

**IMPROVING LEARNING AND TEACHING THROUGH
AUTOMATED SHORT ANSWER MARKING**

A thesis submitted to The University of Manchester for the degree of Doctor of
Philosophy in the Faculty of Engineering and Physical Sciences

2010

RAHEEL SIDDIQI

SCHOOL OF COMPUTER SCIENCE

Table of Contents

ABSTRACT	9
DECLARATION.....	10
COPYRIGHT STATEMENT	11
ACKNOWLEDGEMENTS.....	12
PUBLICATION AND CONFERENCE PAPERS BASED ON THE RESEARCH PRESENTED IN THIS THESIS	13
1. INTRODUCTION	14
1.1 Short-Answer Questions.....	14
1.2 Role of Students' Information in Teacher's Decision Making	15
1.3 Key Features and Purposes behind Assessment Tasks (in general).....	18
1.4 Classroom Assessment and its Advantages	19
1.4.1 Importance of Classroom Assessment for Effective Teaching	20
1.4.2 Importance of Classroom Assessment in Improving Student Learning	20
1.5 Utilizing IndusMarker to Improve Teaching and Learning	21
2. SIMILAR SYSTEMS.....	23
2.1 Short-Answer Marking Systems.....	24
2.2 C-rater	25
2.2.1 C-rater's Approach to Marking Students' Responses	26
2.2.2 C-rater's Evaluation	30
2.3 The Oxford-UCLES System	31
2.3.1 The Oxford-UCLES System's Approach to Marking Students' Responses.....	32
2.3.2 The Oxford-UCLES System's Evaluation	34

2.4 Automark	35
2.4.1 Automark’s Approach to Marking Students’ Responses.....	35
2.4.2 Automark’s Evaluation	38
2.4.2.1 The Blind Experiment.....	39
2.4.2.2 The Moderation Experiment.....	40
2.5 Related Techniques.....	42
2.5.1 Neural Network	42
2.5.2 Text Mining.....	43
2.6 Summary of the Similar Work.....	44
3. “ANSWER STRUCTURE” SPECIFICATION LANGUAGES AND INDUSMARKER’S ARCHITECTURE.....	45
3.1 The Question Answer Language (QAL)	45
3.2 The Question Answer Markup Language (QAML).....	52
3.3 IndusMarker’s Architecture.....	60
3.3.1 The Answer Text Analyzer’s Architecture	61
3.3.2 The QAML Structure Editor’s Architecture	62
4. DESIGN AND DEVELOPMENT OF INDUSMARKER’S COMPONENTS.....	65
4.1 Design and Development of the Answer Text Analyzer’s Sub-Components	65
4.1.1 The Spell Checker	65
4.1.2 The Linguistic Features Analyzer	66
4.1.3 Overall Algorithm for the Structure Matcher and the Marks Calculator	67
4.1.4 Structure Matching Algorithms for Various QAL Constructs	69
4.1.4.1 “Text String” Matching Algorithm.....	71
4.1.4.2 “Noun Phrase Required” Matching Algorithm.....	71
4.1.4.3 “Verb Group Required” Matching Algorithm	72
4.1.4.4 “No Noun Phrase / No Verb Group / No Word” Condition Matching Algorithm	73
4.1.4.5 “Alternative Options” Matching Algorithm	75
4.1.4.6 “Allowed Permutation” Matching Algorithm	77
4.1.4.7 “Not” Condition Matching Algorithm	78
4.1.4.8 “Sub-pattern” Matching Algorithm.....	79
4.2 Design and Development of the QAML Structure Editor’s Sub-Components	80

4.2.1 The Validator	81
4.2.2 The Automatic QAML Generator	84
4.2.3 The Transformer	88
5. INDUSMARKER'S EVALUATION	90
5.1 Evaluation Method and Criteria.....	90
5.2 Knowledge Targets and the Design of Short-Answer Questions	91
5.3 IndusMarker's Evaluation at Bahria University	93
5.3.1 Allowed Short-Answer Question Types for OOP Tests	94
5.3.2 The OOP Tests Creation and the Required Structures Formulation and Validation Phase	99
5.3.2.1 First OOP Test	100
5.3.2.2 Second OOP Test.....	101
5.3.2.3 Third OOP Test	104
5.3.2.4 Fourth OOP Test.....	107
5.3.2.5 Fifth OOP Test	109
5.3.2.6 Sixth OOP Test.....	111
5.3.2.7 Analysis of Evaluation Results.....	113
5.3.3 Using IndusMarker to Conduct the OOP Practice Tests.....	117
5.4 IndusMarker's Evaluation at the City School.....	119
5.4.1 Allowed Short-Answer Question Types for Science Tests.....	120
5.4.2 The Science Tests Creation and the Required Structures Formulation and Validation Phase	122
5.4.2.1 First Science Test.....	125
5.4.2.2 Second Science Test.....	127
5.4.2.3 Third Science Test	128
5.4.2.4 Analysis of Structure Testing and other Evaluation Results	130
5.4.3 Using IndusMarker to Conduct the Science Practice Tests.....	135
5.5 Comparison and Lessons Learnt from the Two Case Studies.....	135
6. DISCUSSION AND CONCLUSION	138
6.1 Impact of IndusMarker on Students' Performance.....	138
6.2 IndusMarker's Performance without "Linguistic Features Analyzer"	141

6.3 Comparison of IndusMarker with Other Similar Systems	144
6.3.1 Comparison of IndusMarker with C-rater	144
6.3.2 Comparison of IndusMarker with the Oxford-UCLES System	147
6.3.3 Comparison of IndusMarker with Automark	149
6.4 Conclusion	150
REFERENCES.....	154
APPENDICES	160
Appendix 1: User Manual.....	160
Appendix 2: Screen Shot of “Examiner’s Main Menu” Screen	166
Appendix 3: Screen Shot of Screen for Asking Examiner to Specify the Number of Questions in the Test	166
Appendix 4: Screen Shot of Screen for Asking Examiner to specify whether or not the Question to be entered has Sub-parts	166
Appendix 5: Screen Shot of Screen for Entering Main Question Text and Number of Sub-parts.....	167
Appendix 6: Screen Shot of Screen for Sub-part Entry	167
Appendix 7: Screen Shot of Screen for Entering Question Text and Question Marks	168
Appendix 8: Screen Shot of “Test Selection” Screen	168
Appendix 9: Screen Shot of “Question Selection” Screen	169
Appendix 10: Screen Shot of “Answer Structure” Formulation & Validation Screen	170
Appendix 11: Screen Shot of “Regions Specification Structure Editor” Screen	171
Appendix 12: Screen Shot of “Practice Results” Screen	172
Appendix 13: Screen Shot of “Individual Student’s Performance” Screen.....	172
Appendix 14: Screen Shot of “Student’s Main Menu” Screen.....	173

Appendix 15: Screen Shot of “Practice Performance History” Screen173
Appendix 16: Screen Shot of “Practice Number Entry” Screen173
Appendix 17: Screen Shot of “Practice’s Question Selection” Screen174

FINAL WORD COUNT=40671 WORDS

List of Tables

Table 1. Tuples for 4 responses.....	28
Table 2. Comparison of human and computerized marking outcomes for the blind experiment.....	40
Table 3. Comparison of human and computerized marking outcomes for the moderation experiment.....	41
Table 4. Summary of the system’s performance on all the allowed short-answer question types.....	114
Table 5. Time taken to formulate and validate the required structures for each of the 6 OOP tests.....	116
Table 6. Example questions for the three question types.....	121
Table 7. Topics covered in each of the three science tests.....	124
Table 8. Summary of the system’s performance on all the allowed short-answer question types for science tests.....	131
Table 9. Key Features of the Two Case Studies.....	136
Table 10. Average marks of students in the final OOP exams taken during the last four terms of the Bahria University’s Islamabad campus.....	139
Table 11. Average marks in grade seven “science” subject at the “Darakhshan campus” and the “PECHS Prep Boys-A” branches of the City School.....	140
Table 12. IndusMarker’s performance with and without consideration of linguistic features.....	143
Table 13. Comparison of IndusMarker’s and the Oxford-UCLES system’s evaluations.....	148

List of Figures

Figure 1. Automark’s Architecture.....	37
Figure 2. Structure of an example mark scheme template	38
Figure 3. An overview of the system architecture	61
Figure 4. Architecture of the “Answer Text Analyzer”	62
Figure 5. Architecture of the “QAML Structure Editor”	63
Figure 6. “Student’s answer entry” screen along with “spelling correction” dialog	66
Figure 7. Screen shot of the “possibility specification” structure editor.....	82
Figure 8. The contents of the “QAML_POSSIBILITY.xml” file displayed via a GUI screen	88
Figure 9. An IndusMarker’s GUI screen for carrying out student’s answer text analysis.....	119
Figure 10. Graph depicting how average human-system agreement rate varies with the allowed short-answer question types for the science tests. The complexity of the question types increases from left to right.....	132
Figure 11. Graph depicting how the average time taken to formulate the required structures varies with the different short-answer question types.....	134
Figure 12. Architecture of the “answer text analyzer” component after removal of the “linguistic features analyzer” sub-component	142
Figure 13. Illustration of an Automark’s mark scheme template	149

Abstract

Automated short-answer marking cannot “guarantee” 100% agreement between the marks generated by a software system and the marks produced separately by a human. This problem has prevented automated marking systems from being used in high-stake short-answer marking. Given this limitation, can an automated short-answer marking system have any practical application? This thesis describes how an automated short-answer marking system, called IndusMarker, can be effectively used to improve learning and teaching.

The design and evaluation of IndusMarker are also presented in the thesis. IndusMarker is designed for factual answers where there is a clear criterion for answers being right or wrong. The system is based on *structure matching*, i.e. matching a pre-specified structure, developed via a purpose-built *structure editor*, with the content of the student’s answer text. An examiner specifies the required structure of an answer in a simple purpose-designed language called Question Answer Markup Language (QAML). The structure editor ensures that users construct correct required structures (with respect to QAML’s syntax and informal semantics) in a form that is suitable for accurate automated marking.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

The following are four notes on copyright and the ownership of intellectual property rights for this thesis:

1. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
2. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.
3. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
4. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see <http://www.campus.manchester.ac.uk/medialibrary/policies/intellectual-property.pdf>), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see <http://www.manchester.ac.uk/library/aboutus/regulations>) and in The University’s policy on presentation of Theses.

Acknowledgements

The author is thankful to the teachers, teaching assistants and students of Bahria University (Pakistan) and the City School (Pakistan) for participating in IndusMarker's evaluation. The author is deeply indebted to Rosheena Siddiqi for her help in organizing IndusMarker's evaluation at the two institutions. The author is also extremely thankful to the higher authorities of Bahria University and the City School for allowing the use of resources at the two institutions and also for allowing the sharing of some of their final exam records. Finally, the author would like to express his gratitude for his supervisor, Dr. Christopher Harrison, whose help and guidance was always available whenever the author needed it.

Publication and Conference Papers Based on the Research Presented in this Thesis

The following paper has been accepted for publication:

- R. Siddiqi, C. J. Harrison and R. Siddiqi, “Improving Teaching and Learning through Automated Short-Answer Marking,” *IEEE Transactions on Learning Technologies*, preprint, 9 Feb. 2010.

The following are the two conference papers based on the research presented in this thesis:

- R. Siddiqi and C. J. Harrison, “A Systematic Approach to the Automated Marking of Short-Answer Questions,” *Proceedings of the 12th IEEE International Multi topic Conference (IEEE INMIC 2008)*, Karachi, Pakistan, pp. 329-332, 2008.
- R. Siddiqi and C. J. Harrison, “On the Automated Assessment of Short Free-Text Responses,” *Paper presented at the 34th annual conference of the International Association for Educational Assessment (IAEA)*, Cambridge, UK, 2008.

1. Introduction

Advances in computer technology and the widespread availability of ever more sophisticated computer-based systems are gradually changing the way teaching and assessment is undertaken [1]. Educational institutions must explore and effectively utilize opportunities provided by such technologies to enhance the “educational experience” of their students. Automated short-answer marking is one of the technologies that may be usefully explored, and in recent years a number of attempts have been made to automate short-answer marking [2], [3], [4], [5], [6]. The most important problem addressed by all of these systems is the accuracy with which automated marking can be performed. These systems are not reliable enough to be used as the sole marker of students’ tests and exams. It is quite reasonable to suggest that it is very difficult to produce an automated system that is as reliable as a human marker. Given this limitation, can an automated short-answer marking system have any practical application? **The main aim of the research described in this thesis is to devise a way in which short-answer marking technology can be usefully exploited to improve teaching and learning.** The design and evaluation of a short-answer marking system called *IndusMarker* is presented in this thesis.

