Architecture Of An Automatic Program Evaluation System

Amit Kumar Mandal[1], Chittaranjan Mandal[1], Christopher M P Reade[2]

[1]SIT, IIT Kharagpur, WB 721302, INDIA, {amitm, chitta}@sit.iitkgp.ernet.in
[2]Kingston Business School, Kingston University, UK, Chris.Reade@kingston.ac.uk

Abstract— We describe a scheme and implementation for the automatic evaluation of programming oriented assignments. The implementation described covers both the internal working of the automatic evaluation and the web interface. Presently, this system can handle only C programs.

I. Introduction

This paper presents an automated tool that assists the students by automatically evaluating, marking and providing critical feedback for the programming assignments submitted by them. The tool also helps the evaluators by providing easier way to set up assignment statement, develop model solution of the problem, supply user defined inputs. The 'Automatic Program Evaluation System' will be integrated with the Web-based course management (WBCM) tool (Mandal et al 2004). WBCM[1] helps an instructor to upload assignments, put up course material, collect student submission and provide online marks and feedback. The problem of automatic and semi-automatic evaluation have been highlighted several times in the past and a considerable amount of innovative work has been suggested to overcome the problem, in this paper we will be discussing the key approaches in the literature. Although our approach is limited to evaluating only C programs, but we have designed and implemented the system in such a way that both students and the evaluators have minimum burden on their part and as the system can be accessed from the web it helps a lot in distant learning programs.

II. motivation

The scheme was motivated by the need to handle programming assignments with large cohorts of students (approximately eight hundred students). In a semester each student is required to complete about nine assignments and three exams. That amounts to 9600 submissions of programs of varying levels of complexity. Even after the load is distributed among 21 evaluators, each person is required to evaluate over 450 programs per semester. The evaluators are busy most of time in testing and grading work and losing their attention towards setting up useful assignments for the students, as a result of which standard of education is getting affected.

[1]WBCM is a course management tool that is being used at IIT Kharagpur.

III. Related Works

After researching for a considerable amount of time we came to know that a lot of relevant work has been done in the field of automatic and semi-automatic evaluation of programming assignments. Some of the early systems include TRY[1] and ASSYST[2]. Scheme-robo[3] is an automatic assessment system for programming exercises written in scheme programming language, Scheme-robo takes as input a solution to an exercise and checks for correctness of function by comparing return values against the model solution. The automatic evaluation system that is being developed nowadays has a web interface, so that the system can be assessed universally through any platform, these sort of system include GAME[4] and Submit![4] and Juedes' system[5]. Other systems such as that from Baker et al [6], Submit![4] developed a mechanism for providing a detailed and rapid feedback to the student. Systems from Luck and Joy [7], Benford et al [8] are integrated with a course maker system in order to manage the files and records in a better way.

IV. Overview Of The System

A. Our focus and approach

Our focus during the development was to build a flexible system that has the capability to test the assignment from all possible dimensions i.e. testing on random numbers, testing on user specified inputs, testing on the amount of execution time needed by the program. The main focuses during the development are (1) Automatic Evaluation of the Programs i.e. Dynamic as well as Static assessment. (2) Minimum burden on the evaluators as well as the students. (3) Security of the system. (4)Careful Grading of the programs. (5)Returning a quick and detailed feedback.

Our approach is to perform White Box test, instead of, Black box or Grey box testing. White box testing helps the evaluator determine whether the programs have been written in a particular way, following a particular algorithm and using certain data structures

B. Example

Let us start with an example that is very common for a data structures course, for example: MergeSort Program. As the regular structure of a C program consist of a main function and a number of other functions performing different activities, we have decided to break down the Mergesort Program into two functions (1) Mergesort

function - This function performs the sorting by recursively calling itself and the merge function. (2) Merge function - This function will be accepting two sorted arrays as input and then merging them into a third array. As our approach is to perform a white box testing, therefore testing and awarding marks on correct output generated by the program will be of no use because, by that we cannot be sure of the sorting algorithm (Quick sort, Selection sort, Heap sort, Insertion sort etc.) applied by the student.

