Code Runner Extension
Final Report

by

Dian LIN

ID:6916490

UPI:dlin368

BTech 451
Information Technology
in
Bachelor of Technology

The University of Auckland

Supervisor: Dr. Wannes van der Mark, Dr. Tariq Khan

ABSTRACT

Code Runner provides an on-line platform for students to complete their programming assignments. It supports different programming languages. It is widely used at the University of the Auckland for courses from Stage One to Stage Three. To enable each student to complete their assignments individually, this study aims to research different ways of Code Runner assignments cheating. The goal is to extend Code Runner to have parameterized exercises so that each student is presented with unique exercises. The outcome ideas from research results will be implemented into the platform in order to make cheating more difficult in Code Runner.

Contents

Chapter 1

Introduction

Moodle is an open-source Learning Management System(LMS) [1], and is an open-source platform. At the moment, Moodle is extensively used in schools and tertiary institutions in Australia and New Zealand. It is also widely used in Europe, especially in Spain and the U.K. The core of Moodle is courses that contain activities and resources, including about 20 different types of activities available such as assignments, quizzes, choices etc [2].

At the Department of Computer Science of the University of Auckland, it is used for programming exercises. Programming assignments usually require environments installation such as Java. Moodle provides a way for students to easily complete their assignments without the installation of programming software and environment path setting but can only be used in browsers. Hence, Moodle is one of the most important tools in the Department of Computer Science to test students learning abilities. It can be used to set up coding assignments, multiple-choice questions and simple quizzes. Moodle is able to run the input source code in the background and immediately gives feedback to participants about their submission. For teachers, this activity-based model combines the activities into sequences and groups. It helps teachers to guide participants. Also, Moodle contains auto-grading and group courses functionalities, which saves the efforts of the manual. It is for this reason that Code Runner being widely accepted by people through the department.

1.1 Code Runner

Code Runner [3] is a Moodle question type that requests students to submit program code to some given specification such as Java or Python function. At the University of Auckland, Code Runner is the main reason for using Moodle, as lecturers or administrators can simply post questions by defining question contents and test cases under specific course list as the assignment build-up. The Sandbox is used to run a

series of test cases for input source code under limited time whilst preventing infinite loops or deadlocks from blocking the system. The outputs are compared with expected answers and used to automatically mark the assignment works upon given a marking guide. The use of Sandbox keeps Code Runner safe since it effectively prevents malicious code injection, which is guarding the system.

Every participant should be able to pass all test cases that are pre-defined by question creators in order to gain the full marks. Code Runner becomes a convenient and useful tool, because it helps students getting more and more familiar with testing topics introduced in lectures. And encouraging students to gain more marks in terms of splitting a large assignment into several simple questions.

Moodle provides a large database for Code Runner to store data. Each question creation will create a new record inserted into Moodle database(MySql Database, will be mentioned in the following section), and instead of searching created questions in Code Runner website, authorised people could simply look up and modify every question details by SQL statement. To connect Code Runner with Database Management System(DBMS), Code Runner becomes more feasible to handle its functionality.

The output of different feedback from Code Runner is presented in Figure 1.1.

Figure 1.1: Code Runner output feedback of correct and wrong answers

1.2 MySQL

MySql is an open-source Relational Database Management System(RDBMS), and written in C and C++ [6]. As the advantage of MySql, it is allowed users to create

relationships between tables by primary keys and foreign keys. This feature helps the database become more consistent, with users being able to look for particular data quickly. In Moodle, with lots of data needing to be recorded and tracked, MySql database is a suitable tool to manage the platform.

1.3 Vulnerability to Cheating

Code Runner is an online tool that automatically tests and marks students' assignments by comparing answers to predefined outputs. This is a significant weakness of Code Runner because it is not able to detect cheating. Students could simply copy-paste the source code from someone else who holds the same question and was already passed all test cases. Those students would be able to gain the marks without putting any effort. Assignments cheating is not allowed by the University. Some damaging effects would be caused by Code Runner assignments cheating including:

- Being unfair to hard working students and lecturers:

  Most students in the University put lots of efforts on their assignments, and usually spend time on studying and preparing assignments. It is quite unfair for those students who are hard-working to be getting the same grade as those who are cheating on assignments. Also, lots of effort is put by lecturers to prepare teaching materials and create assignments. If cheating becomes possible, it would be unfair to lecturers as well.

- A loss of faith in Code Runner:

  This would have a significant effect impact on the University. Once cheating becomes a possible way to solve the assignments, the quality of students is not able to be evaluated. The reputation of the University will be reduced due to students losing their faith in Code Runner.

- More students being able to cheat on assignments in Code Runner:

  If cheating is allowed in Code Runner, more students would cheat in order to cut down on time spent on the assignment. As students could get the same grade even they cheat, the learning ability of students is poor. Even though cheating students could pass the course, it does not mean that those students have obtained the knowledge that required by the course.

The basic principle of whatever assignments or Code Runner tests at the University of Auckland is that it encourages students to do individual works in order to obtain the skills and ideas that required by the courses. Therefore, if Code Runner

is going to be treated as one of the main assignment parts, it is necessary to push up barriers against cheating in Code Runner.

1.4 Study Outline

In this project, we will first research on the Code Runner program before idea implementation. In research, useful information will be obtained, and become the guideline to implement ideas properly, so that we will not waste our time on unnecessary coding. Based on research results, the following ideas' implementation would be much simpler and easier because of the corrected guide direction. The workflow indicates what the steps of the project will be:

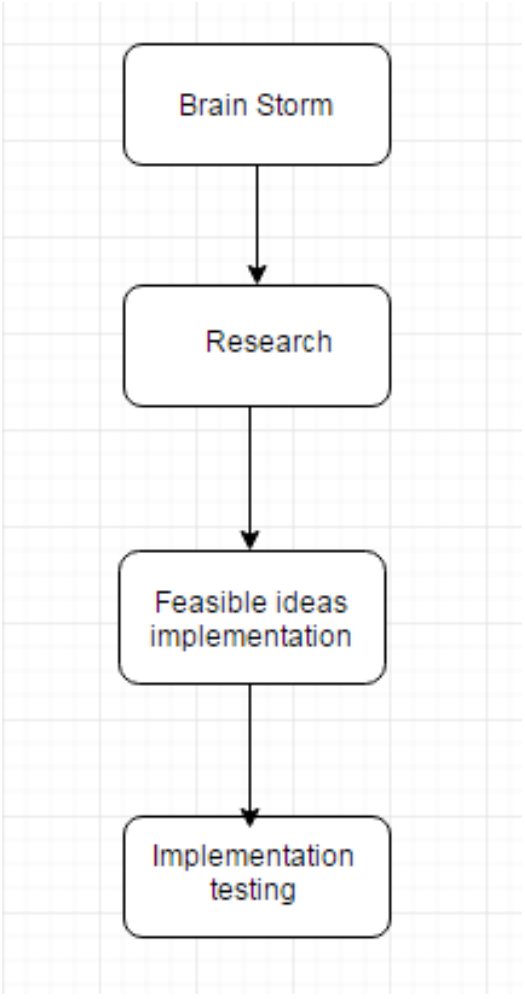

Figure 1.2: Study flow chart

Chapter 2

Brain Storm

Before we starting research and ideas implementation, some of the new ideas should be generated to provide a direction for barriers build-up against cheating.

2.1 Research Part

2.1.1 Similarity Checking without Comments

This straightforward idea is to check the source code for similarity. Due to different courses would have different assignments, the assignment difficult may differ to each other. Based on course requirements, setting up similarity tolerance between source codes is one barrier to against cheating.

2.1.2 Programming Variation

For the same programming language, source code could be possibly found from websites. To determine if the answers to Code Runner questions could be easily found on the website, some research would have to be done.

2.1.3 Question Recycling

Code Runner will store student answers from previous years. If question creators reuse questions which were tested last semester or in previous years, it actually provides a way for current students to copy the answers from previous years from someone else who has done the paper. This is definitely unfair. This anti-cheat idea requires research to see how many or what percentage questions would be reused, then make a decision on whether previous answers should be cleaned or not from Code Runner database.

2.2 Implementation Part

2.2.1 Similarity Checking with Comments

The similarity checking is actually a basic way to identify if participants cheat in assignments. Instead of simply checking the source code of submissions, there is a possibility of requiring students to add comments to their programs. Since comments added depend on personal thoughts, it is highly likely to expect that the similarity would be reduced a lot, despite the size of the assignments being short. In the University, assignments aim to help students to have a better understanding on each concept taught in lectures and improve their learning skills. After this requirement is added into Code Runner, this goal will have been more or less achieved. It is also obvious to see who is cheating in Code Runner. However, the downside of this idea is that, in order to check similarities among more than 2 submissions, each time we will have to check a new submission with the rest of submissions to see the similarity percentage. This is actually a Brute Force approach, so a time penalty will be occurred. Therefore, time penalty and idea performance tests in Code Runner have to be done.

2.2.2 Functionality Addition

Additional functionality could be added into the Code Runner platform (code implementation) so that Code Runner is able to have several options for one question, and that each option has a similar difficulty to the others. Assigning each option to every student, in this case, would result in the students having different questions(options), which reduces the possibility of assignment cheating. The Table 2.1 and Table 2.2 give a layout of ideas.

Based on Table 2.1, the same assignment handout could possibly cause two ways of cheating, copy-pasting and asking for someone else's coding respectively. In both cases, they are easily done. The prevention for copy-paste is to check the similarity between submissions. However, there is no guarantee that similarity between submissions in Code Runner is relatively low, it is necessary to do a research in order to determine if the similarity checking is a feasible way against copy-paste cheating. On the other hand, the prevention of biological cheating is also easily prevented but requires extra time for tutors or lecturers to make assignment interview. Ensure that students are able to answer questions individually.

Based on Table 2.2, different assignment handouts become harder to be prevented but more efficient. Students are looking for similar solutions from someone else or search source code online. The difficulty of similar solutions search is intermediate as we could create as many as possible assignments and assign to everyone. In this case, there is no guarantee that no one will cheat but the probability of cheating will be reduced, and that is what we supposed to achieve. The other cheating way is to search online solution, this cheating way can be also happened in same assignment handout. But rather than search online, if students are being assigned the same assignment, it is more likely to look for the solution around them. The difficulty of search solution online is hard, as people could search different programming languages and

Table 2.1: **Same Assignment**

Cheating ways	Difficulty	Prevention	Changes	Feasibility
Submission Copy and paste	Easy	Similarity checking	Sandbox to check codes similarity	Common way against cheating. Research required
Others who have finished assignment coding for students	Easy	Assignment interview	Extra time required	Further discussion

Table 2.2: **Different Assignments**

Cheating ways	Difficulty	Prevention	Changes	Feasibility
Looking for similar solutions	Intermediate	Everyone has unique assignment	Create more variants in one question	Unique assignment reduces cheating probability
Online solutions or languages translation	Hard	Pre-defined questions preferred	define class used in Code Runner	Research required to see if it's feasible

make language translation. Even though programming translation is an unavoidable personal skill, there is still no guarantee that people with high programming skill will not cheat. Pre-defined questions are much more preferred in this case but it requires research to see if it is feasible.

Chapter 3

Research

Each research for this chapter was conducted in ground lab room, Department of Computer Science, the University of Auckland.

3.1 Similarity Checking Research (without Comments)

3.1.1 Research Process

This study will focus on the coding part in Code Runner. Most Code Runner users are Stage One students, and programming coding is the main part for those Code Runner students. In order to reduce the research error, the obtained dataset should follow some basic rules including:

- People from different backgrounds who did not know each other.
- Source code obtained should be comparable.
- Source code with different lengths should be selected including longest and shortest ones.

3.1.2 Analysis

In order to compare the similarity between different source codes, CopyScape [4] is used to determine the similarity in percentage.

The shortest code length which is 3 coding lines is firstly being checked, the similarity is 100%, which is normally acceptable due to short program length. Then different source codes with different lengths are compared, the average similarity percentage

reaches 77%, while the longest program gives the similarity with around 56.5%. There is a high probability that Code Runner submissions are similar even though student are not cheating. The summarized statistic table is presented below:

Table 3.1: Summarized Table

Program Length	Similarity Percentage
3 (shortest)	100%
3 lines - 67 lines	68% - 77%
67 lines	56.5%
Median length: 30 lines	Median similarity: 76.7%

Therefore, students who use Code Runner for assignments normally write very short programs. We determined that the median coding length in Code Runner is around 25 to 30 lines. This implies that high percentage similar of the submissions would occur. Furthermore, all courses are using the same question for all participants. Since students in each course were lectured by same lecturers, reading the same slides, attend same tutorials and see same examples, it is highly likely to have the high similarity coding results if the median program length is relatively short. In this case, even though students are not cheating each other, a high similarity is still possible.

3.1.3 Conclusion

Similarity checking without comments in Code Runner server is not a feasible anti-cheating approach. A high similarity between submissions due to short program length, questions are same and the solution is mostly based on lecture topics.

3.2 Programming Variation Research

3.2.1 Research Process

Three example types of questions were selected for our research. Instead of random selection, we selected questions from the typical topics, such as the recursive question, pre-defined question. Also, three common but popular sorting ways, Bubble-sort, Insertion-sort and Merge-sort, which is an important topic, were selected as well.

After question selection, we started searching the solutions or available sources on the website. Although Python is used for Stage one courses coding, different solutions of programming languages were still searched. First of all, three sorting algorithms were targeted, the answers(source code) that presented below and would be able to pass to Code Runner questions as these codes have passed all test cases that we used

to test the code accuracy. Three search terms and quantitative data of hits to pages are presented in Figure 3.1, Figure 3.2, Figure 3.3.Three searched source codes are presented in Figure 3.4, Figure 3.5, Figure 3.6 [5].

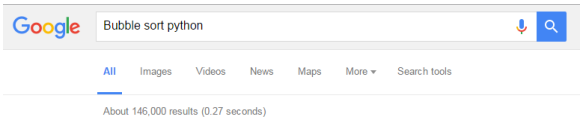

Figure 3.1: Search term and hits to pages for Bubble-sort

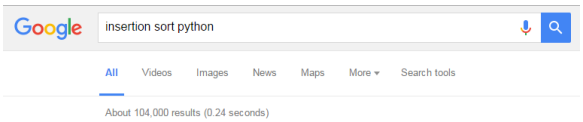

Figure 3.2: Search term and hits to pages for Insertion-sort

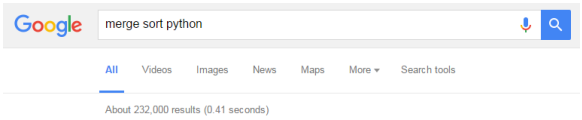

Figure 3.3: Search term and hits to pages for Merge-sort

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
```

Figure 3.4: Searched source code for Bubble-sort

Secondly, solutions of the recursive algorithm for a sum-up question has been searched from the website. It provides the solution and explanation from more than one programming language for the recursive algorithm of sum-up including Python, Java,etc. Research term and hits to pages are presented in Figure 3.7. The Source code is presented in Figure 3.8 [5].
However, solutions of pre-defined questions were not easy to search as each pre-defined question requires using the same method and variable names from class which has already defined by question creators.