The purpose of this chapter is to present the case for the research undertaken. The area of research is introduced along with some key terms and concepts that are necessary for the argument in favor of the research undertaken. Section 1.1 explains the term *short-answer questions* and also presents the benefits associated with short-answer questions and practice tests. Section 1.2 discusses the role of students’ information in teacher’s decision making. Section 1.3 describes assessment tasks and outlines purposes behind such tasks. The concept of *classroom assessment* and its advantages are explained in Section 1.4. Section 1.5 summarizes the information presented in earlier sections and elaborates the nature of the research undertaken and presented in this thesis.

1.1 Short-Answer Questions

Questions are an essential component of effective instruction [14]. If questions are effectively delivered, they facilitate student learning and thinking and provide the opportunity for academics to assess how well students are mastering course content [16].

Questions may be categorized as *short-answer*, *multiple choice*, *essay* etc. The two most commonly used categories are multiple-choice and short-answer questions [17].

In the case of the research described in this thesis, “short-answers” implies free text entry, requiring answers that have to be constructed rather than selected, ranging from phrases to 3 to 4 sentences. Moreover, the research described here concerns objective questions rather than subjective questions. Thus, the criterion of right and wrong for an answer must be clear. An example of a short-answer objective question (in the context of a programming language) is: “What is the main difference between structures and arrays?” The correct answer is: “Arrays can only hold multiple data items of the same type, but structures can hold multiple data items of different data types”. Correct student responses are expected to be paraphrases of this concept and therefore, **the primary task of the automated marking system is to recognize which answers are paraphrases of the correct concept and which are not.**

From a learning and retention point of view, short-answer tests are more effective than multiple-choice tests [17]. This argument is supported by a recent research study [18]. The study shows that short-answer questions (that require recall) are more beneficial than multiple choice questions (that require recognition) for subsequent memory performance. The study also reveals that taking a practice test is a more potent learning device than additional study of the target material and it leads to better performance in a subsequent final test. In addition, experimental results also demonstrate that short-answer tests produce greater gains in student performance in final tests than multiple-choice tests. **These findings provide a good rationale for researching the area of automated short-answer marking so that students’ learning and retention of course content can be improved.**

1.2 Role of Students’ Information in Teacher’s Decision Making

“What makes a good teacher?” This question has been debated at least since formal education began, if not long before. It is a difficult question to answer because, as Rabinowitz and Traver [7] pointed out almost a half-century ago, the good teacher “does not exist pure and serene, available for scientific scrutiny, but is instead a fiction of the minds of men”. Some have argued that good teachers possess certain traits, qualities or characteristics.

These teachers are understanding, friendly, responsible, enthusiastic, imaginative and emotionally stable [8].

Since the 1970s, there has been a group of educators and researchers, like Shavelson [9], who have argued that the key to being a good teacher lies in the decisions that teachers make. Shavelson [9] states that “Any teaching act is the result of a decision, whether conscious or unconscious, that the teacher makes after the complex cognitive processing of available information. This reasoning leads to the hypothesis that the basic teaching skill is decision making.” In addition to emphasizing the importance of decision making, Shavelson [9] made a critically important point. Namely, teachers make their decisions “after the complex cognitive processing of available information.” Thus, there is an essential link between available information and decision making. Without information, it is difficult to make good decisions. Although good information does not necessarily produce wise decisions, having access to good information is certainly an asset for the decision maker.

Teachers have many sources of information that can be used in making decisions [10]. The critical issue facing teachers is what information to use and how to use it to make the best decisions possible in the time available. Time is important because many decisions need to be made before the teacher has all the information that they would like to have. The more students’ information that teachers have at the time of decision making, the more likely a better quality of teachers’ decisions results. Timely provision of students’ information to teachers is one of the issues addressed by the author’s research.

The awareness that a decision needs to be made is often stated in the form of a “should” question [10]. Examples of everyday decisions facing teachers:

1. “Should I send a note to Ali’s parents informing them that he constantly interrupts the class and invite them to meet me to discuss the problem?”
2. “Should I stop this lesson to deal with the increasing noise level in the room or should I just ignore it, hoping it will go away?”
3. “What should I do to get Sarah back on task?”
4. “Should I tell students they will have a choice of activities tomorrow if they complete their group projects by the end of the class period?”
5. “What marks should I give Saleem for his answer?”

6. “Should I move on to the next unit or should I spend a few more days teaching the same material again before moving on?”

Although all of these are “should” questions, they differ in three important ways. First, the odd-numbered questions deal with individual students, whereas the even-numbered questions deal with the entire class. Second, the first two questions deal with classroom behavior, the second two questions with student effort, and the third two questions with student achievement. Third, some of the decisions (e.g. Questions 2, 3 and perhaps 6) must be made on the spot, whereas for others (e.g. Questions 1, 4 and to a certain extent 5) teachers have more time to make their decisions. These “should” questions (and their related decisions) then can be differentiated in terms of (a) the *focus* of the decision (individual student or group), (b) the *basis* for the decision (classroom behavior, effort or achievement) and (c) the *timing* of the decision (immediate or longer term). **The author’s research involves designing a system that can aid in teacher’s decision making based on students’ achievement.** The focus of the decision may be an individual student or group and the timing of decision may be immediate or longer term. The system presented in this thesis is not meant for measuring student’s effort or classroom behavior.

How do teachers get the information about students that they need to make decisions? In general, they have three alternatives [10]. First, they can examine information that already exists, such as information included in students’ stored records. The files containing such records typically include students’ grades in previous semesters or year(s), participation in extracurricular activities, health reports and the like. Second, teachers can observe students in their *natural habitats* – as students sit in their classrooms, interact with other students, read on their own, complete written work at their desks or tables, and so on. Finally, they can assign specific tasks to students (e.g. ask them questions, undertake practice tests etc.) and see how well students perform these tasks. This third source of information is referred to as *assessment tasks*. Since the research described in this thesis is about a particular kind of assessment task, it is worthwhile presenting an overview of key features and purposes behind assessment tasks (in general) in the next section.

1.3 Key Features and Purposes behind Assessment Tasks (in general)

The term *assessment task* is more clearly defined next. Following tradition, if teachers want to know whether their students have learned what they were supposed to learn, teachers administer quizzes, tests etc. These assessment instruments typically contain a series of items (e.g. questions to be answered, incomplete sentences to be completed, matches to be made between entries in one column and those in another etc.). In some cases, the instrument may contain a single item. In such cases, this item often requires that the student produce an extended response (e.g. “Write an essay about...”). All the items included on these instruments, regardless of their structure, format or number, are referred to as *assessment tasks* [10].

Different assessment tasks may require different responses from students [10]. The nature of the required response is inherent in the verb included in the task description (“**Write** an essay about...”) or in the directions given to students about the tasks (“**Circle** the option that ...”). In general, these verbs ask students to perform some action (e.g. *write*, *demonstrate*) or select from among alternative possible responses to the task (e.g. *circle*, *choose*). The first set of tasks is referred to as *performance tasks*, whereas the second set of tasks is referred to as *selection tasks*. **The research described in this thesis focuses on a specific kind of performance task called short-answer questions.**

In exploring the purposes of assessment, Rowntree [11] identified six broad categories:

1. Selection
2. Maintaining standards – or quality control
3. Motivation of students
4. Feedback to students
5. Feedback to teachers
6. Preparation for life

To what extent and in what ways do these purposes support student learning? It can be argued that selection and quality control benefit stakeholders other than students, though students need to be assured of the quality of the awards they achieve [12]. Rowntree [11] talks of the “constant prod from assessment” which encourages learning. Thus, the motivational purpose may be said to be more directly related to the needs of students than

other purposes of assessment. However, Rowtree argued that motivational assessment may be seen as an instrument of coercion, a way of getting students to do something they would not do otherwise. In this way, motivational assessment also benefits the teacher. Thus, motivation has two aspects. It encompasses both encouragement and coercion [12]. Feedback is beneficial to both students and teachers, and perceived as such by both students and teachers. **IndusMarker (the automated short-answer marking system described in this thesis) is primarily designed for the third, fourth and fifth purposes listed above i.e. it is designed to motivate students and provide feedback to both teachers and students.** As shown later in chapter 5, IndusMarker's proposed use is to conduct practice tests so that timely feedback may be provided to both teachers and students, and students may be motivated to perform revision of the topics covered earlier. Moreover, IndusMarker is designed for *classroom assessments* which are different from traditional assessments. Classroom assessment and its advantages are explained in the next section.

1.4 Classroom Assessment and its Advantages

Traditionally, assessments are used as evaluation devices that are administered when instructional activities are completed and are used primarily for assigning students' grades. Assessments, however, can also be used to "improve education". Classroom assessments are considered to be well-suited for this task [13]. Classroom assessments are tests, writing assignments etc. that teachers administer on a regular basis in their classrooms. Teachers trust the results from these assessments because of their direct relation to classroom instructional goals. Plus, results are immediate and easy to analyze at the individual student level. To use classroom assessments to make improvements, however, teachers must change both their view of assessments and their interpretation of results. Specifically, they need to see their assessments as an integral part of the instruction process and as crucial for helping students learn.

The term "practice tests" is used to refer to a kind of classroom assessment. Practice tests are low-stake tests taken during term-time. Marks obtained in these tests are not counted towards the final grade of students. The purposes and benefits of practice tests are the same as those of classroom assessments in general. The benefits of classroom assessment

are: (1) it can improve teaching, and (2) it can improve student learning. The two benefits are elaborated in the following two sub-sections:

1.4.1 Importance of Classroom Assessment for Effective Teaching

Effective teachers continually assess their students relative to learning goals and adjust their instruction on the basis of assessment results [14]. Based on assessment results, teachers make diagnostic decisions about individual students as well as group strengths, weaknesses and needs. Information gathered through assessment allows the teacher to diagnose the specific area that needs further attention or where progress is being made. The diagnosis includes an assessment of why a student may be having difficulty so that appropriate instructional activities can be prescribed.

An important aspect of teaching is communicating expectations to students, and assessments are used continuously during instruction to indicate what is expected of students. The nature of the tests teachers give and how they evaluate student answers communicate standards students are expected to meet.

1.4.2 Importance of Classroom Assessment in Improving Student Learning

Classroom assessment occurs during the teaching and learning process rather than after it and has as its primary focus the ongoing improvement of learning for all students [15]. Teachers who assess for learning use classroom assessment activities to involve students directly and more deeply in their own learning, increasing their confidence and motivation to learn by emphasizing progress and achievement rather than failure and non-achievement. In the “assessment for learning” model, assessment is an instructional tool that promotes learning rather than an event designed solely for the purpose of evaluation and assigning grades. Also, when students become involved in the assessment process, assessment for learning begins to look more like teaching and less like testing.

Students are often thought to be passive participants in assessment rather than engaged users of the information that assessment can produce. In the context of classroom assessment, however, students can use assessment to take responsibility for and improve their own learning. Student involvement in assessment does not mean that students control decisions regarding what will or will not be learned or tested. It also does not mean that they

assign their own grades. Instead, student involvement means that students learn to use assessment information to manage their own learning so that they understand how they learn best, know exactly where they are in relation to the defined learning targets, and plan and take the next steps in their learning.

1.5 Utilizing IndusMarker to Improve Teaching and Learning

In this chapter, a number of terms such as “classroom assessment”, “short-answer questions” and “practice tests” have been explained along with their benefits and their relevance to the research undertaken. The role that students’ information can play in enhancing the quality of teacher’s decision making has also been emphasized. Key points are:

1. The basic teaching skill is decision making and without students’ information, it is difficult to make good decisions. It is also important that students’ information is available on time. Assessment is a means of gathering information about students that can be used to aid teachers in the decision-making process.
2. Classroom assessments are tests, writing assignments etc. that teachers administer on a regular basis in their classrooms. Practice tests are a form of classroom assessment. Practice tests are low-stake tests taken during term-time. Marks obtained in these tests are not counted towards the final grade of students.
3. Classroom assessments (such as practice tests) are effective in improving teaching and students’ learning.
4. Experimental reports [18], [19], [20] have repeatedly demonstrated that taking a practice test on studied material promotes subsequent learning and retention of that material on a final test/exam. In addition, practice tests produce learning/retention advantages beyond that enjoyed from repeated study.
5. Short-answer tests produce more robust benefits than multiple choice tests. Studies [18], [20] show that short-answer tests (that require recall) are more beneficial than multiple-choice tests (that require recognition) for subsequent memory performance.