B.1 Our approach

In this section we will discuss one approach by which we can test the mergesort program. Our aim is to test that, the student writes the mergesort and the merge function correctly. Our strategy is to check each and every step in the mergesort program. Fig 1 explains the general working of the mergesort program. Our approach is to get catch of each and every step i.e. merge #1,2,3,4,5,6,7. This can be done by using an extra two dimensional array of size (n) * (n-1). The idea is to transfer the array contents to this two dimensional array at the end of the merge function. At the end of the program execution, the two-dimensional array will look very similar to fig 2. We can compare this array with the array which is formed by the model solution and if both the array matches exactly, the student will be awarded marks. As we have mentioned earlier that each time the merge function is called the contents of the array should be moved to the two dimensional array. To perform this operation a function is required; this function should be provided with the assignment statement so that the students can reach the submission stage easily. The details of the function are shown in fig 3.

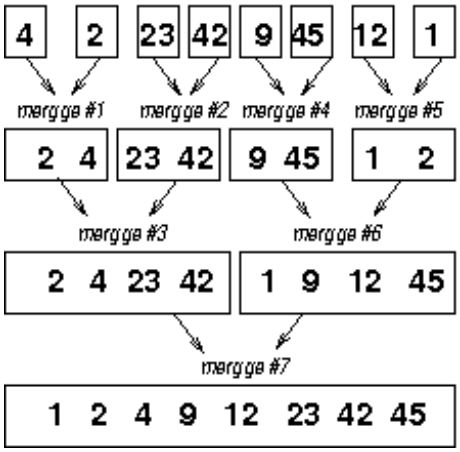

Fig. 1
WORKING PRINCIPLE OF MERGESORT ALGORITHM

B.2 Assignment statement for the mergesort program

Assignment statement is one of the most crucial part of the whole autoevaluation process. If the assignment statement is not setup up in a correct manner, it would be difficult for the student to understand the problem statement properly and most of them will end up with a faulty submission. We have thought up some of the general purpose properties that an assignment statement should have (1) Simple and easy to understand language should be used, so that it becomes easy for the student to understand the problem statement. (2) Prototypes of the function to be submitted by the student must be specified clearly in the statement. (3)Necessary makefile and the auxiliary function should be supplied as a part of the assignment statement to make it easier for the student to reach the submission stage.

	0	1	2	3	4	5	6	7
0	2	4	23	42	9	45	12	1
1	2	4	23	42	9	45	12	1
2	2	4	23	42	9	45	12	1
3	2	4	23	42	9	45	12	1
4	2	4	23	42	9	45	1	12
5	2	4	23	42	1	9	12	45
6	1	2	4	9	12	23	42	45

Fig. 2
TWO DIMENSIONAL ARRAY

```
transfer(int *b,int size,int *a)
  { int i;
    for(i=0;i<size;i++){
      *(b+depth*size+i) = *(a+i); }
    depth = depth+1;
  }
```

Fig. 3
AN AUXILIARY TRANSFER FUNCTION

B.3 Teachers' interaction with the system

The Automatic Evaluation process cannot be accomplished without the valuable support of the evaluator. The evaluator needs to communicate a large number of inputs to the system. Since the number of inputs is large, providing a web interface is not a good idea to accept the values because it is cumbersome both for the evaluator to enter the values and for the system to accept and manage the data properly. Our idea is that, the evaluator will provide the inputs in a single XML file. As the system is to be integrated with WBCM[9], we can use WBCM to upload the XML file from the evaluator. In this XML file the evaluator is required to use predefined xml tags and attributes to specify the inputs. The XML is used to specify the following information: (1)Name of the files to be submitted by the students (2)How to generate the test cases (3)How to generate the makefile (4)Marks distribution for each function to be tested (5)Necessary inputs to carry out static analysis of the program. Apart from the XML file the evaluator is also required to submit the model so-

```
<source_files>
<file name="main_mergesort.c">
<text>File containing the main function</text>
</file> <file name="merge.c">
<text>File containing the merge function</text>
</file> <file name="mergesort.c">
<text>File containing the mergesort function
</text></file></source_files>
```