```
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

Figure 3.5: Searched source code for Insertion-sort

```
def mergeSort(alist):
    print("Splitting ",alist)
    if len(alist)>1:
        mid = len(alist)//2
        lefthalf = alist[:mid]
        righthalf = alist[mid:]

        mergeSort(lefthalf)
        mergeSort(righthalf)

        i=0
        j=0
        k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                alist[k]=lefthalf[i]
                i=i+1
            else:
                alist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            alist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
            alist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Merging ",alist)

alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)
print(alist)
```

Figure 3.6: Searched source code for Merge-sort

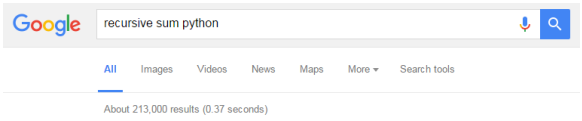

Figure 3.7: Search term and hits to pages for sum recursive

3.2.2 Analysis

Based on the result of research and source obtained, students were asked to solve the questions that based on lecture topics in Code Runner. Because questions could be popular or rare, solutions of popular topics are easily found out while rare ones could

```
def listsum(numList):
    if len(numList) == 1:
        return numList[0]
    else:
        return numList[0] + listsum(numList[1:])

print(listsum([1,3,5,7,9]))

```

Figure 3.8: Searched source code for recursive

only obtain the general description. By considering three sorting ways in the research, the full source code can be found on the websites. it is easy to do programming languages translation by translating known code to other languages. It reveals that programming languages translation is an unavoidable personal skill which is not able to be detected by Code Runner. One the other hand, if we look at the research result of pre-defined questions. Code Runner actually defines its own class for students to solve the question based on the unique variables and method names which were already provided by questions. In this case, it is highly unlikely to get the full source code but conception from the Internet, which is helpful for Code Runner participants as it encourages students to solve the problems independently by reviewing the lecture slides, recordings, and try to understand the provided class.

3.2.3 Conclusion

Answers to Code Runner questions are possibly being obtained on the websites. The programming languages translation skill cannot be detected by Code Runner. Pre-defined questions become a preferred question creation way in Code Runner, it prevents students from searching source code online, instead, encourages participants to do more revision and learn themselves (individual works). This implies that pre-defined type is another barrier against cheating, and should be widely used in Code Runner. However, it can't prevent copy-paste cheating, hence, functionalities should be implemented into Code Runner platform.

3.3 Questions Recycling

3.3.1 Research Process

In case of this research section, lab assignments of one Computer Science course in Code Runner are used as the research objects. For each lab assignment of each semester, the general question ideas have been noted. And question similarity check will be done after practical research has been finished. There are 8 lab assignments and 10 lab assignments are all noted for Summer School and Semester 1 of 2015 respectively. Also, test cases for each question are noticed.

3.3.2 Research Data

There are totally 40 questions for Summer School lab assignment and totally 41 questions for Semester1 lab assignment in Code Runner. By analyzing the noted data, there are 35 questions are exactly matching, in other words, around 85% of questions have been reused. Furthermore, 34 questions used the exactly same test cases, and only one reused question has been redefined its test cases. Also, for the rest of 6 questions, 3 of them actually reuse the answer from previous question. The data are presented in Table 3.2 and questions reviewed are presented in Figure 3.9 and Figure 3.10.

Table 3.2: Question and Testcases Reused

	No. of Questions	Percentage
Question reused	35	85%
New defined question	6	15%
Total questions	41	100%
Test Cases reused	34	83%
New defined test cases	7	17%
Total test cases	41	100%

3.3.3 Analysis

Based on the research result, it is highly likely to see that most question creators reuse the question data from previous years. Also, it believes that question creators would simply move a number of questions from pre-defined Question Bank without changing question definition or background test cases. It not only makes question creation easier, instead, it actually provides an invisible way for cheaters to get an unfair pass.

3.3.4 Conclusion

Therefore, in order to efficiently prevent cheating in Code Runner, it is necessary to clean the Code Runner database records from previous years, especially answers. Or at least, students views are not able to display their codes any more after the semester ends. So that students cannot use these answers to cheat in Code Runner.

```
Lab 01:
Q1 prime number check
Q2 sum value
Q3 count_vowels
Q4 reverse list
Q5 check anagram
Q6 check repeat

Lab 02:
Q1 create a Person object storing name and age
Q2 create Rectangle constructor
Q3 perimeter of rectangle
Q4 area of rectangle
Q5 statement of actually object creation
Q6 print rectangle in a x b format
Q7 rectangles in same shape checking
Q8 rectangles addition

Lab 03:
Q1 square_root_divide(x,y) exception handle
Q2 keyError exception handle
Q3 IndexError and TypeError exception handle
Q4 IOError exception handle
Q5 ValueError exception handle

Lab 04: Peerwise assignment(no permission to access)

Lab 05:
Q1 Stack:push() pop() peek()
Q2 Stack: balanced_brackets test
Q3 Stack: postfix expression
Q4 Queue: enqueue() dequeue()
Q5 Stack and Queue: reverse Queue order
Q6 create link chain

Lab 06:
Q1 complete add() in Linkedlist
Q2 complete remove_from_tail() in Linkedlist
Q3 complete print_iterator_explicit() in Linkedlist
Q4 define Odds class and OddsIterator class
Q5 define Squares and SquaresIterator class

Lab 07:
Q1 sum-up(recursion)
Q2 print_between (recursion)
Q3 find maximum number in list(recursion)
Q4 mirror or palindrome test (recursion)
Q5 define choose() function (recusion)

Lab 08:
Q1 define bubble_sort() and swap() functions to sort a list in DECREASING order
Q2 define merge_sort() //different testcases noticed
Q3 create small tree
Q4 create bigger tree
Q5 sum all tree nodes
```

Figure 3.9: Question revision for Semester one of 2015

```
Lab 01:
Q1 check vowel
Q2 sum value
Q3 reverse list
Q4 check anagram
Q5 check repeat

Lab 02:
Q1 create a Person object storing name and age
Q2 create Rectangle constructor
Q3 rectangle perimeter and are calculation
Q4 print rectangle in a x b format
Q5 statement of actually object creation
Q6 rectangles in same shape checking
Q7 rectangle addition
Q8 put each student details in a list

Lab 03:
Q1 square_root_divide(x,y) exception handle
Q2 keyError exception handle
Q3 IndexError and TypeError exception handle
Q4 IOError exception handle
Q5 break the program

Lab 04: Peerwise assignment (no permission to access)

Lab 05:
Q1 Stack:push() pop() peek()
Q2 Stack: balanced_brackets test
Q3 Stack: postfix expression
Q4 Queue: enqueue() dequeue()
Q5 Stack and Queue: reverse Queue order

Lab 06:
Q1 complete add() in Linkedlist
Q2 complete remove_from_tail() in Linkedlist
Q3 complete print_iterator_explicit() in Linkedlist
Q4 define Odds class and OddsIterator class
Q5 define Squares and SquaresIterator class

Lab 07:
Q1 sum-up(recursion)
Q2 print_between (recursion)
Q3 find maximum number in list(recursion)
Q4 mirror or palindrome test (recursion)
Q5 define choose() function (recusion)

Lab 08: HashTables, not involved by 2015s1.

Lab 09:
Q1 define bubble_sort() and swap() functions to sort a list in ASCENDING order
Q2 reuse bubble_sort() by considering call times
Q3 define merge sort() //different testcases noticed

Lab 10:
Q1 create small tree
Q2 create bigger tree
Q3 calculate size of tree
Q4 depth of tree
Q5 find minimum and maximum node values
```

Figure 3.10: Question revision for Summer School of 2015

Chapter 4

Anti-cheat Idea Description

4.1 Personalized Assessment

The idea here is to create more than one option for each question creation, and each option contains different variants. For example, lecturer or administrator could create Boolean question with options Even and Odd, for each option, has its own test cases and expected answers. Then assigning options to different students to make sure every student is offered different assignments. In this case, the probability of assignment cheating will be reduced, and one barrier against cheating is pushed up. The requirements of the implementation are shown below.

Requirements:

- Even difficulty for assignment variants.
- Being easy to use.
- Maintain the functionality that Code Runner normally has.
- Being possible to integrate Moodle/CodeRunner.

4.2 Similarity Checking with comments

We have shown that similarity checking is not a feasible way to detect plagiarism due to limited size of coding assignments in Code Runner. However, comments could be used to add more distinction.
This idea aims to check the similarity between commented submissions from students and output feedback in percentage for each newest submission. After all submissions have been submitted, a brief summary table of similarity would be generated, and display the highest similarity percentage of each submission by comparing to the rest of ones.

Chapter 5

Mock Up

The most promising idea to implement for Code Runner is shown in Chapter 4 "Personalized assessment" Section to make sure every participant gets different questions in Code Runner. In order to realize this idea, more functionalities should be added to question creation page in Code Runner.

5.1 Proof of Concept

The idea above relates to question creation page but other functionalities are possible to remain the same as what they used to be in Code Runner. Since Code Runner is a large question type in Moodle, the modification in question creation page would cause impacts on relative classes and variables. Hence, lots of works will have to do in order to build up the barrier. As so far, no evidence shows that if the implementation will make it work in Code Runner, thus a prototype test should be made before moving to Code Runner. Here we will use Java to build the prototype.
For the prototype, the GUI mock-up should be same as Code Runner but additional 'Option' input area for question creators. Also, it might be good to keep question options in the teachers' interface to show all question variants. While student interface should be equivalent as Code Runner presents currently.

Java is chosen as the language to build up the prototype, as an open source library in Java, JComponent is used to build the interface. The GUI can be presented properly using two components of JComponent, JFrame and JPanel. Java connection library allows Java programming to easily and fast connect to MySql database that Code Runner uses to maintain all question variants. Since building up a connection between MySql database and prototype is the most significant step.

The following images show the interface of question creation that presents in Code Runner, which is what the prototype supposes to have as the outcome.

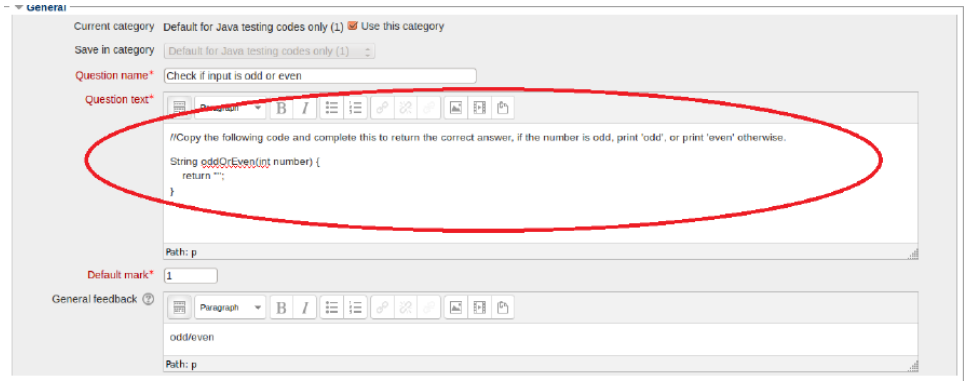

Figure 5.1: Question Creation page in Code Runner

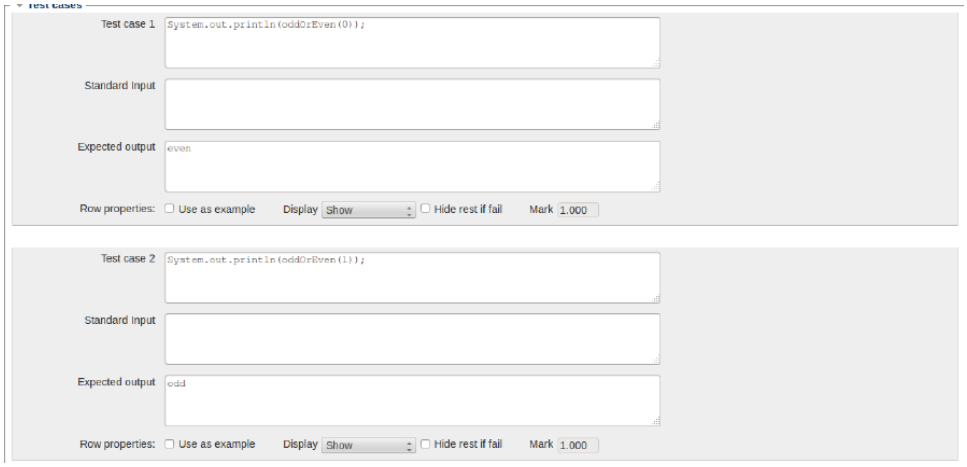

Figure 5.2: Question Creation page in Code Runner

As what have presented above, question name, question description and test cases are all necessary parts that will be inserted into MySql database for students answer comparison. Therefore, the prototype not only has to present proper GUI mock-up, but also make it functional, which indicates that all inputs from question creation text area in GUI mock-up should be inserted into corresponding tables as what Code Runner does, and should be able to run students input as Java programming.

5.2 GUI Mock-up Implementation

JFrame and JPanel in Java library are mainly used to present interface of question creation, “Option” functionality is added into the interface, and for each option created, the variants records for the option will be inserted into local table (JTable, not MySql database at the moment) which allows teachers to view all variants of corresponding options created. Furthermore, “Preview” button will bring teacher view to student view(the interface from student version) for further testing. The prototype

interface shows below.

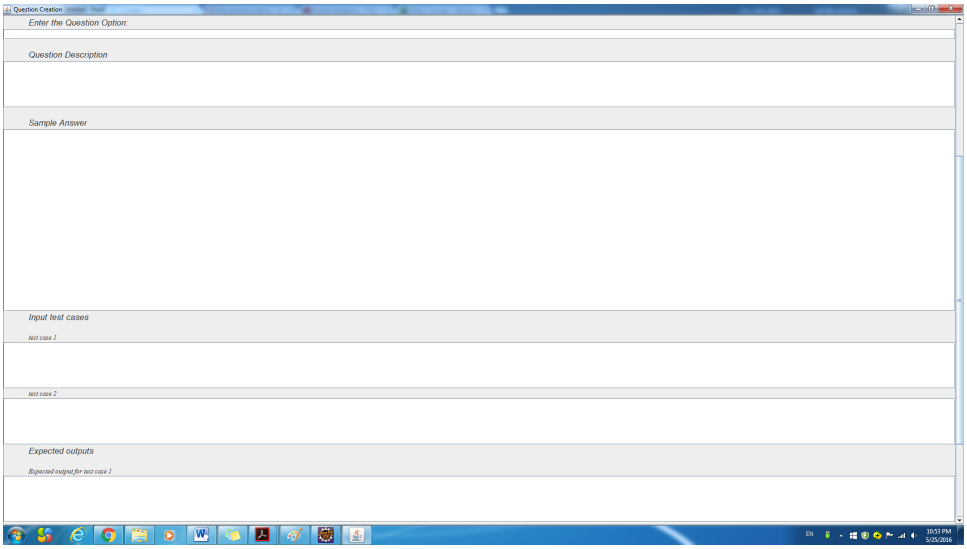

Figure 5.3: Question Creation page at GUI mock-up

The variants in teachers view (local database) will be stored once option created and displayed at question creation page, shows below(Odd and Even options as the examples).