Given the above listed points, it was determined that a short-answer marking system, called IndusMarker, should be developed. IndusMarker can be used to conduct short-answer

practice tests with the aim of supporting improvements in both teaching and students' learning.

IndusMarker exploits *structure matching*, i.e. matching a pre-specified structure, developed via a purpose-built *structure editor* [21], with the content of the student's answer text. The examiner specifies the required structure of an answer in a simple purpose-designed language. The language was initially called *Question Answer Language (QAL)* but was subsequently redefined as a sublanguage of XML and named *Question Answer Markup Language (QAML)*.

Chapter 2 discusses similar systems. Chapters 3 and 4 are devoted to the presentation of QAL, QAML and IndusMarker's system design & implementation. IndusMarker's evaluation is presented in chapter 5 while chapter 6 discusses some important issues such as impact of IndusMarker on students' performance and also presents the conclusion.

2. Similar Systems

An important aim of the research described in this thesis is to develop a novel approach for automatically marking short-answer questions. For an approach to be novel, it is necessary to examine the “state-of-the-art” (and, of course, past important developments if significant) and hence a number of similar systems were studied. Automated free-text marking systems can be divided into two main categories: (1) automated essay marking systems, and (2) automated short-answer marking systems. In 2003, Leacock and Chodorow [2] pointed out that research in the area of automated free-text marking has largely concentrated on essay marking rather than on short-answer marking. This trend is still apparent today.

Essay marking systems award marks based on either the content or style (i.e. “writing quality”) or both. Research in this area has been undertaken since the 1960s but there was no significant success until the mid-1990s. This is because the computing power and software technology required by such systems have only become widely available since the mid-1990s [22]. As a result of technological improvements, many automated essay marking systems have emerged. The most widely known are Project Essay Grade (PEG), e-rater and systems based on Latent Semantic Analysis such as Intelligent Essay Assessor (IEA) [22], [23]. These systems have been used with varying degrees of success by large testing companies, universities and state-owned institutions.

The research described in this thesis is concerned with automated short-answer marking systems and therefore these systems are discussed in detail rather than essay marking systems. Section 2.1 discusses short-answer marking systems in general while Sections 2.2 to 2.4 present overviews of the three short-answer marking systems that exist. As the author has used structure-editing and structure-matching for the purpose of system development, it is also important to provide a brief overview of related techniques i.e. the techniques that could have been used instead of structure-editing and structure matching. Two related techniques are overviewed in Section 2.5. Section 2.6 presents a summary of the related work and associated systems.

2.1 Short-Answer Marking Systems

Short-answer marking systems are designed for short, factual answers where there is a clear criterion for right and wrong (i.e. right and wrong answers). In such systems, the award of marks is based on content rather than style. Poor writing “quality” is normally tolerated. Not all short-answer questions are appropriate for computerized marking. Situations where short answer questions are inappropriate for computerized marking are:

- The correct response may be expressed in a large number of ways (i.e. the short-answer question is subjective).
- Responses are complex in nature (i.e. identification of correct and incorrect answers is not clear-cut).

An example of a short-answer question that is inappropriate for computerized testing is: “*Define the term ‘Democracy’*”. There are numerous standard definitions of the term ‘Democracy’. Moreover, various respondents may have their own perspective on the term and may define it differently. In other words, the expected responses will likely be subjective and not simply paraphrases of a single concept. The criterion of right and wrong for students’ answers is also not very clear.

An example of a short-answer question appropriate for computerized testing is: “*How do we terminate a statement in Java*”. The answer is relatively simple: “*A Java statement is terminated using a semicolon*”. Correct student responses are expected to be paraphrases of this concept and therefore, the primary task of the assessment software is to recognize which students’ answers are paraphrases of the correct concept and which are not.

In some cases, short-answer questions considered unsuitable for computerized tests can be modified and adapted for such tests. The following are two guidelines for modifying initially unsuitable questions:

- The short-answer question should try to constrain students to writing about just one particular fact or concept.
- Longer response items should be broken into smaller, more specific items.

When one modifies a short-answer question to make it amenable to computerized testing, the resulting question(s) are usually not exactly equivalent to the original question. But computerized marking will typically be more accurate on the refined question(s) rather

than on the original one. For example, consider the following two versions of the same question. The first version is not suitable for computerized marking but the second modified version is:

Version 1

Explain the difference between passing primitive data types and passing reference data types as arguments in Java.

Version 2

1. How do we pass primitive data type arguments to a method in Java?
2. When a primitive data type argument is passed, will the changes made to the corresponding parameter be retained after the method returns?
3. What happens in computer memory when a primitive data type argument is passed to a method?
4. How do we pass reference data type arguments to a method in Java?
5. When a reference data type is passed, what will the passed-in reference refer to once the method call has returned?
6. What happens in computer memory when a reference data type argument is passed to a method?

The following three existing short-answer marking systems were examined to understand the “state-of-the-art”:

1. C-rater at Educational Testing Service (ETS) [2], [24],
2. the Oxford-UCLES system at the University of Oxford [3], [4], [5], and
3. Automark at Intelligent Assessment Technologies [6].

Sections 2.2 to 2.4 provide overview of each of these systems.

2.2 C-rater

C-rater is an automated “marking engine” developed by ETS [2], [24]. It is designed to mark short content-based free-text responses. It is not designed to mark “open-ended” questions, such as those that ask respondents for their personal opinion about something.

A question is designed to elicit from the student one or more concepts that constitute the correct answer. However, there is an enormous number of ways that a single concept can be expressed in a natural language. To score short-answer responses, the scoring engine must

be able to recognize when a concept is expressed and when it is not. The set of correct responses are considered as *paraphrases* of the correct answer, and hence the c-rater scoring engine is considered as a *paraphrase recognizer* that identifies the *members* of this *set* (of paraphrases).

For example, consider the question: “*Why is 26th January 2001 an important date for the Indian state of Gujarat?*” Some possible correct student responses are:

- *There was an earthquake in Gujarat on that day.*
- *Many people died in an earthquake.*
- *An earthquake occurred and many people died.*
- *Thousands of people were killed as a result of an earthquake.*

A possible incorrect response is: *There was false news of an earthquake in some parts of Gujarat that panicked people across the state.* Note that there are some words, e.g. *earthquake* and *people* common to both the correct and incorrect responses. The task of C-rater is to identify that the first four responses are paraphrases of the correct concept while the fifth one is not.

2.2.1 C-rater’s Approach to Marking Students’ Responses

A model of the correct answer has to be created by a “content expert”. C-rater’s task is to map the student’s response on to this model and, in so doing, check the correctness of the student’s response. Before this mapping can take place, the student’s response is first converted to a *canonical representation* (i.e. a non-ambiguous, mutually exclusive representation of “knowledge”). C-rater then matches the concept(s) found in the student’s response with those found in the model of the correct answer and makes a decision about the marks based on the number of matches.

In order to generate canonical representations, the variations in the students’ responses have to be *normalized*. The designers of C-rater have identified four primary sources of variations in students’ answers: syntactic variations (e.g. “*The democrats dominate the US congress*” and “*The US congress is dominated by the democrats*”); pronoun reference (e.g. “*Alan bought the cake and ate it*”); morphological variations (e.g. *hide, hides, hided, hidden*) and the use of synonyms and similar words (e.g. *decrease, lessen*,

minimize). Spelling and typographical errors are the fifth source of variation and even though such errors are not considered when studying paraphrases, C-rater needs to correct these errors itself for accurate marking to be possible. A brief overview of how C-rater handles these sources of variations is given below.

In content-based responses, the semantic domain is limited. If a student makes a typing or spelling error in their response, then that error can be automatically corrected because the correct word may easily be identified through the restricted domain that consists of the question, the model answer etc. For example, consider the question: “*Why did Albert Einstein leave Germany and settle in the US in 1933?*” Now, suppose, if someone responds “*Abert Einsien*” instead of “*Albert Einstein*”, then C-rater automatically corrects the spelling of Einstein’s name.

When a misspelled word is submitted along with a student’s response, the spelling correction module of c-rater identifies the misspelled word because it cannot be located in its own dictionary. Once a misspelled word has been identified, the spelling correction module uses the *edit distance algorithm* to compute the number of keystrokes that separate the misspelled word from words in the semantic domain. The misspelled word is replaced with the closest matching word.

This approach to automatically correcting mis-spelt words introduces a problem. Not all mis-spelt words result in non-words. For example, if a student wanted to use the word “race” in his answer and he mistakenly types “rack” and submits the answer without correcting the typing error. The word “rack” itself is a valid English word and will be found by the c-rater spelling correction module in its dictionary. C-rater will be unable to detect such typing mistakes and therefore automatic correction in such cases will not occur.

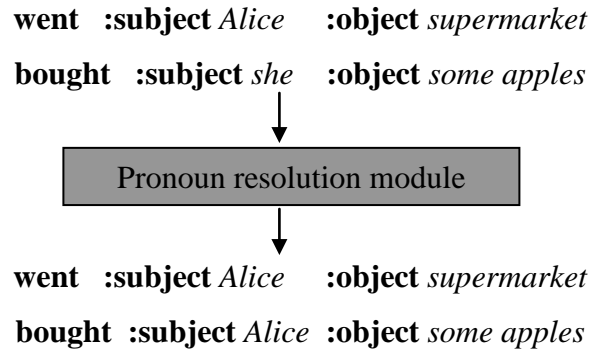
Leacock and Chodorow [2] have found through their research that consideration of word order is very important for automated short-answer marking. Syntactic variation is the major source of paraphrasing. A canonical syntactic representation is created by C-rater which generates a predicate argument structure, or *tuples*, for each sentence of the student’s response. A tuple consists of verb in each clause of a sentence along with its arguments (such as subject and object). For example, consider the question: “*What is the primary function of red blood cells in the human body?*” Table 1 below shows tuples for four

possible responses to this question. The syntax of the three correct responses is different but their tuples are similar i.e. all three have “*Red blood cells*” as the subject of the main clause and “*oxygen*” as the object of main or sub-ordinate clauses. The wording of the fourth answer is similar to that of the first three answers but it is still marked incorrect because the object of this sentence is “*food*” rather than “*oxygen*”.

Table 1. Tuples for 4 responses

Score	Sentence and tuple
Credit	Red blood cells carry oxygen from lungs to body tissues through blood. carry :subject <i>Red blood cells</i> :object <i>oxygen</i>
Credit	Red blood cells travel through our body to deliver oxygen and remove waste. travel :subject <i>Red blood cells</i> :object <i>our body</i> deliver :object <i>oxygen</i> remove :object <i>waste</i>
Credit	Red blood cells have the important job of carrying oxygen. have :subject <i>Red blood cells</i> :object <i>important job</i> carrying :object <i>oxygen</i>
No credit	Red blood cells transports food to various parts of human body. transports :subject <i>Red blood cells</i> :object <i>food</i>

Pronoun resolution is the next important step. The pronoun resolution component of C-rater identifies all the noun phrases that precede the pronoun and all the noun phrases that are in the question. It then decides which noun phrase the pronoun refers to. For example, consider this sentence: “*Alice went to a supermarket where she bought some apples*”. Consider next, below, the predicate-argument structure of this sentence before and after pronoun resolution:



Next, the morphological analysis component converts the inflected and derived forms of words to their base forms. For example, *adds*, *added*, *adding* and *addition* are all inflected and derived forms of the same base form *add*.

Negated words are also converted to their base form e.g. *illiterate* is converted to *literate*. But the meaning is retained as **not** in the tuple. For example, consider the sentence: “*Peter is illiterate*”. Its predicate argument structure is:

be :subject *Peter* :not :object *literate*

This enables C-rater to effectively mark the scope of negation. For example, C-rater is able to differentiate between the following two sentences which a “bag-of-words” approach¹ cannot: “*Alan is literate but Jack is illiterate*” and “*Alan is illiterate but Jack is literate*”.

The final step is how C-rater deals with the use of similar words and synonyms in student’s responses as these words also need to be normalized. C-rater uses a *word similarity matrix* for this purpose [2]. The word similarity matrix has entries for a very large number of English words and with each word there is an associated list of similar word items. Similar words here mean words used in similar contexts. This list also contains some antonyms as antonyms are also often used in similar contexts. It is the task of a content expert to remove antonyms and other inappropriate words from the similar words list. When a student’s response is evaluated, C-rater tries to match each base form in the student’s response with the base forms of the model answer and all the associated similar word lists. If a match is found, then the word in the response is replaced with the word from the model answer.