Fig. 4
XML SPEC FOR CHECKING FILE SUBMISSION

```
<xs:element name = "source_files">
<xs:complexType><xs:sequence>
<xs:element name="file"maxOccurs='unbounded'>
<xs:complexType><xs:sequence>
<xs:element name="text"minOccurs="0">
<xs:simpleType><xs:restriction base="xs:string">
<xs:minLength value="0"/>
<xs:maxLength value="75"/>
</xs:restriction></xs:simpleType>
</xs:element></xs:sequence>
<xs:attribute name="name" type="xs:string">
</xs:attribute></xs:complexType>
</xs:element></xs:sequence>
</xs:complexType></xs:element>
```

Fig. 5
XML SCHEMA VALIDATING XML IN FIG 4

lution for the problem. After the XML file is uploaded, the system will test the validity of the XML file against a XML schema[2]. The purpose of the XML Schema is to define legal building blocks of an XML document. An XML Schema can be used to define the elements that can appear in a document, defines attribute that can appear in a document, defines datatypes of elements and attribute etc. Fig 4 shows only a part of the main XML file that is submitted. This portion of the XML is used by the module that checks for proper file submissions. Fig 5 shows the part of the XML schema that is used to validate this part of the XML file as shown in fig 4.

B.4 Students' interaction with the system

As the system is to be integrated with WBCM, we can use the WBCM to generate dynamic web pages specific to a particular assignment to receive student's submission. The students have the freedom to resubmit the files any number of times within the starting and closing dates of a particular assignment. As we are talking about the multiple submissions, one of the reasons for the multiple submissions during the initial stages is the presence of compilation errors in the student's program. The compilation error may occur because the student has not properly obeyed to the constraints mentioned in the assignment statement. It may be possible that the code works well on the student's computer but whenever he/she is submitting the same to the system, the system is giving compilation error. This may occur due to many reasons for example: the student have not confirmed to the function prototype mentioned in the assignment statement. If at all there is any compilation error the system will immediately provide necessary feedback to the student about the compilation errors. The student has to correct the code and resubmit the programs within the closing date.

[2] XML Schema describes the structure of the XML document, XML Schema is a XML based alternative to DTD

```
random integers:
 (array/single)
 (un-sorted/sorted ascending/sorted descending)
 (positive/negative/mixed)
random floats:
 (array/single)
 (un-sorted/sorted ascending/sorted descending)
 (positive/negative/mixed)
strings:
 (array/single)
 (fixed length/variable length
```

Fig. 6
SUPPORTED INPUT GENERATION OPTIONS

B.5 Input generation and testing The programs

The Automatic Program Evaluation System is a sophisticated tool, which evaluates the program on following criteria: (1)Correctness on dynamically generated random numbers (2)Correctness on user defined inputs (3) Correctness on time as well as space complexity (4)Style Assessment (5)Number of Looping statements (5)Number of Conditional statement. Initially the programs are tested on randomly generated inputs. Evaluators have the option to write down their own routines to generate inputs, alternatively the system provides some assistance in the generation of inputs. Currently the options are supported are shown in fig 6.

The evaluator can express his/her choice of the random numbers on which he/she wants the programming assignments to be tested in the XML file. Fig 7 shows the example of the XML statement that is used to generate an array.