Question table below after first question being created

A	B	C	D	E	F	G
Option	Question Descripti...	Sample Code	TestCase1	Expected output1	TestCase2	Expected output1
Odd	This is Odd test	public static boole...	System.out.println(...	false	System.out.println(...	true
Even	This is even test	public static boole...	System.out.println(...	true	System.out.println(...	false

Figure 5.4: Local variants database(JTable)

The student view can be shown up by “preview” function which is also available in Code Runner. The emulated student assignment view shows in Figure 5.5.

After GUI mock-up creation, the basic framework of Code Runner prototype is successfully implemented, and ready to build functionality into GUI mock-up so that it is able to deal with the input data and output comparison as what Code Runner does.

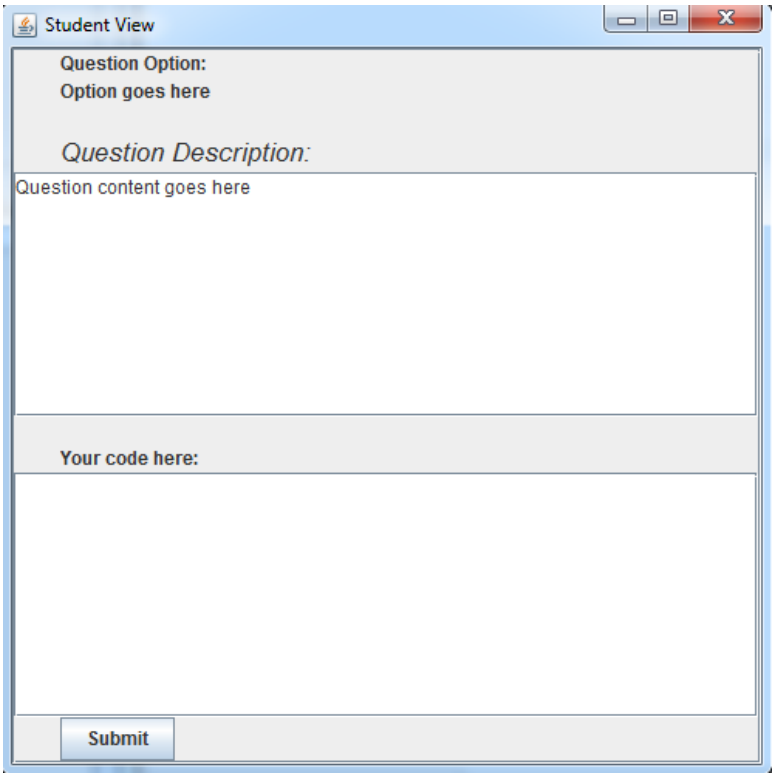

Figure 5.5: Student question view

5.3 Functionality Build-in

5.3.1 Preparation

Before implementing the functionality, MySql database schema and relationships need to be reviewed, as all data display in Code Runner are retrieved from the database. By adding questions to a specific course, 4 tables are modified, which indicates that those 4 tables are related to question creation and answers comparison. Because the expected outputs are also defined in question creation part, thus when implementing functionality build-up, question variants inputs should be treated as records and inserted into corresponding tables as Code Runner did for further discussion. The following 4 circled tables in MySql database will be used.

5.3.2 Functionality of Prototype

DataBase Schema

The database schema of 4 tables above are needed, to insert records into the corresponding tables. The following 4 tables(Table 4.1-Table 4.4) present the schema for each useful header that probably be used in further.

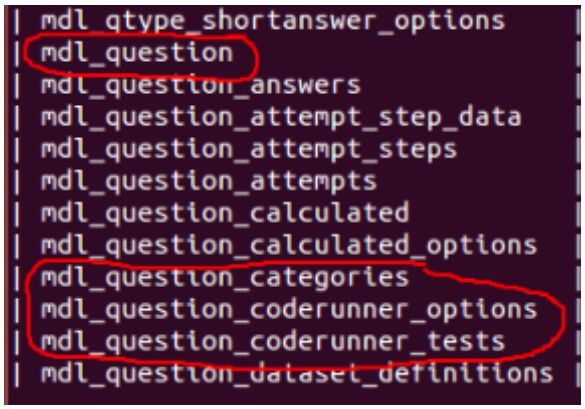

Figure 5.6: MySql database tables

Table 5.1: mdl_question schema

Field	Type	Null	Key
id	bigint(10)	No	PRI
category	bigint(10)	No	MUL
name	varchar(255)	No	MUL
questiontext	longtext	No	
qtype	varchar(20)	No	

Table *mdl_question* above gives several headers that used in question creation part. This is the question table which maintains all questions id including Code Runner questions etc. Header *category* indicates the course number in Code Runner such as Compscixxx. Headers *name* and *questiontext* give the question name and question description. The *qtype* in this project is defined as keyword "CodeRunner".

Table 5.2: mdl_question_categories schema

Field	Type	Null	Key
id	bigint(10)	No	PRI
name	varchar(255)	No	
contextid	bigint(10)	No	MUL

Table *mdl_question_categories* gives the course description with *id* as primary key and maintains the course name.

Table *mdl_question_coderunner_tests* links the question test cases to correspond-

Table 5.3: mdl_question_coderunner_tests schema

Field	Type	Null	Key
id	bigint(10)	No	PRI
questionid	bigint(10)	No	MUL
testcode	longtext	YES	
stdin	longtext	YES	
expected	longtext	YES	
mark	decimal(8,3)	No	

ing questions by building up the relationship chain. Header *testcode* stores the test cases. *stdin* and *expected* headers indicate the sample answer and expected output for each test case respectively. The header *mark* in Code Runner question type is using "allornothing" model which means students are able to gain full mark only if their submission passes all test cases, otherwise zero.

Table 5.4: mdl_question_coderunner_options schema

Field	Type	Null	Key
id	bigint(10)	No	PRI
questionid	bigint(10)	No	MUL
coderunnertype	varchar(255)	NO	MUL
prototypetype	tinyint(1)	No	MUL
allornothing	tinyint(1)	NO	

Table *mdl_question_coderunner_options* gives the options for question creation in Code Runner. The *coderunnertype* provides the options of programming languages such as Java, Python. The *prototypetype* normally is defined as 1 if the question is created and *allornothing* is the one that has been declared above.

The Entity Relationship Diagram(ERD) for above four tables are presented in Figure 5.7.

After related schema have been checked, the functionality is now ready to implement.

Functionality Implementation

Based on the database schema above, in the prototype, necessary SQL should be coded to make sure that each button click will lead to records insertion or deletion

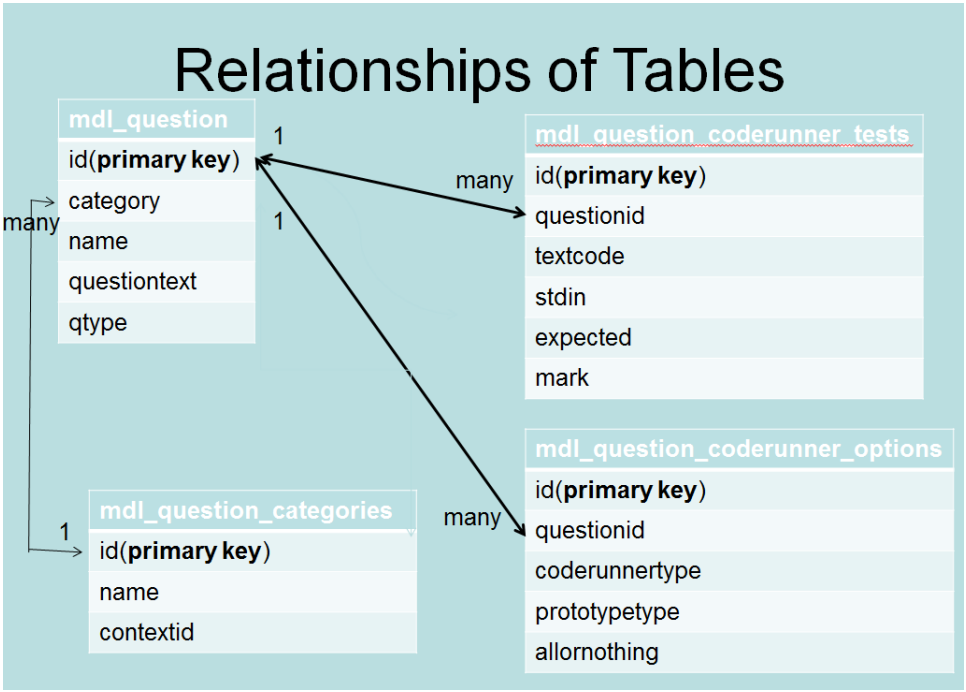

Figure 5.7: Relationship of tables

from MySql Database. For example, for each time the "Save" button is clicked, the SQL presented in Figure 5.8 will be executed by MySql Database connection to insert input variants. Where "Default for CS101" is the already existed course in Code Runner.

For "Deletion" button clicked, corresponding records in the local database and

```
        try{
                stmt = conn.createStatement();
                String insertCategories = "INSERT INTO mdl_question_categories (name,contextid,info) " +
"VALUES ('Default for CS101','17','The default category for questions shared in context firstTesting.')";
                String insertQuestion = "INSERT INTO mdl_question
(category,name,questiontext,questiontextformat,generalfeedback,qtype) " + "VALUES ('17','"+option
+"','"+questionDescription+"','1','odd/even','coderunner')";
                stmt.executeUpdate(insertCategories);
                stmt.executeUpdate(insertQuestion);
                String getQuestionid = "SELECT id FROM mdl_question ORDER BY id DESC LIMIT 1";
                rs = stmt.executeQuery(getQuestionid);
                if (rs.next()){
                        questionid = (int)rs.getLong(1);
                }
                String insertCrOptions = "INSERT INTO mdl_question_coderunner_options
(questionid,coderunnertype)" + "VALUES('"+questionid +"','java_method')";
                String insertCrTest1 = "INSERT INTO mdl_question_coderunner_tests
(questionid,testcode,expected)" + "VALUES('"+questionid+"','"+case1+"','"+out1+"')";
                String insertCrTest2 = "INSERT INTO mdl_question_coderunner_tests
(questionid,testcode,expected)" + "VALUES('"+questionid+"','"+case2+"','"+out2+"')";
                stmt.executeUpdate(insertCrOptions);
                stmt.executeUpdate(insertCrTest1);
                stmt.executeUpdate(insertCrTest2);
        }catch (SQLException se){
                se.printStackTrace();
        }
```

Figure 5.8: insertion SQL in java program

MySql Database server should be both removed. The SQL statement presented in Figure 5.9 will be executed by MySql Database server to delete the selected record.

Furthermore, refer to Figure 5.5, the student's input should be added into a runnable

```
        try{
                stmt = conn.createStatement();
                String getQuestionid = "SELECT id FROM mdl_question ORDER BY id DESC LIMIT 1";
                rs = stmt.executeQuery(getQuestionid);
                if(rs.next()){
                        questionid = (int)rs.getLong(1);
                }
                String getCategoryid = "SELECT id FROM mdl_question_categories ORDER BY id DESC LIMIT 1";
                rs = stmt.executeQuery(getCategoryid);
                if(rs.next()){
                        categoryid = (int)rs.getLong(1);
                }
                int categoryRecordToDelete = categoryid - deleteClicked - saveClicked + Integer.parseInt
((String)table.getValueAt(rowNumber,0));
                int questionRecordToDelete = questionid - deleteClicked - saveClicked + Integer.parseInt
((String)table.getValueAt(rowNumber,0));
                String deleteCategories = "DELETE FROM mdl_question_categories where
id='"+categoryRecordToDelete+"'";
                String deleteQuestion = "DELETE FROM mdl_question where id='"+questionRecordToDelete+"'";
                String deleteCrOptions = "DELETE FROM mdl_question_coderunner_options where questionid='" +
questionRecordToDelete + "'";
                String deleteCrTests = "DELETE FROM mdl_question_coderunner_tests where questionid='" +
questionRecordToDelete + "'";
                stmt.executeUpdate(deleteCategories);
                stmt.executeUpdate(deleteQuestion);
                stmt.executeUpdate(deleteCrOptions);
                stmt.executeUpdate(deleteCrTests);
        }catch (SQLException se){
                se.printStackTrace();
        }
```

Figure 5.9: deletion SQL in java program

Java program once "submit" button is clicked. The prototype should be able to run students input at the background and bring the feedback immediately. By checking the way that Code Runner did is to create a new Java file contains student's input and run the file at the Sandbox, which is presented in Figure 5.10.

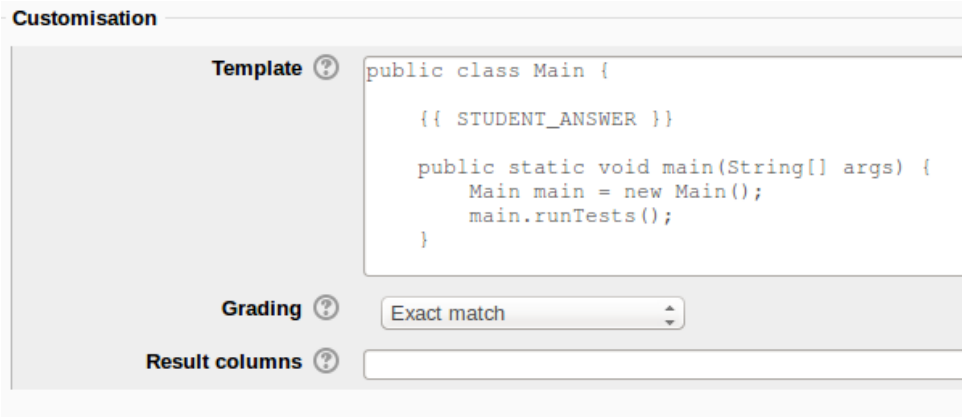

Figure 5.10: deletion SQL in java program

In order to follow the way that Code Runner did, the codes presented in Figure 5.11 will create a java file which contains student input and executed it at the Java Running Environment(JRE). Finally, the program will retrieve the expected answer by unique questionid from MySql Database server and compare answers in order to bring the feedback to the terminal.
By now, it seems proof-of-concept process has been done, all required user interfaces

```
                                        try {
                                            writer = new BufferedWriter(new OutputStreamWriter(
                                                  new FileOutputStream("StudentAns.java"), "utf-8"));
                                            writer.write("public class StudentAns{\n"+studentAns+"\npublic
static void main(String[] args){"+testcase1 +"}}");
                                        } catch (IOException ex) {
                                            System.out.println("Exception caught");
```

Figure 5.11: Java file making code in java program

and functionality have been implemented in Java. The next step should bring the proof-of-concept program to testing part in terms of checking MySql Database records and answers comparison results.