¹ Approaches, such as Latent Semantic Analysis, that do not use contextual information are termed *bag-of-word* approaches because they treat a response as simply a set of unordered words.

Once a student's response has been converted to a normalized canonical representation, it is then compared with the canonical representation of the model answer. For each relation in the model answer's canonical representation, C-rater tries to find a comparable relation in the canonical representation of the response. There will not always be a one-to-one correspondence between arguments in the canonical representation of the model answer and those in the correct responses. A content expert specifies those elements that are required in a response during the process of building model answers to the questions.

Since many of the students' responses are ungrammatical or fragmentary, the matching algorithm is fairly forgiving. However, in allowing for various degrees of ungrammatical input, there is a tradeoff. If it is strictly enforced, then too many correct answers will be missed. If it is too lax, then the order problem of the "bag-of-words" approach appears and too many incorrect responses are given undue credit.

2.2.2 C-rater's Evaluation

C-rater has been evaluated in two relatively large-scale assessment programs [2]. The first was the National Assessment of Educational Progress (NAEP) Math Online Project. C-rater was used to evaluate written explanations of the reasoning behind particular solutions to some mathematical problems. Five such questions were used in the evaluation process. The second program was the online scoring and administration of Indiana's English 11 End of Course Assessment pilot study. In this case, C-rater was required to assess seven reading comprehension questions. The answers to these questions were more open-ended than those to the questions in NAEP Math Online Project. In these experiments, none of the test questions were designed with C-rater in mind. In fact, those who developed the questions were not even aware of its existence.

In the NAEP assessment, the average length of the responses was 1.2 sentences or 15 words. Between 245 and 250 randomly chosen student responses were scored by two human judges and by C-rater. The average agreement rate between C-rater and the first human judge was 84.4% while between C-rater and the second human judge it was 83.6%. The average agreement rate between the two human judges was 90.8%. This means that C-rater's performance was encouraging in the case of the NAEP assessment.

In the Indiana pilot study, student responses were longer and the average length was around 2.8 sentences or 43 words. One hundred student responses were used for each question and were scored separately by C-rater and a human judge. Leacock and Chodorow [2] summarized the evaluation results: “On average, C-rater and the human readers were in agreement 84% of the time”. The average kappa value² was 0.74. As stated by Fleiss [25], “Values greater than 0.75 or so may be taken to represent excellent agreement beyond chance, values below 0.4 or so may be taken to represent poor agreement beyond chance, and values between 0.4 and 0.75 may be taken to represent fair to good agreement beyond chance”. So, according to standards set by Fleiss [25], C-rater’s performance was good.

C-rater’s errors fall into two categories: *misses* and *false positives*. A *miss* refers to C-rater’s inability to recognize a correct concept in a response. This results in less credit being awarded to the response. A *false positive*, on the other hand, occurs when a C-rater assigns too much credit for a response, i.e. credit is awarded for concept(s) that are not present in the response.

The NAEP and Indiana assessments were also carried out using a “bag-of-words” approach³. Performance dropped by 12% in the case of NAEP assessment and by 30% in the case of Indiana pilot study. The conclusion is that the C-rater’s use of predicate-argument structure and similar words had resulted in its superior performance.

2.3 The Oxford-UCLES System

Many of the *University of Cambridge Local Examinations Syndicate (UCLES)*’s exam questions are short-answer questions which are worth one or two marks and require free text responses of approximately five lines maximum [3], [4], [5]. Such questions are considered to be a useful and integral part of UCLES exams. Automated marking of short-answers is therefore desired by UCLES. An Information Extraction (IE)-based system was developed at Oxford University in an attempt to fulfill this need of UCLES. The project was funded by UCLES and work began in summer 2002. The system’s prototype has been

² Kappa values correct for the level of agreement that is expected by chance.

³ A simple content vector analysis (CVA) classifier based on the Vector space model was used.

evaluated using General Certificate of Secondary Education (GCSE) biology examination answers. The work shares the same aims, and uses many similar techniques (although independently developed) as the systems described in [2] and [6]. As far as it is possible to make sensible comparisons on systems that are not tested using the same data, all of this work achieves comparable levels of accuracy.

Two examples of GCSE Biology short-answer questions (along with their answer keys) suitable for marking by the Oxford-UCLES system are given below:

Example #1

Write down two things about asexual reproduction in plants which is different from sexual reproduction.

Answer key (any two):

- Can be done at any time
- Does not need 2 gametes/parents
- No fertilization
- No meiosis involved
- No genetic variation

Example #2

What is the function of white blood cells?

Answer key (any one):

- Protect the body against disease.
- Safeguard the body against infections.
- Defend the body against both infectious disease and foreign materials.
- Help human body fight against infections.

2.3.1 The Oxford-UCLES System's Approach to Marking Students' Responses

Information Extraction (IE) techniques were adopted for use in the system. According to Sukkariet al. [3], the reasons for this choice were that these techniques do not require complete and accurate parsing, they are relatively robust in the face of

ungrammatical and incomplete sentences and they are also easy to implement. IE techniques are classified in to two categories: ‘knowledge engineering’ and ‘machine learning’. The difference is that in the ‘knowledge engineering’ approach the information extraction patterns are discovered by a human expert while in the ‘machine learning’ approach the patterns are “learned” by the software itself.

The ‘knowledge engineering’ approach is more accurate and requires less training data but it requires considerable skill and effort (and hence time) on the part of the knowledge engineer [26]. On the other hand, the ‘machine learning’ approach is not as accurate as the ‘knowledge engineering’ approach. The ‘machine learning’ approach is suitable when no skilled knowledge engineer is available, training data is plentiful and the highest possible performance is not critical. Both the IE approaches have been tried (one at a time) in the system [3], [4], [5] and the resulting performances have been evaluated. First, the use of the ‘knowledge engineering’ approach in the system is considered below.

The students’ answers are first subjected to *shallow parsing*. The resulting parsed text is then used by the system’s ‘pattern matcher’ component to match it with the hand-crafted patterns. The hand-crafted patterns must conform to the rules set out by the grammar⁴. The result of the pattern matching process is fed into the system’s ‘marker’ component which makes the final decision about the marks.

As already mentioned, a human expert discovers information extraction patterns in the ‘knowledge engineering’ approach. Appelt and Israel [26] specified three crucial steps to accomplish the task of pattern writing by hand:

1. Determine all ways in which target information is expressed in a given corpus.
2. Determine all possible variants of these ways.
3. Write patterns of those ways.

Sukkarieh et al. [3], [4], [5] abstracted patterns over three sets of data: (1) sample answers provided by examiners, (2) answers prepared by themselves, and (3) students’ answers provided by UCLES.

A *pattern* is essentially various *paraphrases* collapsed into one [5]. A set of patterns is associated with each question. This set is further divided into *bags* or *equivalence classes*.

⁴ Grammar here refers to the grammar of the language in which the hand-written patterns should be specified (see [3], [4] for rules of this grammar).

The members of an equivalence class are related by an *equivalence relation* i.e. a member of an equivalence class convey the same message and/or information as other members of the same equivalence class. The marking algorithm compares student's answer with various equivalence classes associated with the question and awards marks according to the number of matches.

The amount of work involved in pattern writing is significant. Human expertise, in both computational linguistic and the exam/test domain, is also required. Automatic customization to new questions is therefore desirable to remove these requirements. Machine learning methods provide ways in which a short-answer marking system can be automatically customized to new questions using a training set of marked answers. A number of machine learning techniques have been tried in the system and their evaluated performances are reported by Sukkarieh et al. [3], [5]. The machine learning techniques that have been tried are: Nearest Neighbor classification, Inductive Logic Programming (ILP), Decision Tree Learning (DTL) and Naïve Bayesian learning (NBayes).

2.3.2 The Oxford-UCLES System's Evaluation

The evaluation of the latest version of the system following the hand-crafted pattern writing approach was carried out using approximately 260 answers for each of the 9 questions taken from a UCLES GCSE biology exam. The full mark for these questions ranged from 1 to 4. 200 marked answers were used as the training set (i.e. the patterns were abstracted over these answers) and 60 unmarked answers were kept for the testing phase. The average percentage agreement between the system and the marks assigned by human examiner was 84% [5]. It was also observed that there was some inconsistency between the marks awarded by human examiner and the marks that should have been awarded if the marking scheme had been followed more carefully. Therefore, the scores awarded by carefully following the marking scheme guidelines were compared with system scores. The average percentage agreement between the two types of scores is 93%. The evaluation results are thus reasonably good and encouraging.

The evaluation results of the application of machine learning techniques in the Oxford-UCLES system shows that while these techniques are promising, they are not

accurate enough at present to replace the hand-crafted pattern matching approach [5]. Currently, such techniques should be used to either aid pattern writing [5] or perhaps act as complementary assessment techniques for extra confirmation.

The system's performance is unsatisfactory in cases where the required degree of inference is beyond the state-of-the-art [5]. The following are some situations where this may occur:

1. **Need for reasoning and making inferences:** for example, a student may answer with *“keep us healthy”* rather than *“protect the body from diseases”*.
2. **Students sometimes use negation of a negation:** for example, the answer *“it is not necessary for a female cat to give birth at a specific time”* is equal to *“a female cat can give birth at any time”*.
3. **Contradictory or inconsistent information:** an example of contradictory information is *“needs photosynthesis and does not need photosynthesis”*. An example of inconsistent information is *“identical twins have the same chromosomes but different DNA”*.

2.4 Automark

Automark has been developed for robust automated marking of short free-text responses [6], [27]. Information Extraction (IE) techniques have been used to extract the concept or meaning behind free text and full effort has been made to make the software system tolerant of errors in typing, spelling, syntax etc. Its marking is primarily based on content analysis but certain style features may also be considered. Sections 2.4.1 and 2.4.2 outline Automark's approach to marking students' responses and its evaluation respectively.

2.4.1 Automark's Approach to Marking Students' Responses

Automark uses *mark scheme templates* to search for specific content in the student answer text. These templates are representatives of valid (or specifically invalid) answers. The templates are developed using an off-line custom written configuration interface. The software system first parses the student answer text and then “intelligently”⁵ matches it with

⁵ But still under the control of some algorithm.

each mark scheme template so that marks for the student answer may be calculated. The answer representation of a mark scheme template may be mapped to a number of input text variations.

Figure 1 (on the next page) illustrates the operation of the system, and shows the main computational operations which the system performs. The Automark system architecture consists of an offline component and an online component. The offline component consists of configuration and storage of mark scheme templates using a customized interface. The online component is concerned with marking a student's answer and consists of a number of stages. First, the student's answer is subjected to syntactic preprocessing that standardizes the student's answer text in terms of spelling and punctuation. Sentence analysis is then performed to decompose answer sentences into syntactic constituents and relationships between these constituents are also identified. The pattern matching module searches for matches between the mark scheme templates and the syntactic constituents of the student text. The results of this matching process are used to formulate feedback. Feedback is in the form of marks but may also include structured feedback (in English) commenting on various features of the student's answer.

