignment submission systems like Dom Judge, Hacker Earth, Hacker Rank, Code Chef etc. are available and are being widely used both in programming contests as well as in evaluating programming submissions. These systems have proved useful and effective. Some of the issues that authors of this article have experienced with these systems are:

1. Requires participants to write code that takes input and prints output in a particular format. We feel that this format is quite limited. First of all, it requires the participants to write input/output code which may not have anything to do with the main problem they are solving. This approach is cumbersome and error-prone particularly when the input and output are through data-structures which anything but the most trivial. Further, as the testing happens based completely on comparison between strings, the evaluation is very brittle. To some extent, all testing based approaches suffer from this issue. But if the testing is done with more visibility into the inner parts of the program (e.g. the various data-types), it can be designed to be a lot of flexible. We substantiate this point through examples in the later sections.
2. Web-server based deployment is somewhat complicated. For example, DOMJudge requires at least three roles to be defined: administrator, judge and team. It also requires us to reserve a machine to install the server. There are separate elaborate manuals for each of these roles. A more palatable deployment would be a local installation not needing web-servers, databases, root privileges etc.

Because of the issues listed above in existing platforms, we decided to device our own approach to automated evaluation of programming assignments. This paper discusses our main design insights and experiences while implementing and using the automation. In particular, we present the following:

1. A taxonomy of question types that requires test cases to be designed in a specific way
2. A methodology of designing questions based on learning objectives
3. A discipline of designing test cases which test what we wish to assess.

2 Types of Test Cases

From the point of view of test cases, there are a number of types of test cases each catering to a specific testing requirement. We elaborate on this aspect below.

2.1 Functions returning value

The simplest type of test cases are those which test if the output returned by a particular function is correct or not, e.g. whether an implementation

of factorial function indeed returns the correct value, or whether the area of a triangle is computed correctly based on its base and height. In most cases, the value returned from a function of this kind is a simple piece of data which can be directly compared with the expected value.

2.2 Functions with output

There are function (procedures, to be strict) which result in variety of side effects, e.g. writing into the output console. Such functions should be called and their output (console) should captured appropriately.

2.3 Programs with output

If the output of the program is what needs to be tested, then the program should be executed and its output should captured appropriately to be compared against an expected output.

2.4 Structural questions

Some test cases may need to check static aspects of the program. Some of these can be dynamically checked through a cleverly designed test case. However, in general, such features are not open to be tested dynamically, but have to be checked by performing static analysis of the source code. For example: Is class A a base class of class B? Does class B override method m defined in its base class A? and so on. Designing test cases to check such aspects may be easy when the programming language in question provides reflection features (e.g. Java, Python). However, in absence of reflection (e.g. C), checking such features may require access to a parser infrastructure for the language.

3 Designing Questions

An important objective of introductory programming courses is to introduce students to basic programming constructs (e.g. branches, loops, functions, recursion, classes and inheritance), and preliminary application of these to solve *very simple* computing problems. These courses stop short of teaching more advanced program design concepts, e.g. data-structure and algorithm design, which are covered in later courses. This results in a fairly regimented nature of what can be asked and how these questions can be answered. The level of abstraction of questions focuses primarily on programming constructs and not on program design. For example, if a question is about implementation of the factorial function, it will mostly be specified if the solution should use iteration or recursion. Note that the objective of such a question would be to test whether the student has understood such programming concepts as loops or recursions, and not to test his/her understanding of the factorial function, which is either assumed to be known or is given in a mathematical form.

4 Designing Test Cases

5 Evaluating Objective Questions

5.1 Multiple Choice

5.2 Match the Following