Chapter 6

Prototype Testing

In order to test whether question variants from text area of GUI mock-up are successfully inserted into MySql Database server and the deletion works, we have to go back to corresponding tables for records checking.

6.1 Insertion Testing

Based on GUI mock-up, Figure 6.1 shows the pen-testing input for question creation. After all variants from text areas are saved, MySql Database shows that records from corresponding tables that declared above have been modified. Inserted records have been circled in Figure 6.2. Refer to input variants, the inserted records are the data from the question that we just created. Therefore, the insertion has been successfully completed.

6.2 Student Answer Testing

After the question has been created and inserted into the server, we go to student question view to check whether the program is able to create Java file contains student answer and run the new student Java program at the background. Figure 6.3 presents the testing code which refers to the question that just created. Afterward, a new Java file contains the testing code has been created and presented in Figure 6.4.

After the "Submit" button that displays is clicked in Figure 5.3, the feedbacks bring back to the terminal for both correct and incorrect answers, which is presented in Figure 6.5 and Figure 6.6 respectively.
By now, variants creation and partial sandbox functionality(read and execute students' input) are both working well and able to connect MySql Database server, since the created variants are able to present to students and insert to the database tables.

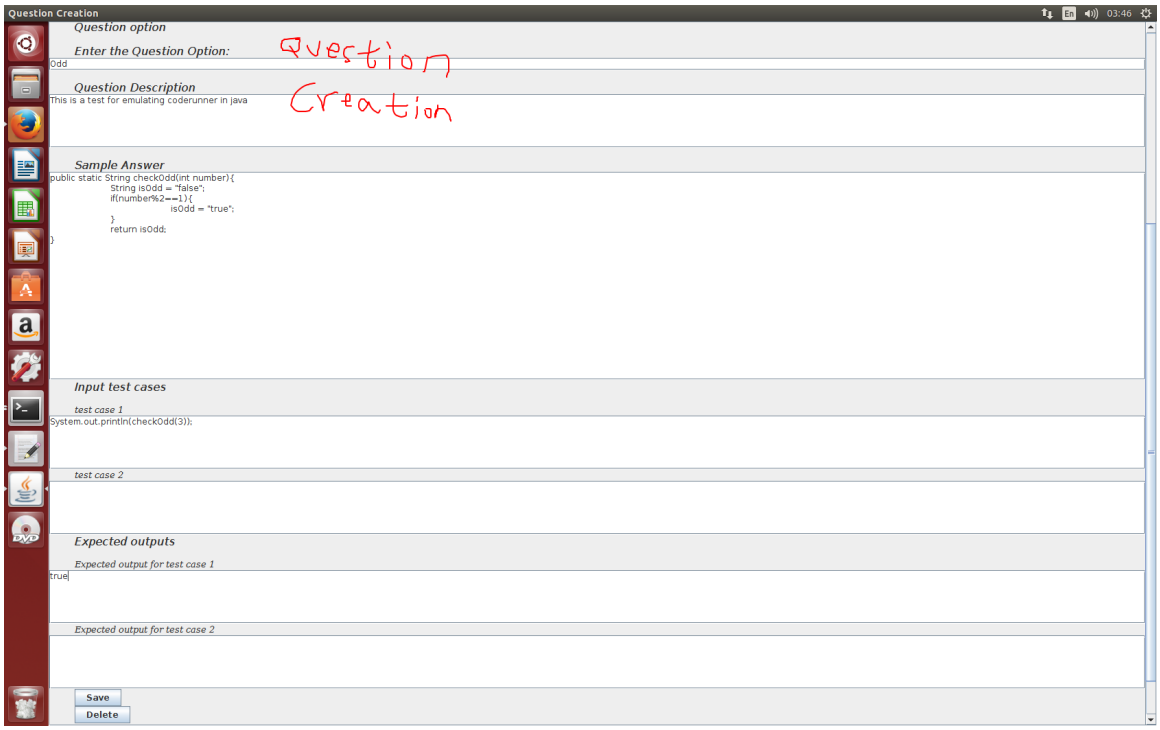

Figure 6.1: Testing variants input

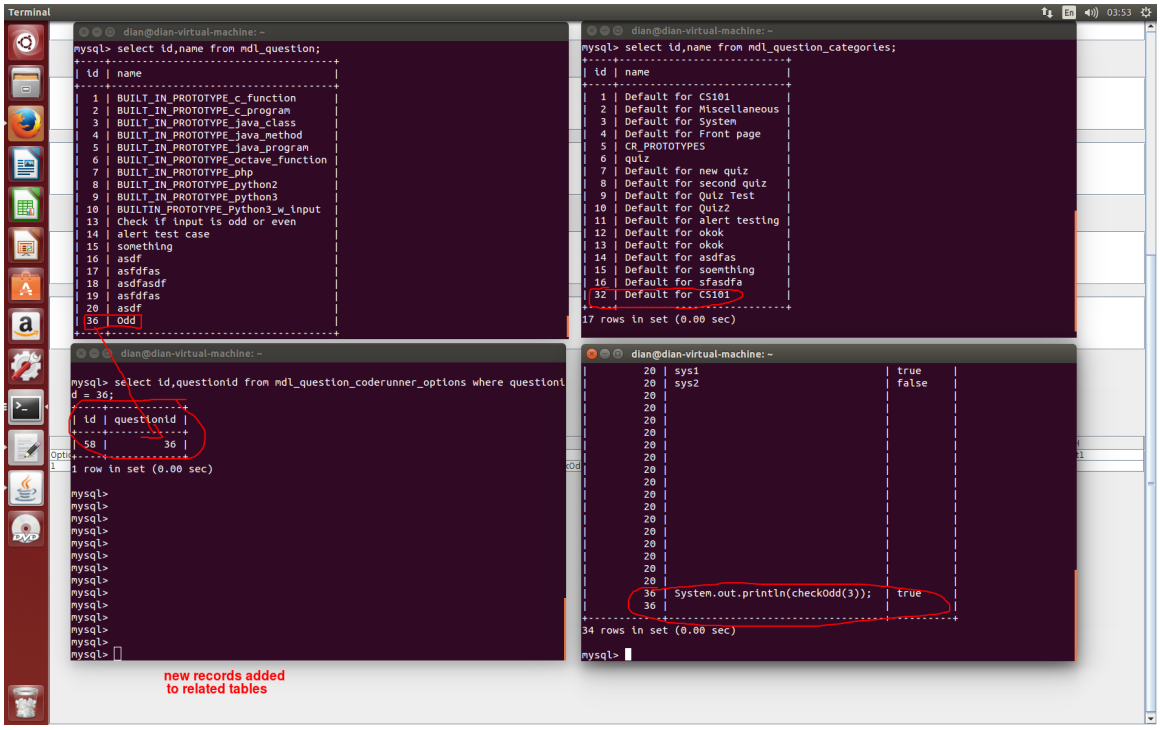

Figure 6.2: Insertion check

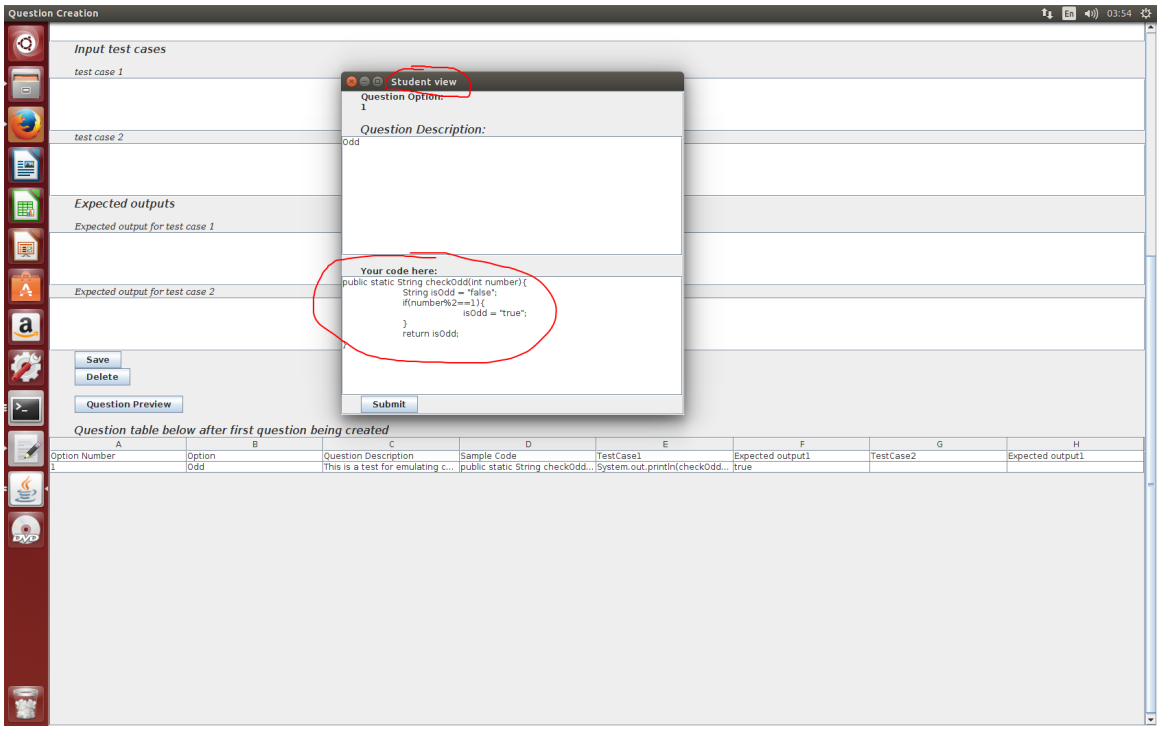

Figure 6.3: Testing code input

StudentAns.java

```
1 public class StudentAns{
2 public static String checkOdd(int number){
3         String isOdd = "false";
4         if(number%2==0){
5                 isOdd = "true";
6         }
7         return isOdd;
8 }
9
10 public static void main(String[] args){System.out.println(checkOdd(3));}}
```

Figure 6.4: Created Java program

Also, the prototype is able to execute answer from students and provide feedback to the terminal. As Code Runner is also able to remove unnecessary variants from the server, which takes us to the last testing step.

6.3 Deletion Testing

Refer to GUI mock-up, the local database table (JTable) gives all the variants that created, if the deletion has been successfully done, record in JTable and MySql Database

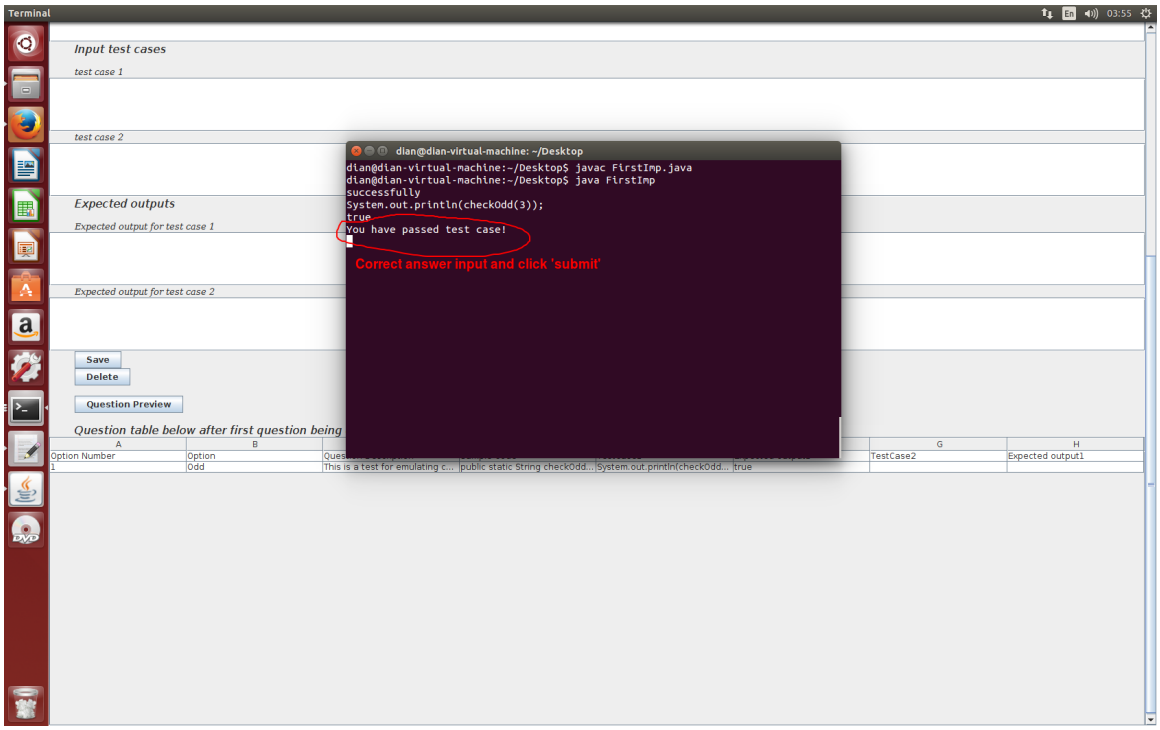

Figure 6.5: Feedback for correct answer

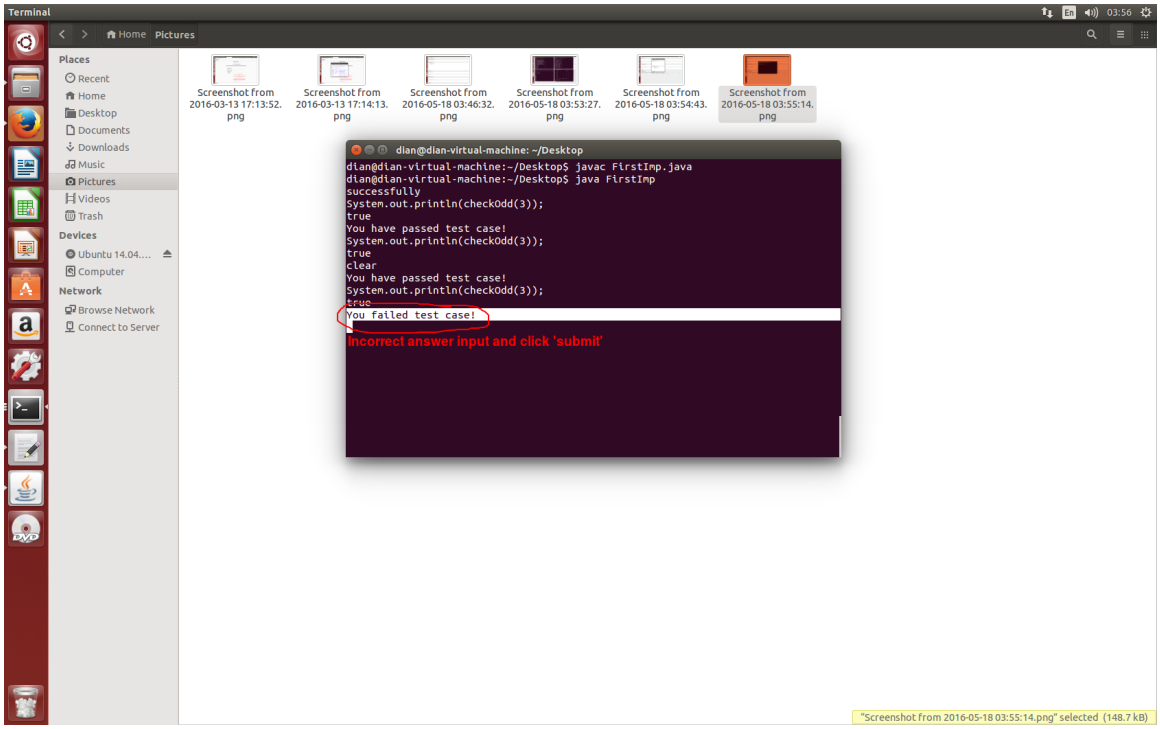

Figure 6.6: Feedback for incorrect answer

will be both removed after “delete” button being clicked.
From Insertion Testing part, it shows that *Odd* option has been inserted, thus in this part, *Odd* option becomes the deleting variant. After “delete” button has been clicked, 4 tables that show in Figure 5.2 is going to be checked again, the inserted variants disappeared which implies that deletion command in the prototype works in MySql Database server. Figure 6.7 presents the deletion testing result.