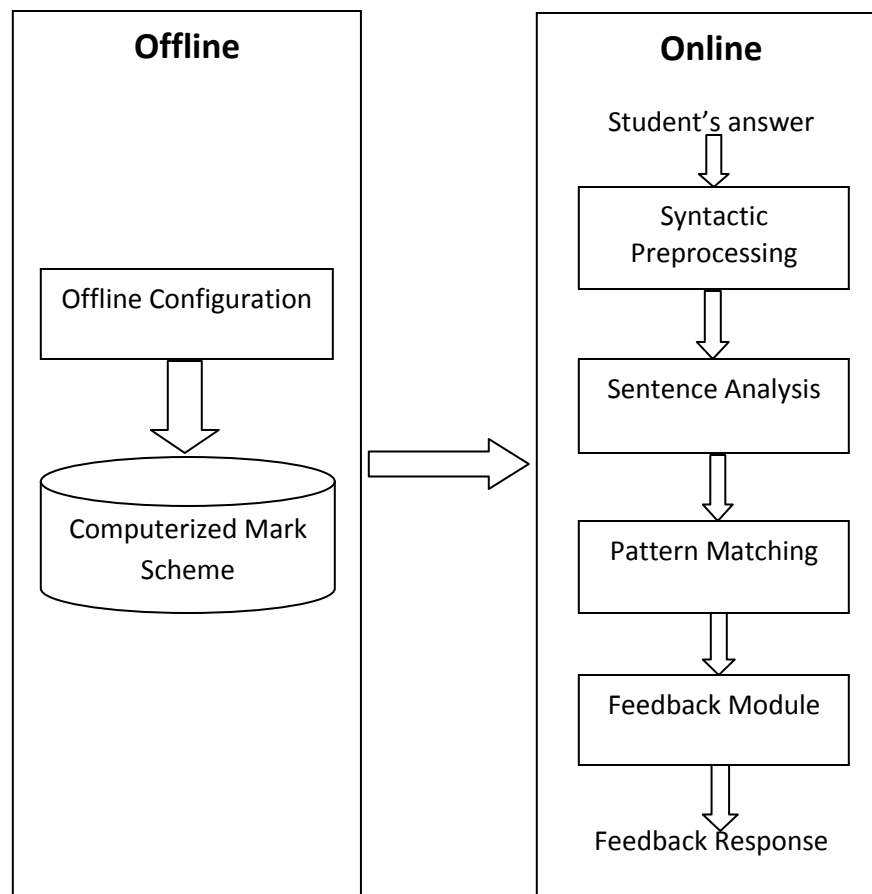


Figure 1. Automark's Architecture

Like human beings, the Automark system attempts to identify the understanding expressed in a student's free-text response, rather than unduly penalizing because of errors in spelling or grammar. Each mark scheme template specifies one particular form of correct or incorrect answers. So, there may be more than one mark scheme template associated with a question.

For example, Figure 2 illustrates a mark scheme template for the answer: **The moon revolves around the earth.** The template is matched against the student's answer and the two entities may be considered matching if the student's answer contains one of the stated verbs (i.e. **rotate, revolve, orbit, travel, move**) with the stated noun (i.e. **moon**) as its subject and **around/round the earth** in its preposition. The verbs used in the student's answer are all *lemmatized* (i.e. converted to the base form e.g. 'went' changed to 'go'), so that the following sentences will be all matched by the template shown in Figure 2:

The moon rotates round the earth.

The moon is orbiting around the earth.

The moon travels in space around the earth.

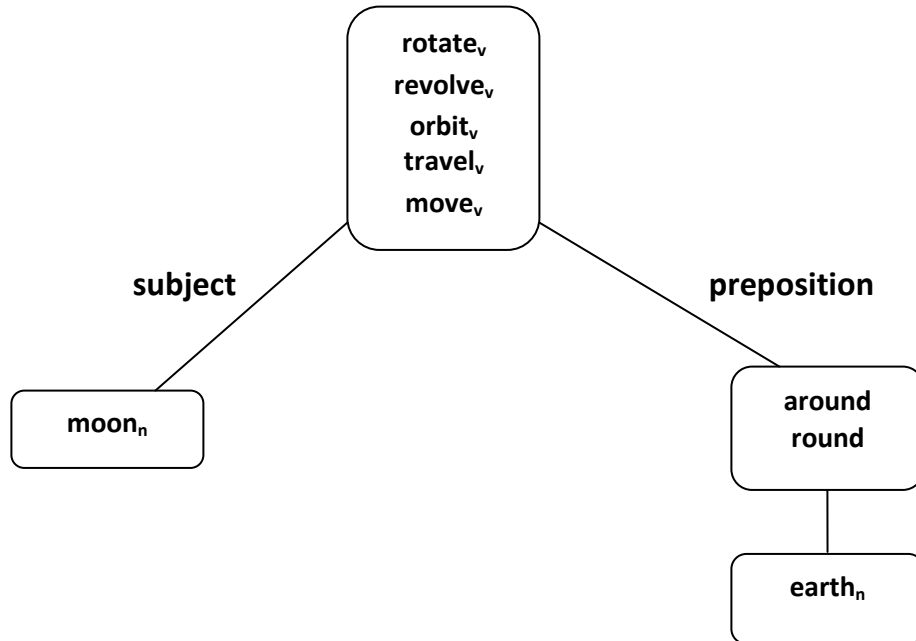


Figure 2. Structure of an example mark scheme template

2.4.2 Automark's Evaluation

Automark was tested in a “real world” scenario of “high importance” tests that were part of the national curriculum assessment of science for pupils at age 11 [6]. The domain had been chosen because in it the likelihood of spelling and syntax errors was very high and therefore through it robustness of the Automark system could be easily tested.

For the purpose of evaluation, four items of varying degrees of open-endedness were selected from the Key Stage 2 1999 Science National Test papers [6]. The form of response of each of the four items, in increasing order of linguistic complexity, was as follows:

- Single word generation (1999, paper B, question 2a)
- Single value generation (1999, paper A, question 7d)
- Generation of a short explanatory sentence (1999, paper A, question 9b)
- Description of a pattern in data (1999, paper B, question 6d)

All four items were scored to one mark except the last, which was scored to two marks. 120 responses were randomly selected for each item. Hand-written pupil responses were transcribed using the exact spelling, syntax and punctuation as written on the test papers. Two experiments were devised to test the software using these data. The first was termed the *blind* experiment and the second one was termed the *moderation* experiment. The two experiments and their results are discussed in Sections 2.4.2.1 and 2.4.2.2 respectively.

2.4.2.1 The Blind Experiment

The blind experiment treated the system as a ‘black box’ and the discrepancies between human and computer marking were analyzed. Mark scheme templates were devised and tested using the model answers from the paper-based mark scheme, augmented by a small number (approximately 50) of answers devised to cover the range of expected pupil responses. The results of this experiment are summarized in Table 2 on the next page.

In the table, ‘n’ represents the number of students’ responses used. It can be easily observed from the evaluation results that Automark’s accuracy decreases substantially as the linguistic complexity of students’ responses increases. Another important point to notice is that the number of false negatives was much higher than the number of false positives.

Table 2. Comparison of human and computerized marking outcomes for the blind experiment

	Paper B, Q. 2a (n=120)	Paper A, Q. 7d (n=120)	Paper A, Q. 9b (n=120)	Paper B, Q. 6d (n=120)	All (n=480)
Item classification	Single word	Single value	Explanatory sentence	Pattern description	
Matches	118	119	111	100	448
% Match	98.3%	99.2%	92.5%	83.3%	93.3%
False positives	1	0	1	0	2
False negatives	1	1	8	20	30

2.4.2.2 The Moderation Experiment

Subsequent to the completion of the blind experiment, the responses used in the blind experiment were used to moderate the unmoderated computerized mark scheme. For the system being described, moderation is required to cope with:

- unexpected but allowable responses;
- unexpected but allowable synonyms;
- and unexpected but allowable phraseology.

Subsequent to moderation, a further test of the marking accuracy was then carried out using the moderated computerized mark scheme. The same data (i.e. students' responses) were used for the moderation experiment and the blind experiment. Consequently, the accuracy figures from the moderation experiment cannot be regarded as indicative of the expected performance on unseen samples. However they do serve the main purpose of the moderation experiment: to identify those errors which are inherent in the software, rather

than those which can be addressed by moderation. Table 3 shows the results of the moderation experiment.

Table 3. Comparison of human and computerized marking outcomes for the moderation experiment

	Paper B, Q. 2a (n=120)	Paper A, Q. 7d (n=120)	Paper A, Q. 9b (n=120)	Paper B, Q. 6d (n=120)	All (n=480)
Item classification	Single word	Single value	Explanatory sentence	Pattern description	
Matches	120	120	118	105	463
% Match	100%	100%	98.3%	87.5%	96.5%
False positives	0	0	1	1	2
False negatives	0	0	1	14	15

Mitchell et al. [6] analyzed the errors encountered in the moderation experiment and this led them to the following conclusions:

- With unmoderated mark schemes, the system generates a number of marking errors when faced with unexpected but allowable responses, synonyms or phraseology.
- The system, after moderation, is able to provide high marking accuracy for items requiring word generation, value generation and short explanatory sentence generation. This is true even with the high incidence of poor spelling and syntax evident in the student responses.
- The system performs less well on the item requiring generation of a description of a pattern in data. This is directly attributable to limitations in the sentence analyzer. More depth and detail is required in identifying the major syntactic relationships within the free-text responses.

- Mitchell et al. [6] believe that the problem of incorrect qualification of correct answers is the most challenging. Students' responses sometimes comprise a correct statement qualified by (or supplemented by) an incorrect statement. Invalid qualifications that negate a correct answer should result in a reduction of the marks awarded. While the characteristics of the set of creditworthy responses may be increased iteratively, algorithms for recognizing nullifying incorrect science may approach the infinite.

2.5 Related Techniques

IndusMarker mainly relies on structure-editing and structure-matching. But other techniques/approaches could have been used instead. Section 2.5.1 and Section 2.5.2 provide brief overview of the two related techniques and the reasons for not selecting them for system development.

2.5.1 Neural Network

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as brain, process information [54]. It is composed of a large number of highly interconnected processing elements (neurons) working together to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. The difference between neural networks and conventional systems is considered next.

Neural networks take a different approach to problem solving than that of conventional systems. Conventional systems use an algorithmic approach i.e. the system follows a set of instructions in order to solve a problem. Unless the specific steps that the system needs to follow are known the system cannot solve the problem. Neural networks, on the other hand, learn by example. They cannot be programmed to perform a specific task. Neural networks are best for situations where the system developers do not fully understand the problem and also do not know exactly how to solve it.

As already stated, pattern recognition is an important application of neural networks. Pattern recognition is used to find patterns and develop classification schemes for data in

very large data sets. Pattern recognition could have been exploited for automated short-answer marking but why wasn't it selected then? There are two important problems that make pattern recognition (using neural network) unsuitable for IndusMarker. Firstly, neural networks require a very large training data set size, and secondly, neural networks require high processing time i.e. they are slow. Another important disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable [54]. Due to these reasons, pattern recognition (using neural network) was not selected for use in IndusMarker.

2.5.2 Text Mining

Text mining, also known as intelligent text analysis or text data mining, refers generally to the process of extracting interesting and non-trivial information and knowledge from unstructured text [55]. The difference between regular data mining and text mining is that in text mining the patterns are extracted from natural language text rather than from structured databases of facts [56]. Databases are designed for programs to process automatically; text is written for people to read. Technology to "read" text (i.e. technology to fully understand text in the same way humans understand text) is not available and it seems highly unlikely that such a technology will be available in the near future. Many researchers think it will require a full simulation of how the mind works before programs that "read" text (like the way humans do) can be written.

However, intelligent use of the techniques taken from fields like machine learning, statistics and computational linguistics enable high-quality information to be derived from text. "High quality" in text mining usually refers to some combination of relevance, novelty and interestingness. Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data and finally evaluation and interpretation of the output. Text mining could have been used for automated marking in IndusMarker but it was not selected because text mining is computationally quite expensive, it requires use of many tools/technologies for various stages of the text mining process and also requires considerably large training data set.

2.6 Summary of the Similar Work

In recent years, a keen interest in automatic content scoring of constructed response items has emerged. Progress in Natural Language Processing (NLP) has made it possible to judge content without having to fully understand the text. A new type of NLP system has emerged employing a technique known as Information Extraction (IE). IE makes use of NLP tools (e.g. parsers, lexical databases, etc.), but rather than attempting an in-depth language analysis, “skims” the input text searching for specific concepts.

Several systems for content scoring exist. However, the only three systems that deal with both short-answers and analytic-based content (i.e. look at content in terms of the main points or concepts expected in an answer) are C-rater at ETS, the Oxford-UCLES system at the University of Oxford and Automark at Intelligent Assessment Technologies. Though the 3 systems were developed independently, they all used a knowledge-engineered IE approach taking advantage of shallow linguistic features that ensure robustness against noisy data (i.e. students’ answers are full of misspellings and grammatical errors). As far as it is possible to make sensible comparisons on systems that are not tested using the same data, the human-system agreement rates for the three systems are comparable. Moreover, none of these systems is accurate and reliable enough to completely replace human marking.

The design and evaluation of the similar systems were discussed in this chapter. The design, evaluation and use of IndusMarker are presented in the next three chapters.

3. “Answer Structure” Specification Languages and IndusMarker’s Architecture

IndusMarker’s design criterion is that IndusMarker should be based on *structure-editing* and *structure-matching* and, unlike other similar systems, reliance on linguistic features analysis should be minimized. Structure-editing is used to support the development of descriptions of correct answer structures for subsequent structure-matching. Structure matching involves matching the pre-specified answer structure with the content of the student’s answer text.