In fig 7 an array(vartype='array') of size 50 (<arraysize>50 </arraysize>)is generated. The array contains positive (range="positive") float (type ="float") ascending(sequence='ascend')values in the range of 10(min= "10") to 5000(max="5000"). Above procedure is iterated 50 (iterations="50") times. Fig 7 also shows that, two user defined inputs have been provided by the evaluator in the XML (source = "included") file itself; these user defined inputs are supplied as it is to the testing procedures. The evaluator has the choice to provide

```
<testing>
<generation type="automatic"iterations="50">
<input vartype="array"type="float"range= "positive"
min="10"max="5000"sequence="ascend">
<arraysize>50</arraysize>
</input>
</generation>
<user_specified source="included">
<input values="6,45,67,32,69,2,4"></input>
<input values="5,89,39,95,79,7"></input>
</user_specified>
</testing>
```

Fig. 7
XML SPEC. FOR INPUT GENERATION

the inputs either directly in the XML file or separately in a text file (if the quantity of inputs is more). The question that arises is about the need for the user defined inputs. As the programs are tested on randomly generated inputs, it may be possible that the programs are not tested for the boundary conditions where the errors are more likely to occur. Therefore to ensure that, the programs are tested for the boundary conditions and other error prone situations, we require some user defined inputs.

V. CONCLUSIONS AND FURTHER WORK

We appreciate our work, as our system has the potential to open up new horizons in the field of Automatic Evaluation of programming assignments by making the mechanism relatively simple to use. We have tried to put our point very simply and explained the whole automatic evaluation process with an example in a very systematic way and ordering the phases serially as they will occur in the practical environment i.e. we have started with deciding the testing approach, then designing the assignment statement, the next step explained is the teacher's interaction with the system, which is followed by student's interaction with the system and at last testing the programs.

Presently we have built the system to work with a single programming language i.e. C language, this is done to avoid our work, from getting weird up. But while developing the system doors have been kept open, so that the system can be upgraded to test programming assignments in other popular languages. In the near future our goal is to extend the system, so that it can evaluate C++ and java programs. As C++ and java compilers are already available on LINUX platform, the system can be upgraded to compile and execute programs in these languages. The evaluator has to supply sufficient information for the preparation of the makefile (for c++ and java programs) in the XML document. Most of the other functions such as checking proper file submission, test case generation, and style analysis for example: Average characters per line, Total Program length, Percentage of space characters, Percentage of blank lines etc. are language independent.

REFERENCES

[1] K. A. Reek, "The try system or how to avoid testing students programs," in *Proceedings of SIGCSE*, pp. 112–116, 1989.

[2] D. Jackson and U. M., "Grading student programming using assyst," in *Proceedings of 28th ACM SIGCSE Tech. Symposium on Computer Science Education*, pp. 335–339, 1997.

[3] R. Saikkonen, L. Malmi, and A. Korhonen, "Fully automatic assessment of programming exercises," in *Proceedings of the 6th annual conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pp. 133–136, 2001.

[4] Y. Pisan, D. Richards, A. Sloane, H. Koncek, and S. Mitchell, "Submit! a web-based system for automatic program critiquing," in *Proceedings of the fifth Australasian Computing Education Conference (ACE 2003)*, pp. 59–68, 2003.

[5] D. W. Juedes, "Experiences in web based grading," in *33rd ASEE/IEEE Frontiers in Education Conference*, Nov 5-8 2003.

[6] R. S. Baker, M. Boilen, M. T. Goodrich, R. Tamassia, and B. A. Stibel, "Tester and visualizers for teaching data structures," in *Proceedings of the ACM 30th SIGCSE Tech. Symposium on Computer Science Education*, pp. 261–265, 1999.

[7] M. Luck and M. Joy, "A secure online submission system," *In Software-Practice and Experience*, no. 8, pp. 721–740, 1999.

[8] S. D. Benford, K. E. Burke, and E. Foxley, "A system to teach programming in a quality controlled environment," *The Software Quality Journal pp 177-197*, pp. 177–197, 1993.

[9] C. Mandal, V. L. Sinha, and C. M. P. Reade, "A web-based course management tool and web services," *Electronic Journal of E-Learning*, no. 1, 2004.