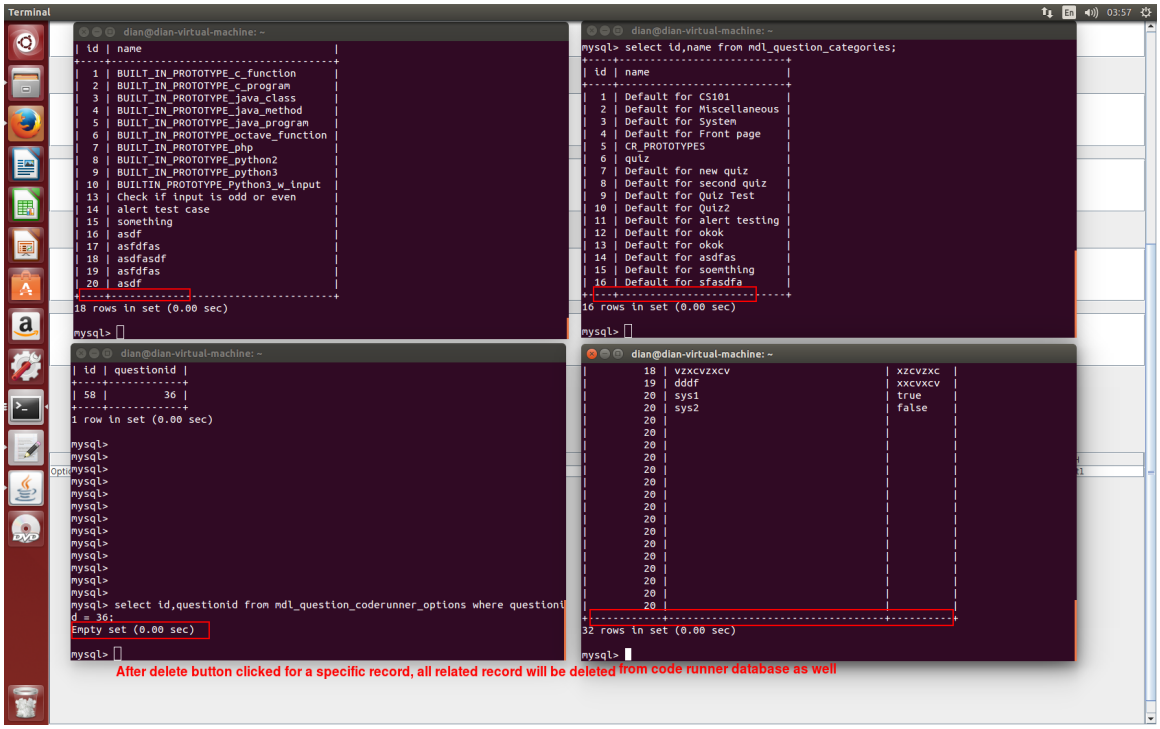

Figure 6.7: Deletion testing

Combining all testing results above, the prototype is able to provide proper interfaces and deal with question creation, deletion and answers comparison tasks as exactly what Code Runner does. Proof-of-concept has been successfully completed in Java. This process proves that the idea from the Brain Storm part is possible to implement in Code Runner site as Moodle is an open-source platform and MySql Database schema matches and allows incoming data from outside. The next step will have to bring the idea from prototype to Code Runner.

Chapter 7

Code Runner Structure Revision

7.1 PHP Files Review

In order to save more time for further implementation, it might be better to review Code Runner on the main structure of plug-in files and decide which part of PHP programming languages needs to be learned. Since PHP is a huge part programming language, it is highly unlikely to learn it systematically within the project period. Therefore, it is important to focus on main part that used by Code Runner and try to make the implementation successful.

As mentioned in previous parts of the report, the most significant point is to make sure the question creators are able to add more options under one question description. Hence, the focusing part is on files that relates to question creation, students answer running and the background database process. By reading through the Code Runner plug-in files, there are two files are coding about the above details, *questiontype.php* and *question.php* respectively.

On the one hand, *questiontype.php* is coded for question creation. It refers to some database tables such as *question_coderuner_options* and *coderunner_categories* which have been mentioned in Chapter 5. The main function of this file is to save question details including question description, testcases and programming language options, as records into database tables. It also implements the deletion of testcases or questions from both Code Runner HTML interface and database tables. It enables Code Runner to obtain the specified prototype from the database for current question context. Also, *questiontype.php* allows Code Runner to import or export question from Moodle XML format as Code Runner contains the list of test cases not answers, so Code Runner needs to override it.

On the other hand, *question.php* is mainly coded for Code Runner behaviors. It is able to get the question data, run the test cases based on students inputs, auto-mark

the grade and return feedback. It allows Code Runner to deal with those responses to ensure students obtain correct mark and useful feedback.

Therefore, in the anti-cheat system, it is important for not only implementing the new user interface but also have to make sure the new system maintains the rest of configurations and functionality. In other words, the new system should work normally as what Code Runner usually supposes to do.

7.2 Coding Structure

The main coding structure of Code Runner shows in *Code Runner Structure* in Appendices.

Where par1, par2 and par3 are parameters that passed into the functions, and each function will be invoked based on user behaviours in Code Runner interface, such as button click, or link review.

Chapter 8

Code Runner Implementation

After a review of the Code Runner structure and prototype have been both conducted, this chapter will briefly describe the implementation made in Code Runner for GUI interface and functionality.

8.1 User Interface Realization

Based on the prototype that has been built in previous chapters, the new anti-cheat system should have a same user interface as the prototype. Recall the original question creation page which shows in Figure 5.1 and Figure 5.2, it is necessary to keep the configuration of question description and test cases fields, but adding a new part into the page, which is called "Question Option". In this case, as declared previously, each question description contains several options, and each option is possible to have a number of test cases. Afterward, when students are trying their unique options in Code Runner, the Sandbox is able to retrieve the corresponding test cases and run at the background. Therefore, for each defined test case in question creation page, it is compulsory to specified which option it belongs to. Because question creator is allowed to create options as many as possible for each question, "Question Option" part enables user to add more blank choice by clicking "blanks for 3 more choices" button. The initial number of blanks for creator is 5 which is same as "Test Cases" part. The new implemented figures in question creation page are presented Figure 8.1 and Figure 8.2 respectively.

After the option part has been created, in the database of Code Runner, it is also necessary to insert the sample answer for the corresponding option, so that lecturers or students are able to review them if required after assignment has been due. Again, the "Sample Answer" part in question creation refers to both option created and test cases, which means each option is supposed to have its own sample answer. Therefore, the structure of "Sample Answer" is similar to "Question Option" part, and present in Figure 8.3.

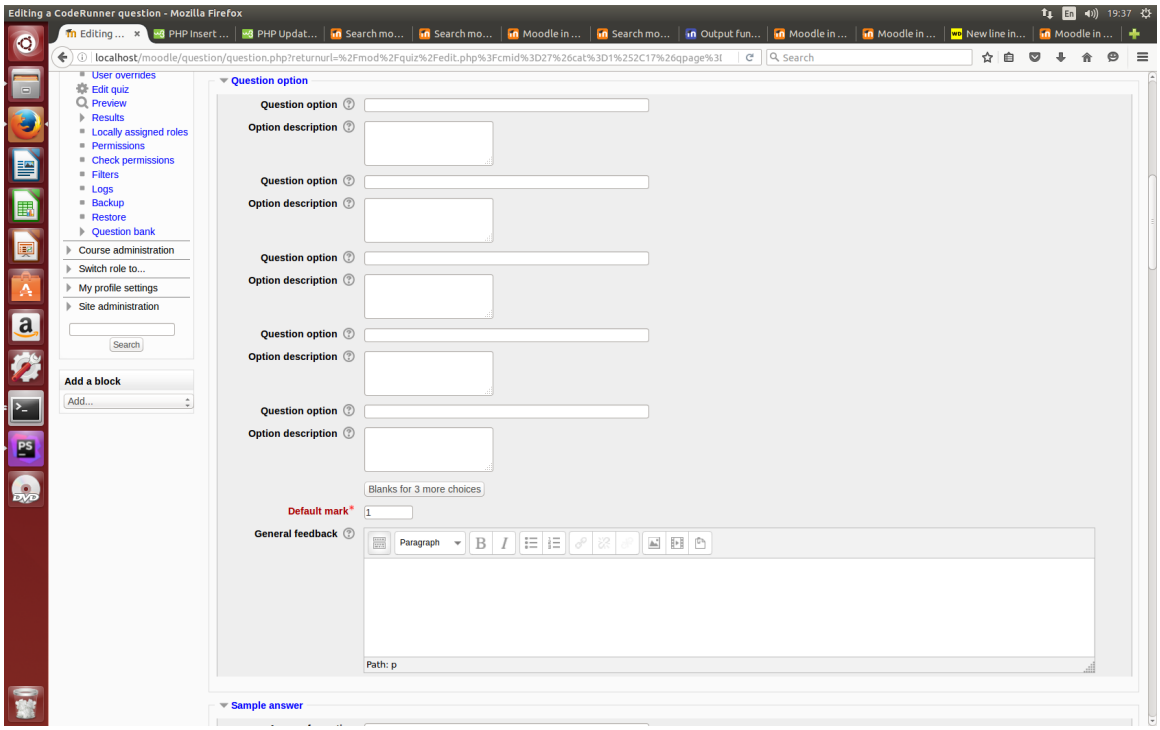

Figure 8.1: Question Option part in question creation

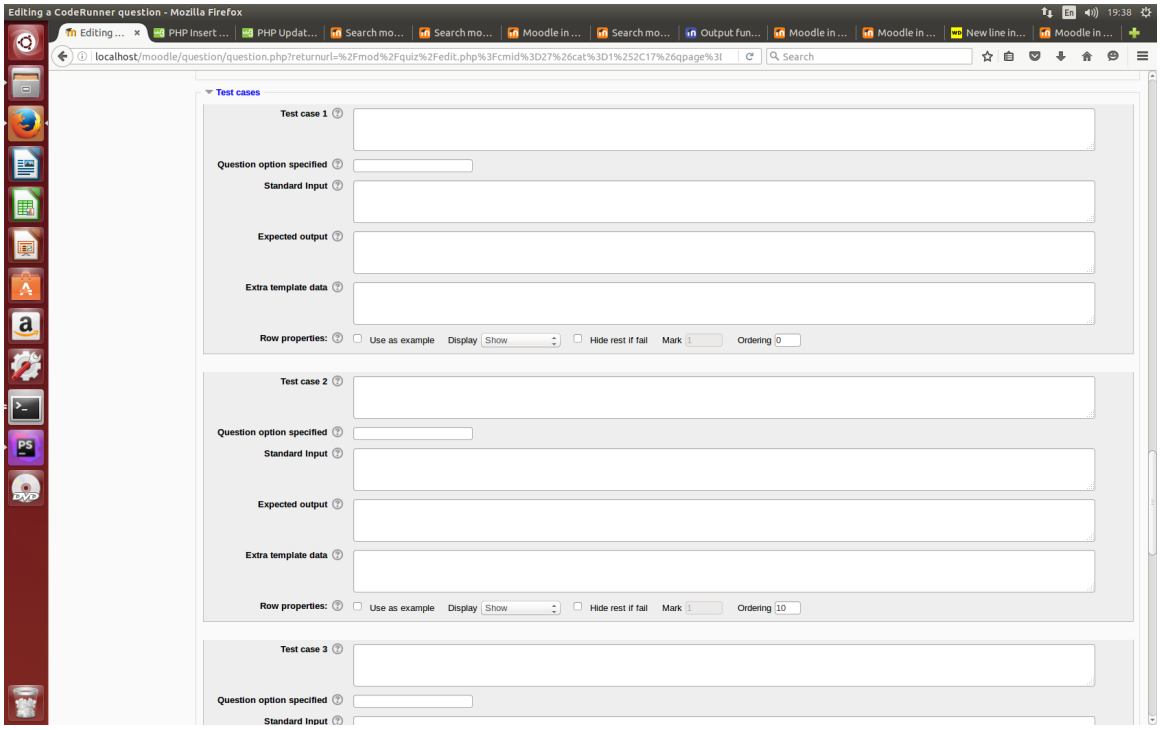

Figure 8.2: Modified Test Cases part in question creation

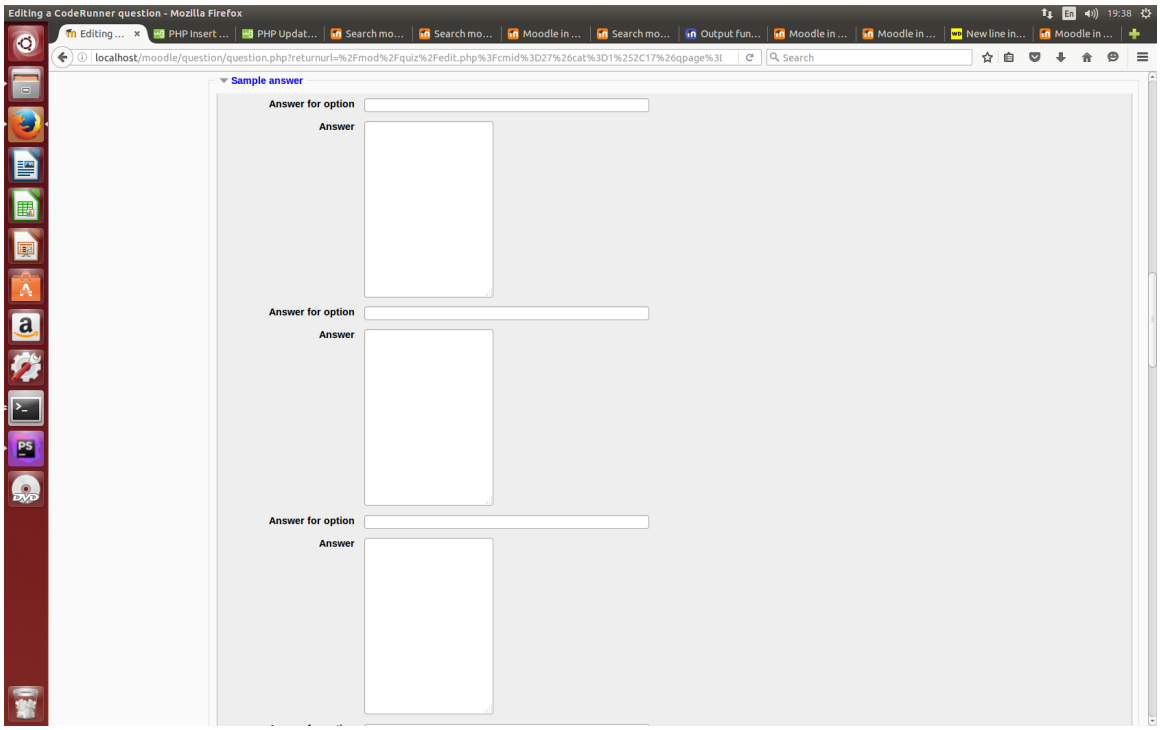

Figure 8.3: Sample Answer part in question creation

So far, a new user interface for question creation part has been built up. Instead of creating one question, the anti-cheat interface allows creator to create a number of options and test cases.