As already stated, an examiner/teacher must specify the required answer structure before IndusMarker performs structure matching. Thus, some means of expressing the required structure was needed. This need was addressed through a new purpose-designed language called *Question Answer Language (QAL)*. QAL was later redefined as a sublanguage of XML (the de facto standard for specifying “semi-structured”⁶ data) and named *Question Answer Markup Language (QAML)*. IndusMarker is designed to use the required structures expressed in QAL/QAML to perform the structure matching process. The purpose of this chapter is to explain the syntax and semantics of QAL and QAML and also to describe IndusMarker’s architecture. QAL and QAML are presented in Section 3.1 and Section 3.2 respectively. The architecture of IndusMarker is presented in Section 3.3.

3.1 The Question Answer Language (QAL)

The syntax and semantics of QAL is intended to be suitable for educators with widely differing computing skills, i.e. QAL is intentionally simple enough to be readily understandable and hence “easy to learn”⁷. To get an overview of how the required structures are written in QAL, consider the following example:

⁶ The term “semi-structured data” is, as its name suggests, used here to denote data that is neither fully-structured (e.g. data represented by relations in a relational database) nor entirely without structure. As such, semi-structured data possesses some structure that can be formally defined, e.g. in terms of the simple algorithmic concepts of sequences, alternatives and iterations (of data elements or other structures).

⁷ The notion that a formal language* (or any other formal system) is “easy to learn” is, of course, problematic. However, both QAL and QAML are significantly less complex than modern (or even earlier) general-purpose programming languages, i.e. both QAL and QAML have significantly, and intentionally, simpler syntax and

Test question: Describe the idea of “function overloading” in a single sentence. (1 mark)

Model answer: Functions having the same name but different signatures.

The examiner must specify the required structure so that all the expected *paraphrases* are elaborated. There are two equally important required parts or *regions* in the model answer for the above question. The first required region is “same name” and the second is “different signatures”. Since both are of equal importance, they are allocated equal marks. The following is the *regions specification* in QAL for the above question:

```
Begin_regions;  
    Begin_region(marks=0.5);  
        "same name"  
    End_region;  
    Begin_region(marks=0.5);  
        "different signature"  
    End_region;  
End_regions;
```

Each region has its own required structure which consists of multiple possibilities (each possibility representing the structure of a possible paraphrase for the region) as shown below:

Region: "same name"

Possibility #1:

```
<main>= ("same"; "name") *2:0.5;;
```

Possibility #2:

```
<main>= ("same"; "identity") *2:0.5;;
```

semantics than such languages. There is evidence, e.g. [28] of the teaching of modern general-purpose programming languages (in the case of the example in this reference the class-based object-oriented general purpose programming language *Java*) to children as young as 9 years old. The author would assume an educator exploiting the IndusMarker system to be at least capable of the kinds of reasoning expected of a nine year old.

* A programming language is a formal language in the sense that it is consciously designed rather than, as in a natural language (e.g. English), “evolving” as a means of communication between humans.

Region: "different signature"

Possibility #1:

```
<main>= ("different"; "data type") *2:0.5;;
```

Possibility #2:

```
<main>= ("different"; "parameter") *2:0.5;;
```

Possibility #3:

```
<main>= ("different"; "signature") *2:0.5;;
```

Possibility #4:

```
<main>= ("different"; "argument") *2:0.5;;
```

The examiner uses a subset of students' answers together with the model answer to predict all acceptable paraphrases for a region. QAL embodies the necessary constructs to express structure for a natural language text. The following is a brief explanation of the QAL's constructs:

1. **Notation:** "....."

Meaning/explanation: Text string.

Example: "polymorphism"

Explanation of the example: "polymorphism" is a text string.

2. **Notation:** <.....>

Meaning/explanation: Main pattern/sub-pattern identifier. A main pattern/sub-pattern has the following form: LHS=RHS. The main pattern/sub-pattern identifier is on LHS. The main pattern/sub-pattern's required structure is specified on RHS. If a pattern has <main> on the LHS, then it is the main pattern for a particular possibility. If a pattern's name on the LHS is something other than "main" e.g. <organize>, then it is a sub-pattern. A possibility always has one main pattern and it may also have one or more sub-patterns. Sub-patterns are called from the main pattern.

Example:

```
<main>=<rbcCount>+<lowHematocrit>:1:;
```

```
<rbcCount>=NP_containing("RBC count" | "red blood cell count");
```

```
<lowHematocrit>=NP_containing("low hematocrit");
```

Explanation of the example: The first pattern in the example above is the main pattern. The next two patterns are sub-patterns. The two sub-patterns are called from the main pattern.

3. **Notation:** +

Meaning/explanation: Sequence. A sequence consists of one or more elements. Elements are listed from left to right and are separated by a “+” symbol. These elements are processed by IndusMarker in the same order as they are listed i.e. one by one from left to right. An element on the left should be matched in the student’s answer text before an element on the right. If some element in a sequence is not matched in the student’s answer text, the whole sequence is deemed to be not matching. An element of a sequence can be any one of the following: (1) “text string” element, (2) “noun phrase required” element, (3) “verb group required” element, (4) “condition” element, (5) “alternative options” element, (6) “allowed permutation” element, (7) “not” element, and (8) “sub-pattern” element.

Example #1: "more"+"one"

Explanation of the example: In the student’s answer text, the text string “more” should appear before the text string “one”. The sequence consists of two elements; both of them are text strings.

Example #2: "upto"+{"programmer", "developer"}

Explanation of the example: The sequence consists of two elements; the first one is a “text string” element while the second is an “alternative options” element. IndusMarker first looks up for the text string “upto” in the student’s answer text and if it is found/matched, then it looks up for the second element. Since the second element is an “alternative options” element, IndusMarker looks for both the words (i.e. “programmer” and “developer”) in the student’s answer text. If any one of the two words (i.e. “programmer” or “developer”) is matched in the student’s answer text, the whole sequence is matched (provided “upto” has been matched earlier). Another important point is that the first element of the sequence should appear in the student’s answer text before the second, otherwise the sequence is not matched.

4. **Notation:** NP_containing(.....)

Meaning/explanation: Noun phrase required. Requirement of a noun phrase containing any of the strings specified in the enclosed brackets. If more than one string is present inside enclosed brackets, then each string is separated from the other through a “|” symbol.

Example: NP_containing("same type" | "same data type")

Explanation of the example: Noun phrase containing either "same type" or "same data type".

5. **Notation:** VG_containing(.....)

Meaning/explanation: Verb group required. Requirement of a verb group containing any of the strings specified in the enclosed brackets. If more than one string is present inside enclosed brackets, then each string is separated from the other through a “|” symbol.

Example: VG_containing("facilitate" | "assist" | "ease" | "help")

Explanation of the example: Verb group containing either “facilitate” or “assist” or “ease” or “help”.

6. **Notation:** [.....]

Meaning/explanation: Condition. The three allowed conditions are NO_WORD, NO_NP and NO_VG. These conditions respectively mean “no word”, “no noun phrase” and “no verb group” allowed at a particular location in the student answer text.

Example #1: <DBMS>+[NO_VG]+<organize>+[NO_NP & NO_VG]+“data”

Explanation of the example: There should be no verb group between the sub-patterns <DBMS> and <organize> and there should also be no noun phrase as well as no verb group between the sub-pattern <organize> and the word “data”.

Example #2: "pointer"+[NO_WORD]+"array"

Explanation of the example: There should be no word between the words “pointer” and “array”.

7. **Notation:** {...}

Meaning/explanation: Alternative options. Any one of the alternatives. Each alternative option is separated from the other through a comma.

Example #1: {"count", "counting"}

Explanation of the example: Any one of the two alternatives.

Example #2: {"type int", "type"&"int"}

Explanation of the example: Either the string "type int" or the string "type" followed by the string "int" (there may be some other string between "type" and "int").

8. **Notation:** (...) *MinNum

Meaning/explanation: Allowed permutation. A specified minimum number of options should appear in the student's answer text in any order. Each option is separated by a semi-colon. Each option is a text string. MinNum should be a positive integer and should not be greater than the number of options listed in the enclosed brackets.

Example #1:

("organization"; "storage"; "access"; "security"; "integrity") *3

Explanation of the example: At least 3 of the 5 options should appear in the student's answer text in any order.

Example #2: ("base"; "constructor") *2

Explanation of the example: Both "base" and "constructor" should appear in the student's answer text in any order.

9. **Notation:** NOT (...)

Meaning/explanation: The word(s) contained in the NOT word list is/are not allowed at a particular location in the student's answer text. If the number of words contained in the NOT word list is more than one, then each word is separated by "|" symbol.

Example:

NP_containing("array")+NOT("structure")+NP_containing("same type" | "same data type")

Explanation of the example: The word "structure" should not appear between the noun phrase containing "array" and the noun phrase containing "same type" or "same data type".

Consider the following two examples to fully understand how patterns are written in QAL and how they should be interpreted:

Example #1

Consider the following pattern written in QAL:

A
B
C
 $\underbrace{\hspace{1.5cm}}$
 $\underbrace{\hspace{3.5cm}}$
 $\underbrace{\hspace{1.5cm}}$
`<main>= ("same"; "identity") *2 : 0.5 ; ;`

An explanation of the marked parts of the pattern is given below:

A=Indicates that this is the main pattern of the possibility. A possibility can also have sub-patterns called from the "main" pattern. This is analogous to the concept of "main" method in Java where program execution starts from the main method but other methods can be called (invoked) from the main method.

B=Indicates that at least two options should appear in at least one of the student's answer sentences (in any order). Since there are only two options (i.e. "same" and "identity"), both should appear in at least one of the student's answer sentences.

C=Indicates the marks that will be added to the student's total if this structure is found in any sentence of the student's answer text.

Example #2

Consider another pattern written in QAL:

C
F
 $\underbrace{\hspace{4.5cm}}$
 $\underbrace{\hspace{4.5cm}}$
A
B
D
E
 $\underbrace{\hspace{3.5cm}}$
 $\underbrace{\hspace{3.5cm}}$
 $\underbrace{\hspace{1.5cm}}$
`<main>={"address", "location"}+"i":1:"i"+{"address", "location":1:1;`

An explanation of the markings on the above pattern is as follows: The above pattern is a main pattern and it consists of two parts. The two parts are marked as "C" and "F". Each main pattern part (i.e. "C" or "F") consists of a required structure and associated marks. "A"

is the required structure and “B” contains the associated marks for the first part of the main pattern. “D” is the required structure and “E” contains the associated marks for the second part of the main pattern. If the required structure of any main pattern part is matched in any sentence of the student’s answer text, the associated marks for that part of the main pattern is added to the student’s total marks.

3.2 The Question Answer Markup Language (QAML)

XML is currently the de facto standard format for data handling and exchange [29]. Another advantage of XML is that the data in XML documents is self-describing. Customized markup languages can also be created using XML and this represents its direct utility. In other words, XML is a *meta-language* and has the ability to define new languages built around a standard format [30], [31]. People, who create these new languages, can also tailor them to their own specific needs. It was decided, for obvious compatibility and associated advantages, to redefine QAL as a sublanguage of XML so that QAL is standardized and all the benefits of XML can be exploited. The new language is called Question Answer Markup Language (QAML).

The first task was to build rules that specify the structure of a QAML document so that the document can be checked to make sure it is set up correctly. There are two types of QAML documents: one type contains “regions specifications” for expected answers and the other type contains “possibility specifications” for a region. Since each region may have multiple possibilities, there may be more than one “possibility specification” document for a region. The following are the Document Type Definitions (DTDs) for “regions specification” and “possibility specification”:

“Regions specification” DTD

```
<!ELEMENT QAML_REGIONS_SPECIFICATION (BODY)>
<!ELEMENT BODY (REGION+)>
<!ELEMENT REGION (TEXT,MARKS)>
<!ELEMENT TEXT (#PCDATA)>
<!ELEMENT MARKS (#PCDATA)>
```

“Possibility specification” DTD

```
<!ELEMENT QAML_POSSIBILITY (MAIN_PATTERN, SUB_PATTERN*)>
<!ELEMENT MAIN_PATTERN (PATTERN_BODY_PART+)>
<!ELEMENT SUB_PATTERN (SUB_PATTERN_NAME, SUB_PATTERN_BODY)>
<!ELEMENT PATTERN_BODY_PART (SEQUENCE, MARKS)>
<!ELEMENT SUB_PATTERN_BODY (SEQUENCE)>
<!ELEMENT SEQUENCE (TEXT | NOUN_PHRASE | VERB_GROUP |
CONDITION | ALTERNATE | MINIMUM_REQUIRED_OPTIONS | NOT |
SUB_PATTERN_NAME)+>
<!ELEMENT NOUN_PHRASE (TEXT+)>
<!ELEMENT VERB_GROUP (TEXT+)>
<!ELEMENT CONDITION (NO_NP | NO_VG | NO_WORD | NO_NP_NO_VG)>
<!ELEMENT NO_NP EMPTY>
<!ELEMENT NO_VG EMPTY>
<!ELEMENT NO_WORD EMPTY>
<!ELEMENT NO_NP_NO_VG EMPTY>
<!ELEMENT ALTERNATE (AND_OPERAND | TEXT)+>
<!ELEMENT AND_OPERAND (TEXT+)>
<!ELEMENT MINIMUM_REQUIRED_OPTIONS (TEXT+, MIN_REQ)>
<!ELEMENT MIN_REQ (#PCDATA)>
<!ELEMENT NOT (TEXT+)>
<!ELEMENT TEXT (#PCDATA)>
<!ELEMENT MARKS (#PCDATA)>
<!ELEMENT SUB_PATTERN_NAME (#PCDATA)>
```

To have a better understanding of how QAML represents the required answer structure for a question, consider the following example:

Test Question: Explain the meaning of the following C++ statement: `int *m[10];` (*1 mark*)

Model Answer: The C++ statement represents declaration of an array of pointers to integer variables. The size of the array is 10.

The following is a possible “regions specification” of the required answer structure for the question:

```
<QAML_REGIONS_SPECIFICATION>
  <BODY>
    <REGION>
      <TEXT>Declaration of an array of pointers to integer
        variables, size 10</TEXT>
      <MARKS>1</MARKS>
    </REGION>
  </BODY>
</QAML_REGIONS_SPECIFICATION>
```

There is only one region in the above QAML “regions specification”. Some other person designing the required structure may decide to have a different “regions specification” e.g. s/he may decide to have more than one region in the “regions specification”. The decision about how many regions to create in a “regions specification” depends mainly on the total number of concepts expected to appear in the student’s answer text and the maximum marks for the question. A designer can choose to have more than one concept in a region. S/he also has to make a decision about the region marks and this depends upon the relative importance of the particular region. The following are the QAML “possibility specifications” for the first two possibilities of the region in the “regions specification”:

Possibility #1:

```
<QAML_POSSIBILITY>
  <MAIN_PATTERN>
    <PATTERN_BODY_PART>
      <SEQUENCE>
        <TEXT>array</TEXT>
```

```

        <TEXT>of</TEXT>
        <TEXT>pointer</TEXT>
    </SEQUENCE>
    <MARKS>1</MARKS>
</PATTERN_BODY_PART>
</MAIN_PATTERN>
</QAML_POSSIBILITY>

```

Possibility #2:

```

<QAML_POSSIBILITY>
    <MAIN_PATTERN>
        <PATTERN_BODY_PART>
            <SEQUENCE>
                <TEXT>pointer</TEXT>
                <CONDITION>
                    <NO_WORD/>
                </CONDITION>
                <TEXT>array</TEXT>
            </SEQUENCE>
            <MARKS>1</MARKS>
        </PATTERN_BODY_PART>
        <PATTERN_BODY_PART>
            <SEQUENCE>
                <TEXT>pointer array</TEXT>
            </SEQUENCE>
            <MARKS>1</MARKS>
        </PATTERN_BODY_PART>
    </MAIN_PATTERN>
</QAML_POSSIBILITY>

```

Consider below another example of how required structures are expressed in QAML:

Test Question: Which C++ operator provides dynamic memory allocation? When this operator is used in an expression to create an object, what does that expression return? If the object is an array, then what does that expression return? (2 marks)

Model Answer: The new operator provides dynamic storage allocation. When this operator is used in an expression to create an object, the expression returns pointer to the object created. If the object is an array, a pointer to the first element is returned.

The following is a possible “regions specification” of the required answer structure for the question (both QAL and QAML representations are given to illustrate how the same specification is expressed in the two languages):

The “regions specification” expressed in QAL

```
Begin_regions;
    Begin_region(marks=0.75);
        "The new operator provides dynamic storage
        allocation"
    End_region;
    Begin_region(marks=0.75);
        "a pointer to the object created"
    End_region;
    Begin_region(marks=0.5);
        "If the object is an array, a pointer to the initial
        element is returned"
    End_region;
End_regions;
```

The “regions specification” expressed in QAML

```
<QAML_REGIONS_SPECIFICATION>
  <BODY>
    <REGION>
      <TEXT>The new operator provides dynamic storage
      allocation</TEXT>
```



```

        <MARKS>0.75</MARKS>
    </REGION>
    <REGION>
        <TEXT>a pointer to the object created</TEXT>
        <MARKS>0.75</MARKS>
    </REGION>
    <REGION>
        <TEXT>If the object is an array, a pointer to the
        first element is returned</TEXT>
        <MARKS>0.5</MARKS>
    </REGION>
</BODY>
</QAML_REGIONS_SPECIFICATION>

```

The above “regions specification” consists of three regions. Each region consists of text and associated marks. Each region’s required structure consists of one or more possibilities. Each possibility is expressed through a “possibility specification”. The “possibility specifications” for each of the three regions are given below (both QAL and QAML representations are given for each “possibility specification” in order to illustrate that the same “possibility specification” can be expressed in both the languages):

Region: "The new operator provides dynamic storage allocation"

Possibility #1:

(The “possibility specification” expressed in QAL is as follows)

```
<main>="new":0.75;;
```

(The “possibility specification” expressed in QAML is as follows)

```

<QAML_POSSIBILITY>
    <MAIN_PATTERN>
        <PATTERN_BODY_PART>
            <SEQUENCE>
                <TEXT>new</TEXT>

```

```

        </SEQUENCE>
        <MARKS>0.75</MARKS>
    </PATTERN_BODY_PART>
</MAIN_PATTERN>
</QAML_POSSIBILITY>

```

Region: "a pointer to the object created"

Possibility #1:

(The “possibility specification” expressed in QAL is as follows)

```

<main>={"pointer", "location", "address"}+{"object", "element"}:0
.75: ("pointer"; "type") *2:0.5:;

```

(The “possibility specification” expressed in QAML is as follows)

```

<QAML_POSSIBILITY>
    <MAIN_PATTERN>
        <PATTERN_BODY_PART>
            <SEQUENCE>
                <ALTERNATE>
                    <TEXT>pointer</TEXT>
                    <TEXT>location</TEXT>
                    <TEXT>address</TEXT>
                </ALTERNATE>
                <ALTERNATE>
                    <TEXT>object</TEXT>
                    <TEXT>element</TEXT>
                </ALTERNATE>
            </SEQUENCE>
            <MARKS>0.75</MARKS>
        </PATTERN_BODY_PART>
    <PATTERN_BODY_PART>
        <SEQUENCE>
            <MINIMUM_REQUIRED_OPTIONS>

```

```

        <TEXT>pointer</TEXT>
        <TEXT>type</TEXT>
        <MIN_REQ>2</MIN_REQ>
    </MINIMUM_REQUIRED_OPTIONS>
</SEQUENCE>
    <MARKS>0.5</MARKS>
</PATTERN_BODY_PART>
</MAIN_PATTERN>
</QAML_POSSIBILITY>

```

Region: "If the object is an array, a pointer to the initial element is returned"

Possibility #1:

(The “possibility specification” expressed in QAL is as follows)

```

<main>={"pointer", "location", "address"}+{"initial", "first"}:0.
5:{"pointer", "location", "address"}+{"begin", "array"}:0.25;;

```

(The “possibility specification” expressed in QAML is as follows)

```

<QAML_POSSIBILITY>
    <MAIN_PATTERN>
        <PATTERN_BODY_PART>
            <SEQUENCE>
                <ALTERNATE>
                    <TEXT>pointer</TEXT>
                    <TEXT>location</TEXT>
                    <TEXT>address</TEXT>
                </ALTERNATE>
                <ALTERNATE>
                    <TEXT>initial</TEXT>
                    <TEXT>first</TEXT>
                </ALTERNATE>
            </SEQUENCE>
        <MARKS>0.5</MARKS>
    </MAIN_PATTERN>
</QAML_POSSIBILITY>

```

```

</PATTERN_BODY_PART>
<PATTERN_BODY_PART>
  <SEQUENCE>
    <ALTERNATE>
      <TEXT>pointer</TEXT>
      <TEXT>location</TEXT>
      <TEXT>address</TEXT>
    </ALTERNATE>
    <ALTERNATE>
      <TEXT>begin</TEXT>
      <TEXT>array</TEXT>
    </ALTERNATE>
  </SEQUENCE>
  <MARKS>0.25</MARKS>
</PATTERN_BODY_PART>
</MAIN_PATTERN>
</QAML_POSSIBILITY>

```

All the three regions (in the above example) have one possibility each and therefore there is one “possibility specification” per region. But a region can have more than one possibility. All the above QAML specifications have to be well-formed and valid. In order for these specifications to be valid, they have to comply with the DTD associated with such specifications. The IndusMarker system has a QAML structure editor that ensures the user builds well-formed and valid QAML specifications.

3.3 IndusMarker’s Architecture

IndusMarker can be roughly divided into two main components: an “answer text analyzer” and a “QAML structure editor”. The system is designed for two types of users: examiners and students. Interactions between the main components and users of the system are depicted in Figure 3 on the next page. The architectural designs of “answer text analyzer” and “QAML structure editor” are discussed in Section 3.3.1 and Section 3.3.2 respectively.

3.3.1 The Answer Text Analyzer's Architecture

This component of IndusMarker has four sub-components: a “spell checker”, a “linguistic features analyzer”, a “structure matcher” and a “marks calculator”. The “linguistic features analyzer” itself has two further sub-components: a “natural language parser” [32] and a “noun phrase and verb group chunker”. Architectural design of the “answer text analyzer” is depicted in Figure 4 on page 62. An “answer text analyzer” performs four main functions: (i) spell-checking, (ii) some basic linguistic analysis – Part Of Speech (POS) tagging and Noun Phrase and Verb Group (NP & VG) chunking, (iii) matching student’s answer text structure with the required structure (as specified in the “regions” and “possibility” specifications), and (iv) computing the total marks of the student for his/her answer based on the result of the matching process.

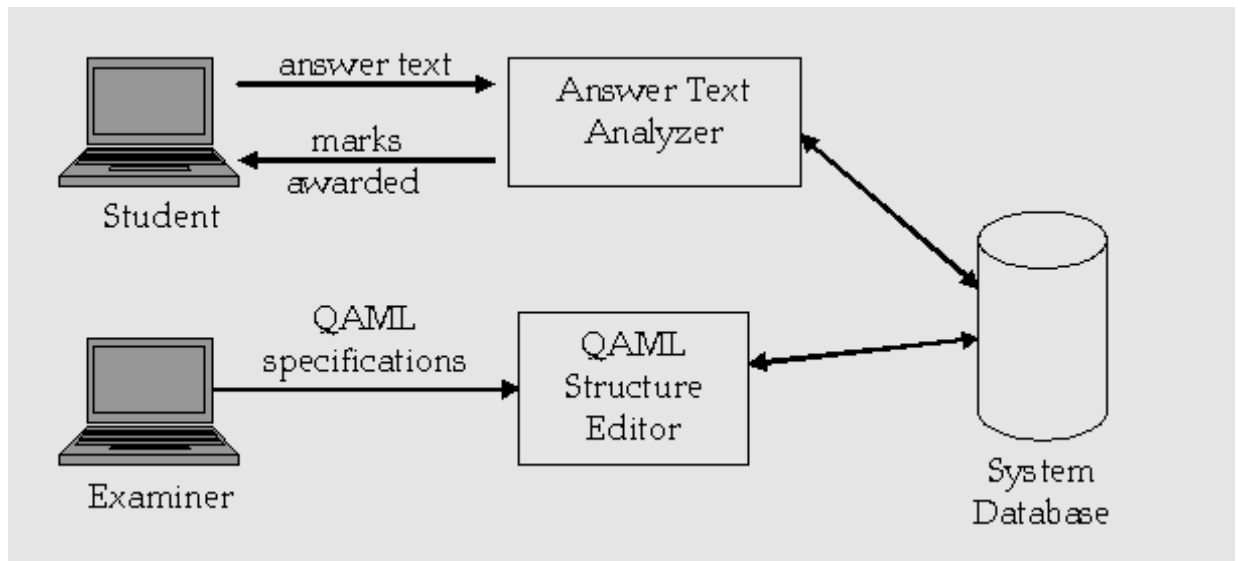


Figure 3. An overview of the system architecture

The spelling mistakes in the student’s answer text are highlighted by the spell checker and correct spelling options are presented to the student for each spelling mistake. It is the responsibility of the student to make the final decision about the correct spelling. Once the student has submitted an answer text, some basic linguistic analysis is performed. A natural language parser and NP & VG chunker are respectively used to perform POS tagging and NP & VG chunking of the student’s answer text. After linguistic analysis, the tagged and chunked student’s answer text is processed by the “structure matcher”. The “structure matcher” matches the pre-specified required “answer structure” with student’s answer text (a

detailed description of the “structure matcher’s” algorithmic design is given in Section 4.1.3 and Section 4.1.4). The result of structure matching is used by “marks calculator” to compute the total marks obtained by the student for his/her answer. The design and development details of the “Answer Text Analyzer’s” components are described thoroughly in Section 4.1.