8.2 New Database Table Creation and Schema Definition

The new user interface is not operational, as there is no function defined for each part of creation. In order to make it work, the first step is to create a new database table with approach schema, then relate it to entire existed Moodle database.
As the figures show in the previous section, the new database needs to have columns of option name, option text, and corresponding test cases for each option. The relationship between newly created table and existed database can be done by linking *question ID* where *options* belongs to in foreign key constraint. Hence, we are able to embed a new table into the existed database. In other words, Code Runner could store new data that defined in question creation page, retrieve them and delete them when required, in SQL statement "Insert", "Select","Delete" respectively. The schema definition is presented in Figure 8.4, which is named *mdl_question_options* in Moodle database.

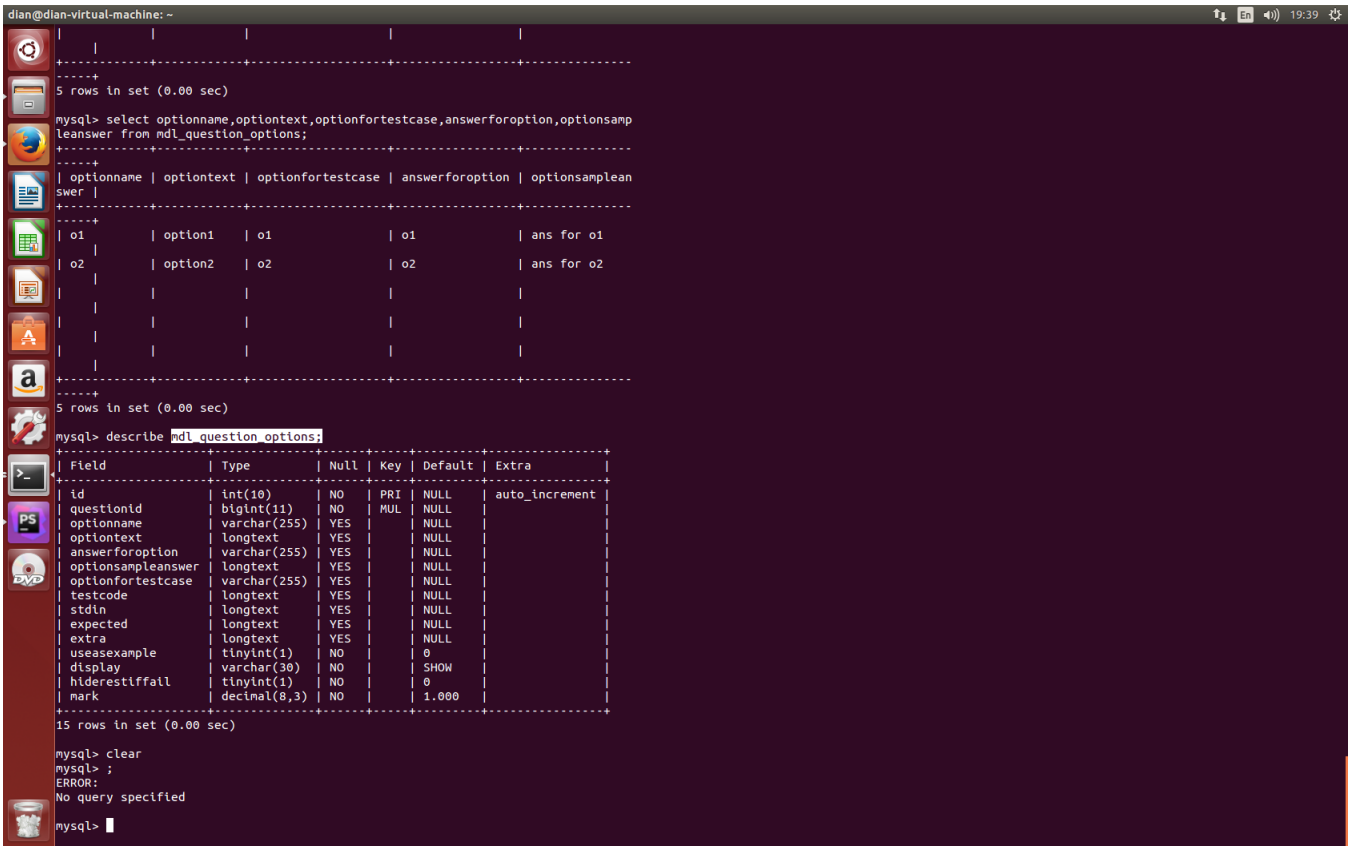

Figure 8.4: Option table schema definition

Where Primary key is an *auto_number* increment, and *questionid* becomes the foreign key that relate to *id* in *mdl_question table*. In this table, all features about test cases are re-defined, from *testcode* to *mark*. Because based on the newly created anti-cheat system, the test cases are not longer working for questions but options. Thus, all functions about test cases in Code Runner will have to redirect to newly created table, *mdl_question_options*, so that the exceptions from Sandbox won't get caught. Now, a new relationship in Code Runner has been built up and presented in Figure 8.5.

By now, a new database system has been defined, and the previous test cases table is ignored and won't be used in the system any longer. In order to prove the user interface and new database system is actually working, one simple test has been done. By defining two options under a question, new records were inserted into the *mdl_question_options* table, which is presented in Figure 8.6.

In Figure 8.6, both options, *Odd* and *Even*, are defined under *question ID* = 57, but for each of them, they are identical to both their own test cases and expected outputs.

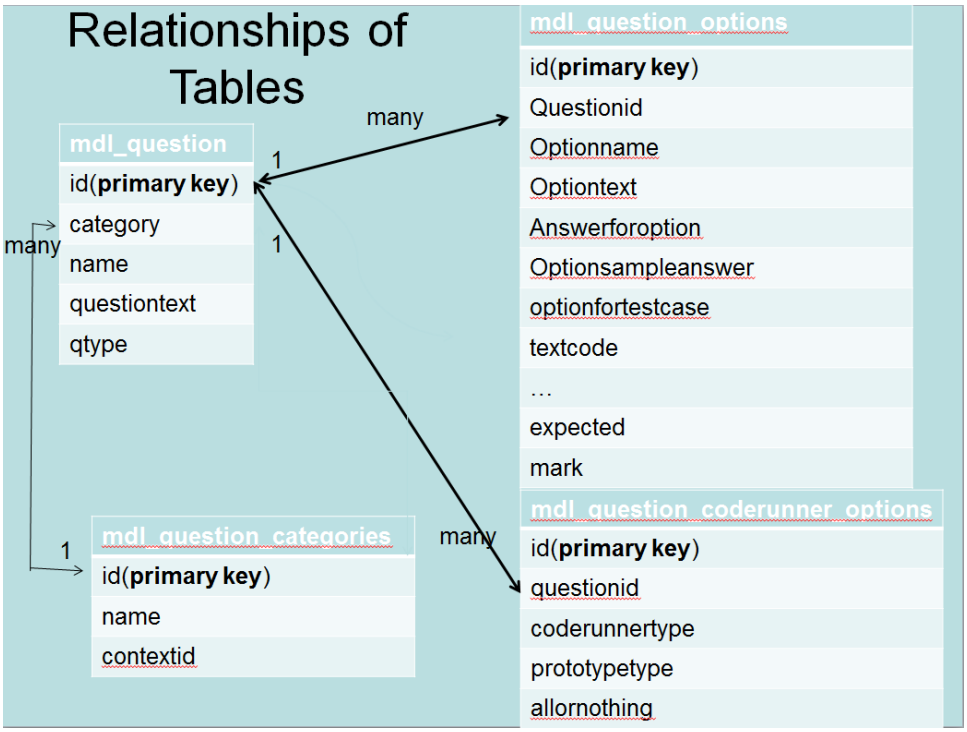

Figure 8.5: New tables relationship in Code Runner

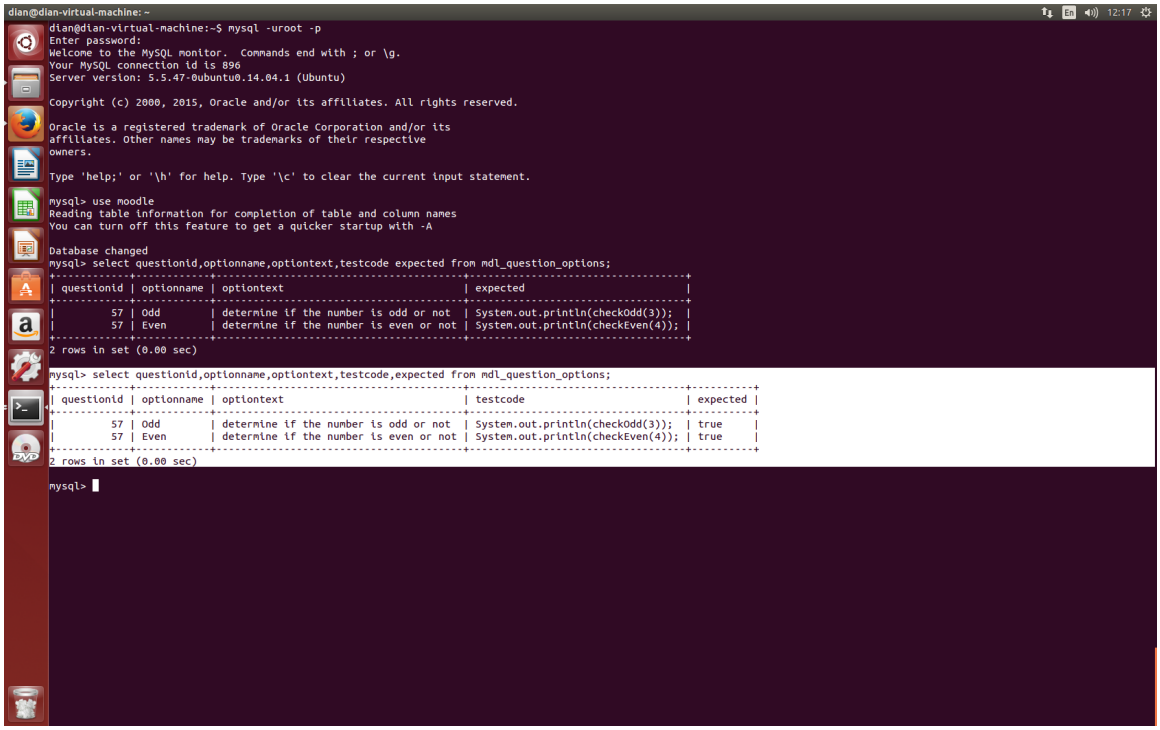

Figure 8.6: Data stored from new user interface

8.3 Functionality Implementation in Code Runner

After everything has been set up, we need to make sure the new anti-cheat system is possible to bring a new question view of options to students as well as running submissions based on corresponding test cases which are stored in *mdl_question_options table*.
Fist of all, in which way that the options assign to each student is significant. It could be done intentionally or make the assignment randomly. In this anti-cheat system, the random assignment becomes more preferred.
The intentional assignment increases manual work-loads upon lecturers or question creators, as for many courses that using Code Runner, the number of enrollments are vary from 200 to 1000. It is highly unlikely to define more than 200 options by creators to ensure every student has a unique question, if assignments are assigned by name, then students are also possible to find out who have the same question option based on their name(last name or first name). Then the system does not provide a feasible anti-cheat. Therefore, random assignment could avoid this situation since there is nothing to refer, to find out same question option holders. So, it makes more difficult to cheat on assignment in Code Runner, which is what we suppose to achieve at the end of the project.
In original Code Runner, it simply selects questions from the database in SQL statement. Since we are now focus on options not question, in order to achieve our goal, in it necessary to change the simply selection statement to random selection code. The modified SQL statement is presented in *Random Selection of Options* in Appendices.

Therefore, after the modification made in option selecting statement, it is able to pass the SQL statement result to student's view interface which inheritances the general interface structure but adding option description. Because of random assignment, after trying several times, it is possible for students to try both options in their attempts in terms of previous Boolean example. Preview version of both options,*Even* and *Odd*, are presented in Figure 8.7 and Figure 8.8 respectively.

As mentioned in last chapter, all functions about test cases in Code Runner have been redirected to newly created database table. Hence, for each particular option, such as *Odd* or *Even*, the students' submission will be run in the Sandbox upon corresponding test cases. Therefore, in Figure 8.7 and Figure 8.8, different input answers based on different options give the feedback about different test cases and individual marks.

8.4 Conclusion

The first idea, Personalized Assessment, has been successfully implemented in Code Runner. With new user interfaces on question creation page and student view, a new database table, schema and relationship have all been created and defined. The new

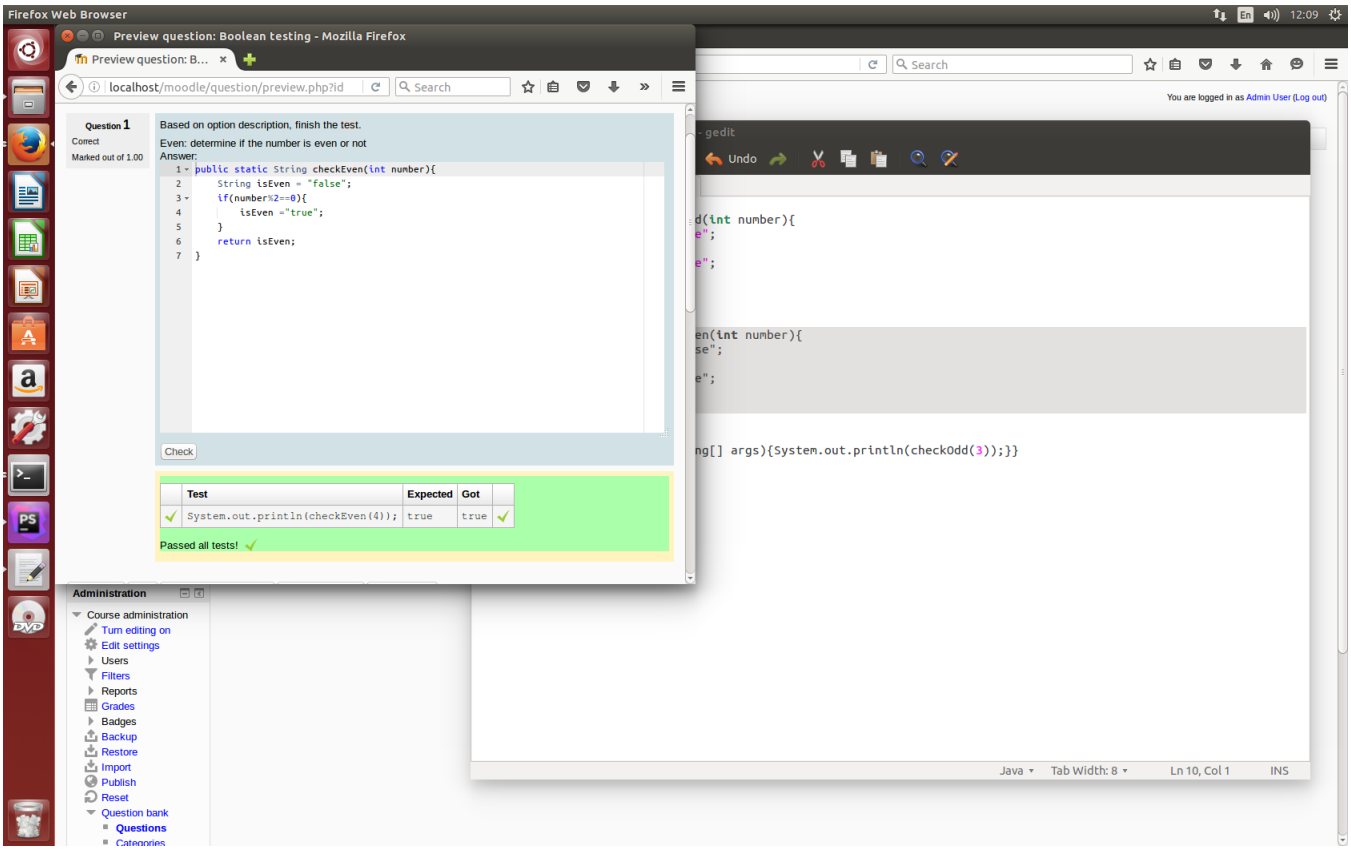

Figure 8.7: Preview of Even option and test

functionality of question creation and students' submission handling are both implemented. The new anti-cheat system maintains the original Sandbox functionality, but makes it more difficult to cheat in Code Runner. Therefore, this anti-cheat system achieves the project goal: push up barriers against cheating in Code Runner.

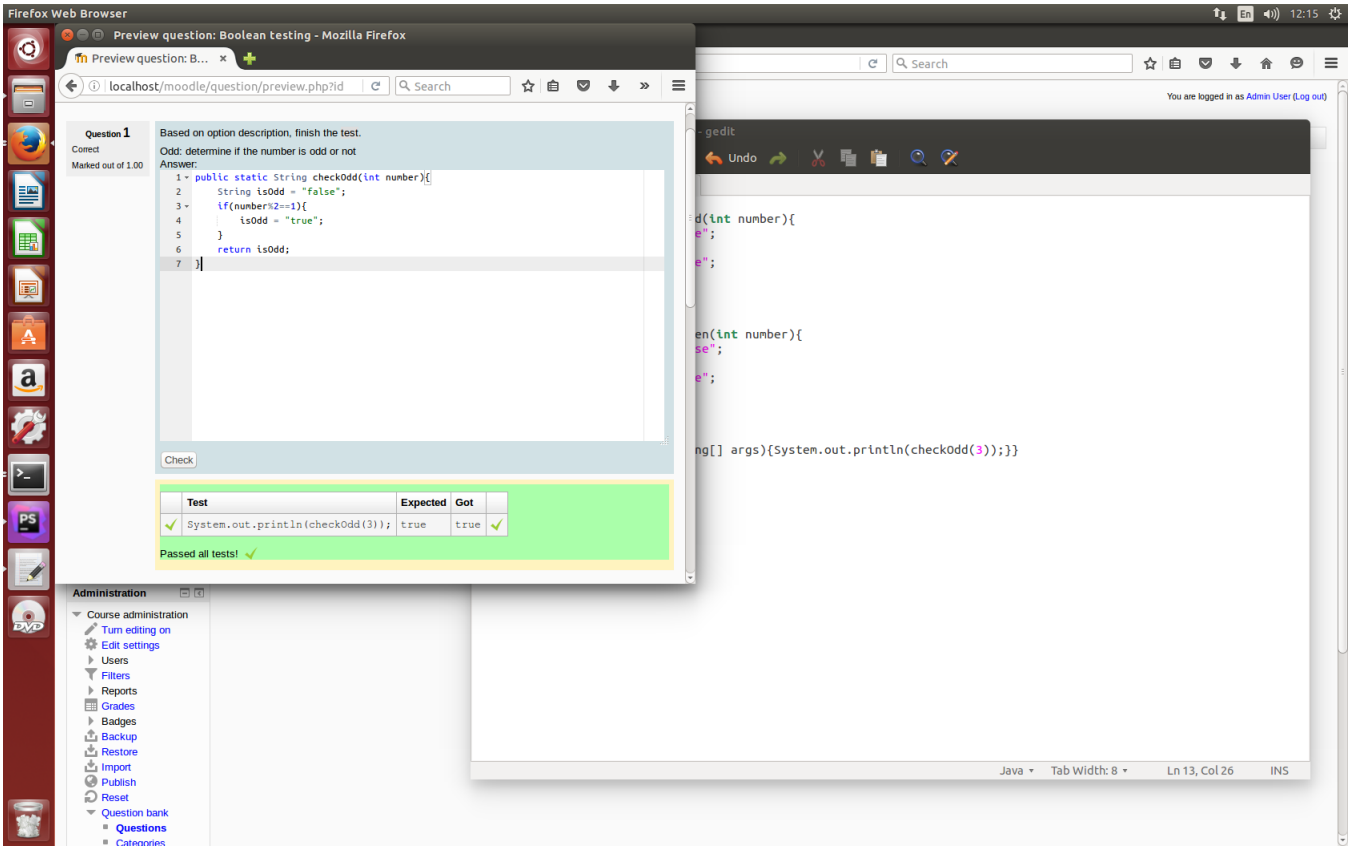

Figure 8.8: Preview of Odd option and test

Chapter 9

Similarity Check With Comments Implementation

In earlier study we dismissed similarity checking as a feasible idea against plagiarism. However, it could be still an useful for analysis purposes if comments are added to distinct submissions.

9.1 Preparation

Since similarity checking in Code Runner is using the Brute Force approach by comparing similarity of each submission to the rest of submitted answers, the computationally expensive will be occurred. Before we implement it straightforward in Code Runner, it is necessary to check whether the time penalty is acceptable in such algorithm.

By using a for-loop in SQL statement, there are 10, 100 and 1000 records were successfully inserted into Moodle Database. Creating a new .php file to test the time performance of similarity checking in Brute Force. The output of the test shows that the time penalty increases from 0.7 seconds when sample has the size of 10 records to around 4.0 seconds when same has the size of 1000 records. The reason why test sample is only up to 1000 records is because of the number of enrollments. It is highly unlikely to believe that the number of enrollments will be much more than 1000. In order to make the implementation realistic, 1000 could be treated as the highest sample size. In terms of testing result, the maximum time for Brute Force is around 4 seconds which is acceptable by students. The content of *testing .php* file is presented in *Time Penalty Test* in Appendices.
The content of testing file is based on the local machine which uses for Code Runner testing.
Where *similar_text* display above is an existed method in PHP library, and it is the

main method that used for Brute Force similarity check. The idea behind the algorithm is *divided and conquer*. It works by finding the longest common string between two inputs and breaking the long text into subsets around the string. Then process the same idea in each subset by making recursive calls until it reaches character and doing comparison. After all, it conquers each subset result and returns the number of matching characters in percentage [7].

9.2 Implementation of Similarity Checking in Code Runner

The similarity checking should be invoked after each submission handed in. Once current Code Runner assignment becomes inactive, an overall similarity table should be generated as a similarity summary for teacher review.
Based on features of Code Runner, after each attempt has been finished, Code Runner will automatically bring students who just attempted the question back to "View" page to see the total marks of the current question. In order to make each student and teacher know about the similarity result, new implementation should have to make "View" page be able to pop up a dialog to display the similarity percentage for current attempt question. Therefore, this implementation should focus on *view.php* file, and make code modification so that the above achievement will be realized. Because the pop-up dialog should be displayed before Code Runner brings students to the mark table, so the implementation should be done before *view.php* being invoked. Furthermore, checking upon each submission should be able to track name, Code Runner ID as well as question ID where the highest similarity occurs. The code modification shown in *Similarity Checking* in Appendices presents the similarity checking implementation and dialog invocation. The implementation is mainly refer to the structure shows in Figure 9.1 [8].

Where *responsesummary* in database table gives students submissions.

Since during the assignment is active, Code Runner will only give the feedback about the newest submission that one student just finished at each time. Hence, only one record would be selected for submission comparison, and only the highest similarity percentage will be recorded. In order to track the student details who has the highest similarity percentage to current submission, table *question_attempts*, table *question_attempt_steps* and table *user* has been inner joined in terms of current question ID. The output of the result is done in javascript, the similarity percentage has been rounded up in 2 decimal places. The output for one submission checking is presented in Figure 9.2.

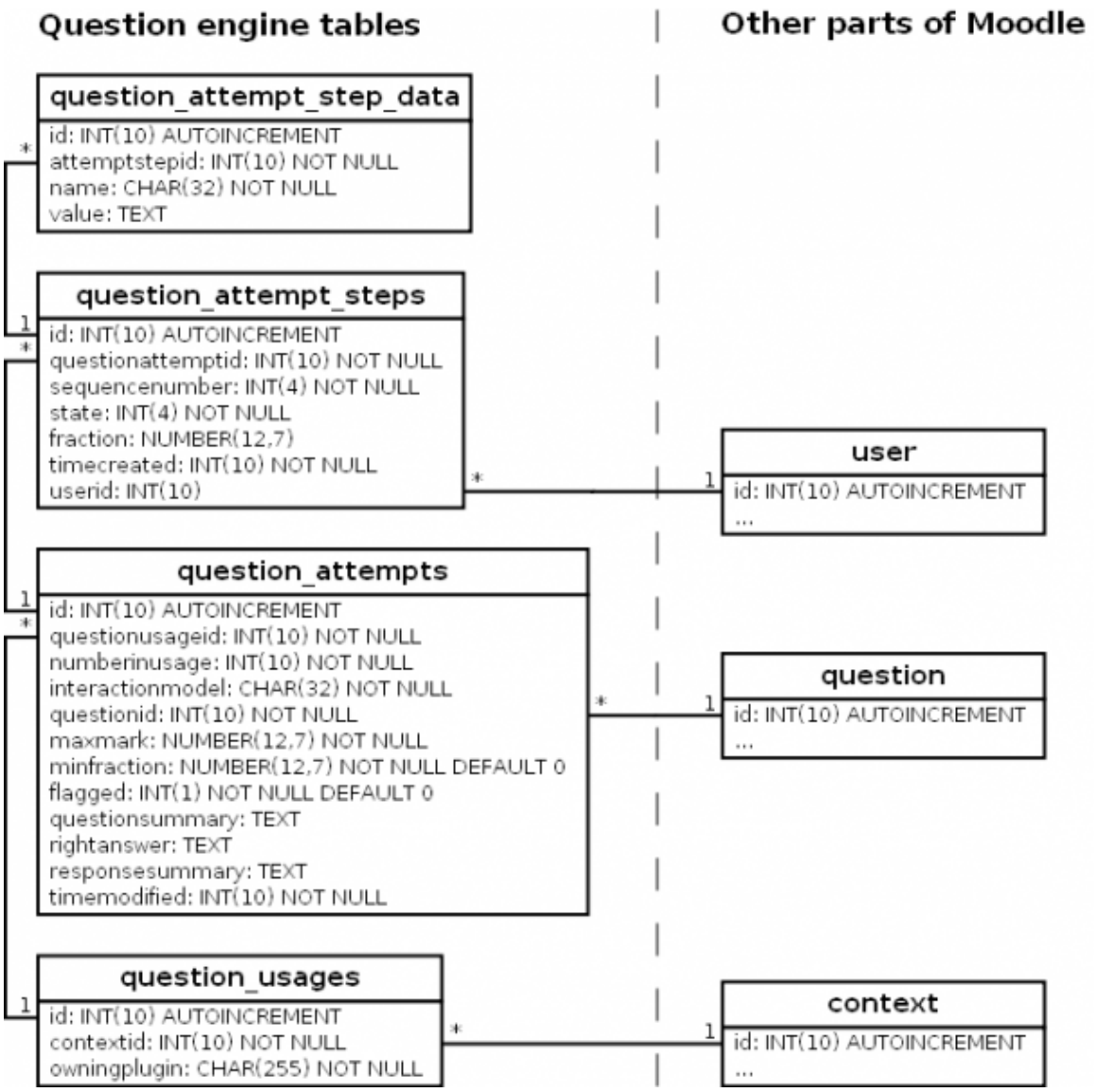

Figure 9.1: Parts of relationships of Moodle database tables

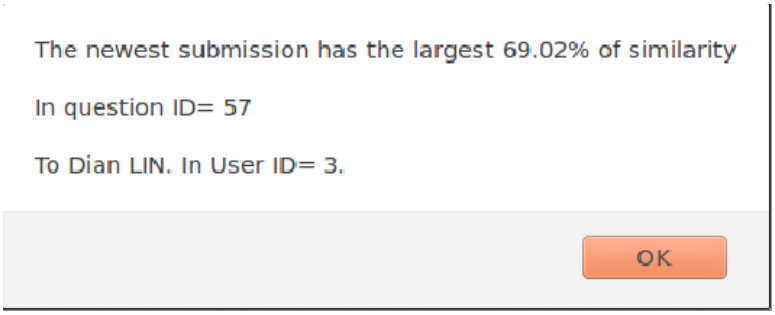

Figure 9.2: Similarity pop-up dialog for one submission

As declared in previous sections, after current assignment being no longer active, a similarity summary table will be generated based on each submission in the database. The idea in this part is same as previous similarity checking above, only the highest similarity percentage will be tracked and displayed. The code implementation is presented in *Summary Similarity Table* in Appendices.
Nested loop has been used in above implementation. Same records are retrieved from a same database for both variables, *recordsToBruteForce1* and *recordsToBruteForce2*. In similarity checking, each object carried by each variable has been run through except itself. As we only store the highest similarity percentage, if one object compares to itself then every submission will definitely have 100% similarity. This implementation guarantees that the similarity of each submission is compared with the rest of submissions.
Where similarity of each submission will be located to corresponding attempt in an array each time. After all, key and value in the array will be retrieved one by one and make an output. The output will present at the top of current course page, shown in Figure 9.3.

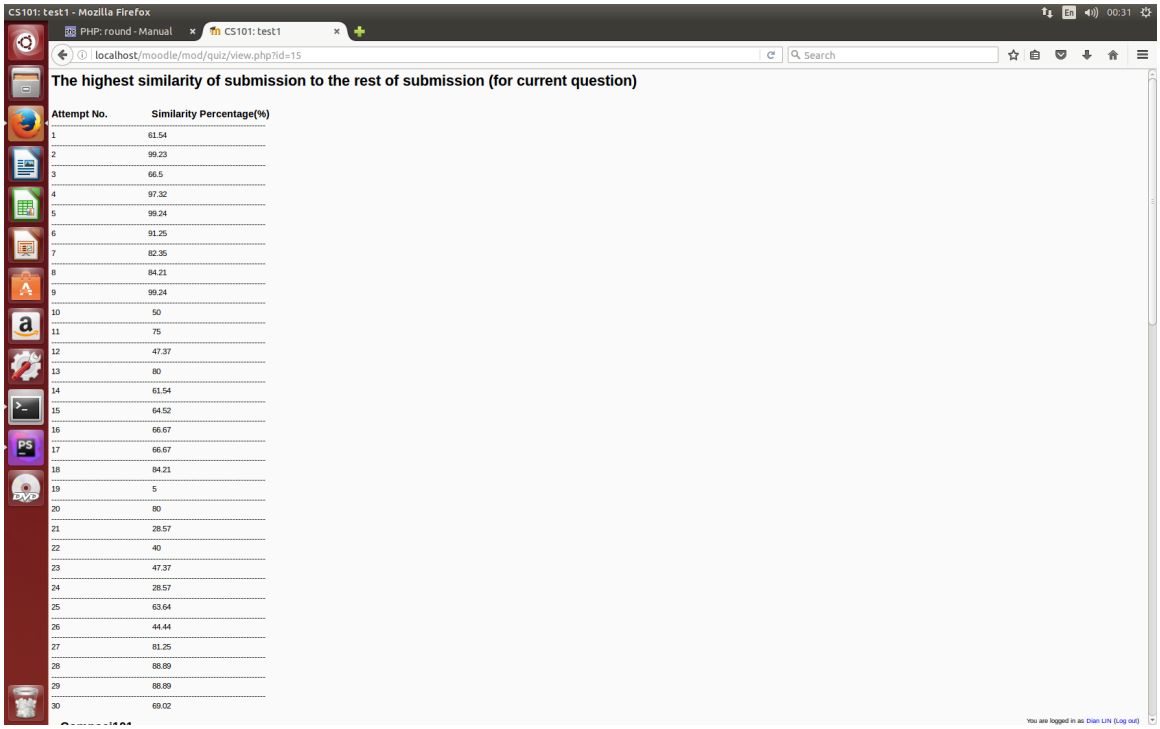

Figure 9.3: Similarity pop-up dialog for one submission

9.3 Analysis

So far, similarity checking with comments has been successfully implemented into Code Runner. Refer to my testing result, although Brute Force has been used when

we check the similarity between new submission and the rest of submitted answers, the time performance is acceptable. In other words, Code Runner is able to give feedback to student under an acceptable duration of run time, less than 2 seconds in the test when the sample size is 100 in Code Runner. Recall that there is around 85% of questions in Code Runner have been reused, which implies that students could copy and paste the corrected submissions from someone else who has passed the Code Runner test in previous years. With comments required, this issue could be highly reduced in terms of different comments added by different students. Therefore, the implementation in this part helps the University of Auckland to avoid one of the significant downsides that existed in Code Runner.
After comments are required for each student to answer the Code Runner question, the similarity between submissions are reduced quite a lot, from 100% to around 69%. Notice that this is for the program with code length of only 7 lines. Based on the Similarity Checking Research(without comments) result in Chapter 3, the median length of coding part in Code Runner is around 30 lines, it is likely to see that similarity between submissions will be much lower if students are required to add comments for their programmings.

9.4 Conclusion

This is a feasible idea against assignment cheating in Code Runner. Time penalty and performance check are both acceptable. Since comments are mainly based on personal thoughts, comments-requirement will efficiently separate short programs. Hence, similarity checking with comments is highly recommended to be implemented in Code Runner to build up a better anti-cheating system.

Chapter 10

Limitation and Summary

10.1 Objective Evaluation

We have built a new anti-cheat system that maintains the most functionality of Code Runner system, it allows adding more than one option in question creation and required comments for students' answers to make similarity checking possible. As we known, there is also a similar system called 'Question Bank' exists in Code Runner, the main difference between new anti-cheat system and Question Bank is that Question Bank asks question creator to create different questions. Hence, it cannot guarantee that the difficulty of each created question are even. However, in new anti-cheat system, as each option bases on each question description, the system is able to give similar assignments to student which avoids unfair assignments.

Nevertheless, any anti-cheat system may not be able to guarantee that there is not possible to cheat in Code Runner. So, students could still cheat by asking someone else to do the assignment for them. Also, for Stage I courses, there are usually more than 500 students for each Computer Science course, it is highly unlikely to expect that lecturers or question creators will create more than 500 options for each question, which implies that cheating is still possible by finding same option offered and copy-pasting solutions.

In case of similarity checking in comments-requirement, most programming languages will ignore the comment contents, and Code Runner does the same thing. Even though cheat is able to be detected, it simply makes the comparison between raw texts instead of considering the content of comments. Therefore, the possibility of random comments added would be occurred.

10.2 Conclusion

Code Runner is a free and open-source Moodle question-type plug-in that lets teachers set questions where the answer is program code. Students are able to develop and test their code using a normal development environment and submit through web browsers [9]. However, being not able to detect cheating is the most significant weakness of Code Runner, and might lead to negative effects among students.

This study bases on the vulnerability of Code Runner to discuss ideas that could be possible to prevent Code Runner from cheating. Research about similarity checking, programming variation and questions recycling have all been done. The research results show that due to short programming length, a high probability of similar submissions occurs frequently in Code Runner. It indicates that the idea about checking similarity without comments is not an effective way to prevent cheating. Furthermore, from the result of Programming Variation, students can possibly search solutions online or do programming languages translations to achieve higher marks. So it is unfair to hard-working students and might cause a loss of faith in Code Runner. Last but not least, questions recycling is widely detected in Code Runner, which provides a easy way for students to get an easy pass on the course. However, this plagiarism can be solved by restricting students access after they have completed the course.

More variants can be defined in one question became one feasible idea to be implemented. In order to prove that new functionality is possible to be added in Code Runner and matches the corresponding database schema, proof-of-concept process has been successfully conducted by building up a prototype in Java program.

In Code Runner implementation, two ideas have been realized by making code modification. On one hand, bring the idea of personalized assessment from proof-of-concept to Code Runner, which guarantees that options are randomly assigned to random students. The option for every student may not be unique due to the large number of enrollment and less options created. Nevertheless, this implementation actually makes cheating much more difficult in Code Runner as it is hard to find out for students which options are assigned. On the other hand, once Code Runner requires students to add comments, it not only helps students to have a better understanding on learning outcomes, but also reduce the similarity of short coding so that similarity checking becomes possible and feasible. Both ideas are reliable to build up a better anti-cheat system.

10.3 Future Work

Due to Code Runner is well conducted on auto-marking part while not good at question(option) auto-generation, the effectiveness is reduced a lot. As instructors have

to define as many options as possible to make sure each student will obtain a unique assessment, manual works are costly. In the future work, the ideas from Problets [10] will need to be introduced into Code Runner. Since it is really good at question auto-generation part by creating one general case, make the replacement of key variables based on each student ID, it can be used for a long time [11].

Nonsense comments provided by students is another limitation for idea *Similarity Checking with Comments* in Code Runner. It would be necessary to build up a comment-network by importing training set of comments about different topics, it actually relates to artificial intelligent. Also, new software tools will have to be introduced into Code Runner in order to retrieve students comment inputs and make further analysis.

Although there are probably still other limitations that we haven't explored yet, the new system has improved the weakness of anti-cheat quite a lot in Code Runner. More research and ideas would have to be conducted in order to against plagiarism in Code Runner.

Bibliography

[1] J.Cole, H.Foster, *Using Moodle:Teaching with the popular open source course management system*, 0'Reilly Media, Inc, 2007.

[2] M.Org, Pedagogy(2013). Retrieved from
URL http://docs.moodle.org/23/en/Pedagogy.

[3] R.Lobb,CODE RUNNER(2013). Retrieved From
URL https://github.com/trampgeek/CodeRunner.

[4] CopyScape. Online diff checker.
URL http://www.copyscape.com/compare.php.

[5] SortSearch. Retrieved from
URL http://interactivepython.org/runestone/static/pythonds/

[6] Wikipedia. Retrieved from
URL https://en.wikipedia.org/wiki/MySQL.

[7] StackOverflow. Retrieved from
http://stackoverflow.com/questions/14136349/how-does-similar-text-work.

[8] Overview of the Moodle question engine. Retrieved from
https://docs.moodle.org/dev/Overview_of_the_Moodle_question_engine#Database_tables

[9] R.Lobb, J.Harlow. *Code Runner: A Tool for Assessing Computer Programming Skills*. The University of Cantebury.

[10] Amruth N.Kumar. *Automated Generation of Self-Explanation Questions in Worked Examples in a Model-Based Tutor*. Ramapo College of New Jersey, Mahwah, UAS.

[11] S.Manoharan. *Personalized Assessment as a Means to Mitigate Plagiarism*. The University of Auckland.

Appendices

.1 *Code Runner Structure*

```
<?php
        public function functionName() (par1, par2, par3 ){

                function_implementation goes here;

                return xxx;
                        .
                        .
                        .
        public function functionName() (par1, par2, par3 ){

                function_implementation goes here;

                return xxx;
}
```

.2 *Random Selection of Options*

```
public function load_for_cache($questionid) {
        global $DB;
        $questiondata = $DB->get_record_sql('
            SELECT qo.optionname,qo.optiontext,q.*, qc.contextid
            FROM {question} q
            JOIN {question_categories} qc ON q.category = qc.id
            JOIN {question_options} qo ON qo.questionid = q.id
            WHERE q.id = :id
            ORDER BY RAND()
            LIMIT 1 ', array('id' => $questionid), MUST_EXIST);
        get_question_options($questiondata);
        return $questiondata;
    }
```

.3 *Time Penalty Test*

```
 <?php
$time_pre = microtime(true);
$servername = "localhost";
$username = "root";
$password = "linqi520";

$inputString = "public static String checkEven(int number)
```

```
        {
                String isEven = 'false'; if(number%2==1){isEven = 'true';
        }
                return isEven;
                }";

// Create connection
$conn = mysql_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully!!\r\n";

$usemoodle = 'use moodle';
$exec = mysql_query($usemoodle,$conn);

$sql = 'SELECT testing FROM forloop';
$retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not get data: ' . mysql_error());
   }

   while($row = mysql_fetch_assoc($retval)) {
      echo "testing Code :{$row['testing']}  \r\n ".
      similar_text($row['testing'], $inputString, $percent);
      echo "Similarity percentage: ". $percent . "\r\n";
        "--------------------------------------\r\n";
   }
mysql_close($conn);
$time_post = microtime(true);
$exec_time = $time_post - $time_pre;
echo "Total time usage:" . $exec_time;
?>
```

.4 *Similarity Checking*

```
$sqlGetnewestRecord = 'SELECT responsesummary  ' .
    'FROM {question_attempts}  ' .
    'WHERE responsesummary != "" ' .
    'ORDER BY id DESC  ' .
```

```
    'LIMIT 1';
$newRecord = $DB->get_records_sql($sqlGetnewestRecord);
foreach ($newRecord as $newestone){
    $string1 = $newestone->responsesummary;
}

$sqlStatement = 'SELECT responsesummary,questionid  ' .
    'FROM {question_attempts}  ' .
    'WHERE responsesummary != ""
    AND id < (SELECT MAX(id) FROM {question_attempts})';
$records = $DB->get_records_sql($sqlStatement);
$highest = 0.0;

foreach($records as $record){
    $string2 = $record -> responsesummary;
    similar_text($string1,$string2,$percent);
    if ($percent > $highest){
        $highest = $percent;
        $questionID = $record -> questionid;
        $sqlStatement2 = 'SELECT firstname, lastname, userid
' .
            'FROM {question_attempts} , {question_attempt_steps}, {user}
' .
            "Where responsesummary = '$string2'
            AND questionid = '$questionID'
            AND {question_attempts}.id =
            {question_attempt_steps}.questionattemptid
            AND {question_attempt_steps}.userid = {user}.id";
        $userdetails = $DB ->get_records_sql($sqlStatement2);
        foreach ($userdetails as $userdetail)
            $userID = $userdetail -> userid;
            $firstName = $userdetail -> firstname;
            $lastName = $userdetail -> lastname;
    }
}
echo '<script language="javascript">';
echo 'alert("The newest submission has the largest '. round($highest,2) .
'% of similarity\r\n \r \nIn question ID= ' .
$questionID. '\r\n \r\nTo '. $firstName.' '.$lastName.'.
In User ID= '.$userID .'.")';
echo '</script>';
```

.5 *Summary Similarity Table* in Appendices

```
$sqlStatement3 = 'SELECT responsesummary  ' .
    'FROM {question_attempts}  ' .
    "WHERE responsesummary != '' AND questionid = '$questionID'";

$recordsToBruteForce1 = $DB ->get_records_sql($sqlStatement3);

$recordsToBruteForce2 = $DB -> get_records_sql($sqlStatement3);

$similarityPercentageArray = array();

foreach($recordsToBruteForce1 as $recordToBruteForce1){
    $string3 = $recordToBruteForce1->responsesummary;
    $highSimilarity = 0.0;
    foreach($recordsToBruteForce2 as $recordToBruteForce2){
        if($recordToBruteForce1 != $recordToBruteForce2){
            $string4 = $recordToBruteForce2 -> responsesummary;
            similar_text($string3,$string4,$outputPercent);
            if($outputPercent>$highSimilarity) {
                $highSimilarity = $outputPercent;
            }
        }
    }
    array_push($similarityPercentageArray, round($highSimilarity,2));
}

echo "<strong><font size='5'>The highest similarity of submission to the
rest of submission (for current question)<br ><br ></font></strong>";

echo "<strong><font size='3'>Attempt No. </font></strong>".
str_repeat(' ', 20)."<strong><font size='3'>
Similarity Percentage(%)<br ></font></strong>";

foreach($similarityPercentageArray as $key=>$value){
    $attemptNumber= $key+1;
    echo "----------------------------------------<br >";
    echo "$attemptNumber" .str_repeat(' ', 45).  "$value<br >";
}
```