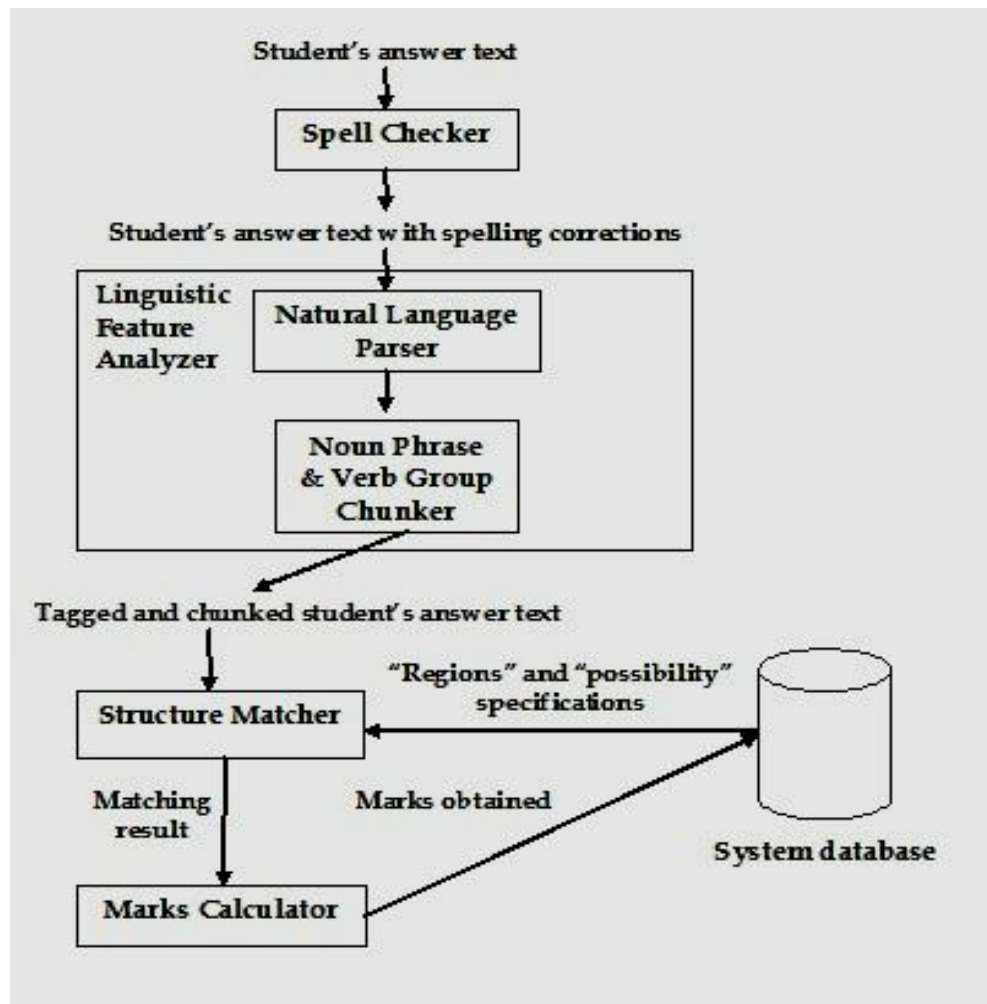


Figure 4. Architecture of the “Answer Text Analyzer”

3.3.2 The QAML Structure Editor’s Architecture

The “QAML structure editor” performs four important functions: (i) provide a suitable Graphical User Interface (GUI) that enables development of structured QAML specifications with relative ease, (ii) automatic QAML document generation, (iii) validation of QAML documents by ensuring that all rules in the associated DTD are followed, and (iv)

transformation of QAML specifications to their respective QAL representations (the reason for this transformation function is that the “answer text analyzer” was designed and implemented for QAL and the idea of QAML came later. Since the “answer text analyzer” had already been developed and tested, there was no need to make any change in this component if the QAML structures can somehow be transformed to QAL equivalents). The architectural design of the “QAML structure editor” is depicted in Figure 5.

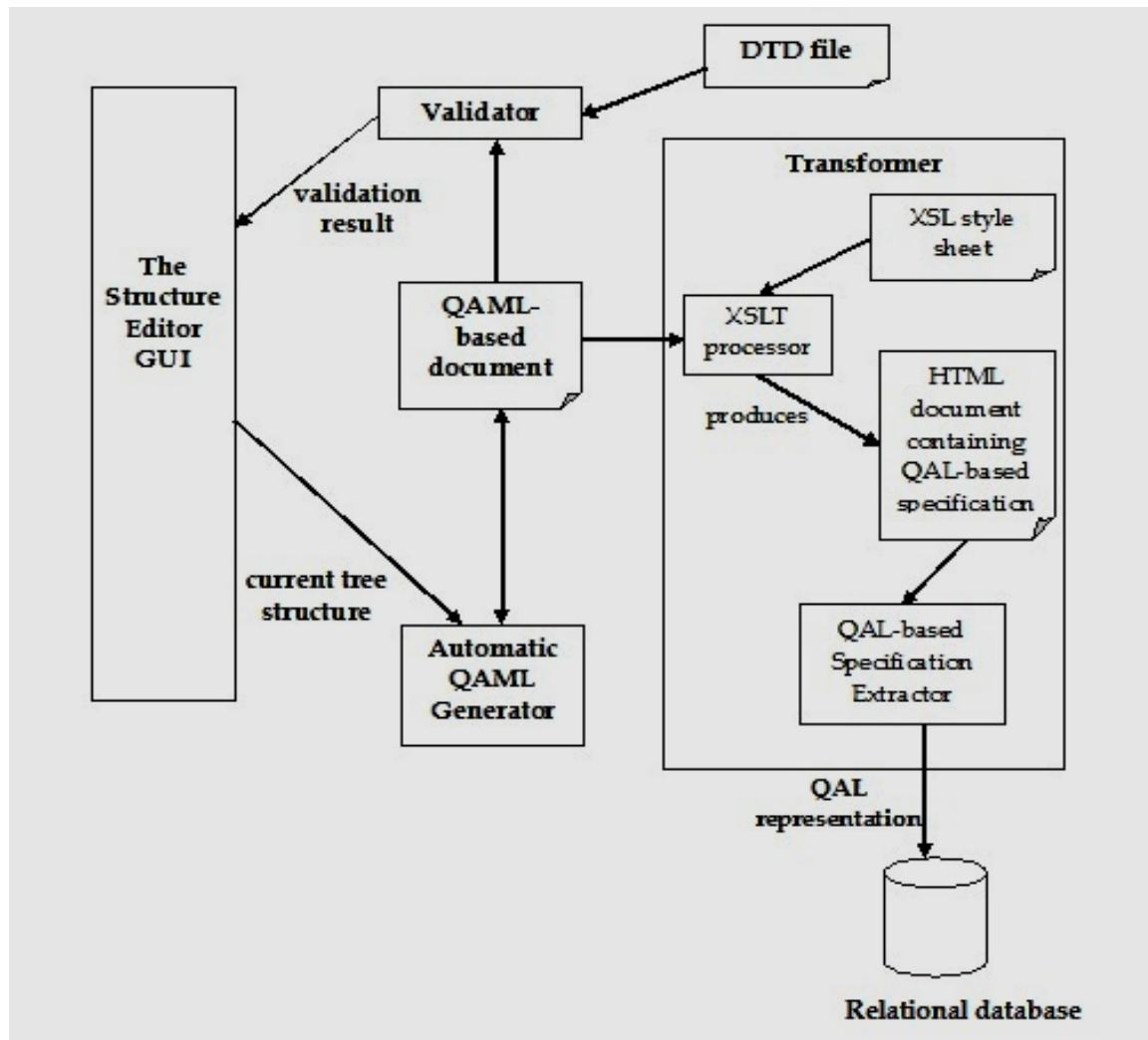


Figure 5. Architecture of the “QAML Structure Editor”

The structure of QAML specification is represented at the user interface as a tree. Nodes may be added or deleted from this tree. Each node represents a QAML element. A leaf node represents either an empty QAML element or parsed character data of a QAML

element. Changes in the tree structure result in corresponding changes in the QAML document.

There are three main sub-components of the “QAML structure editor”: (i) “automatic QAML generator”, (ii) “validator” and (iii) “transformer”. The “automatic QAML generator” serializes the contents of the QAML document. The “validator” is used to verify whether the QAML document conforms to all the constraints specified in DTD. Once the QAML document has been validated, the QAML specifications are converted by the “transformer” to their QAL representation for storage in a relational database. The “transformer” component itself consists of two sub-components: (i) “XSLT processor” and (ii) “QAL-based specification extractor”. The “XSLT processor” takes in QAML-based document and XSL style sheet and produces a HTML document containing QAL-based specification which is then used by a “QAL-based specification extractor” to produce storable QAL representation. The design and development details of the “QAML structure editor’s” sub-components are described thoroughly in Section 4.2.

4. Design and Development of IndusMarker's Components

IndusMarker's development can be sub-divided into two parts i.e. the "answer text analyzer" development and the "QAML structure editor" development. A number of technologies, including Java, XML, DTD, XSL/XSLT, SAX, XPath, the Stanford Parser, JOrtho etc, have been used in the system development. Section 4.1 and Section 4.2 describe how these technologies are utilized. Algorithms developed during the system's design that perform various functionalities within the system, are also described in these sections.

4.1 Design and Development of the Answer Text Analyzer's Sub-Components

As already stated in Section 3.3.1 and depicted in Figure 4, the "answer text analyzer" component itself consists of four sub-components: the "spell checker", the "linguistic features analyzer", the "structure matcher" and the "marks calculator". The purpose of this section is to provide implementation details / algorithms for these sub-components. This section is further divided into four sub-sections. Section 4.1.1 presents implementation of "spell checker", Section 4.1.2 presents implementation of "Linguistic Features Analyzer", Section 4.1.3 presents the overall algorithm for the "structure matcher" and the "marks calculator" while Section 4.1.4 presents the structure matching algorithms used in the "structure matcher" for matching various QAL constructs with student's answer text.

4.1.1 The Spell Checker

The spell-checker used is called JOrtho (Java Orthography) [33]. It is an Open Source spell checker and is entirely written in Java. Its dictionaries are based on the free Wiktionary project [34]. The JOrtho library works with any JTextComponent from the Swing framework. In the case of the system implemented here, the spell checker is registered with JTextArea component in which the student's answer is supposed to be entered. The JOrtho library, when bound to a JTextComponent (such as JTextArea), highlights the potentially incorrectly spelt word and offers a context menu with suggestions

for a correct form of the word. The students, entering their answer, must select a correct spelling before submitting their answer because incorrect spelling will not be automatically corrected once the answer has been submitted. Figure 6 depicts the IndusMarker’s GUI screen for student’s answer entry. The question is displayed on top of the text area provided for student’s answer. The spelling mistakes in student’s answer are highlighted and suggestions for correct spelling are displayed in a separate dialog. The dialog pops up whenever there is a spelling error and the student presses the “F7” button. The dialog allows users to perform a number of functions. Most importantly, it allows the student to select and insert the correct spelling in the answer text.

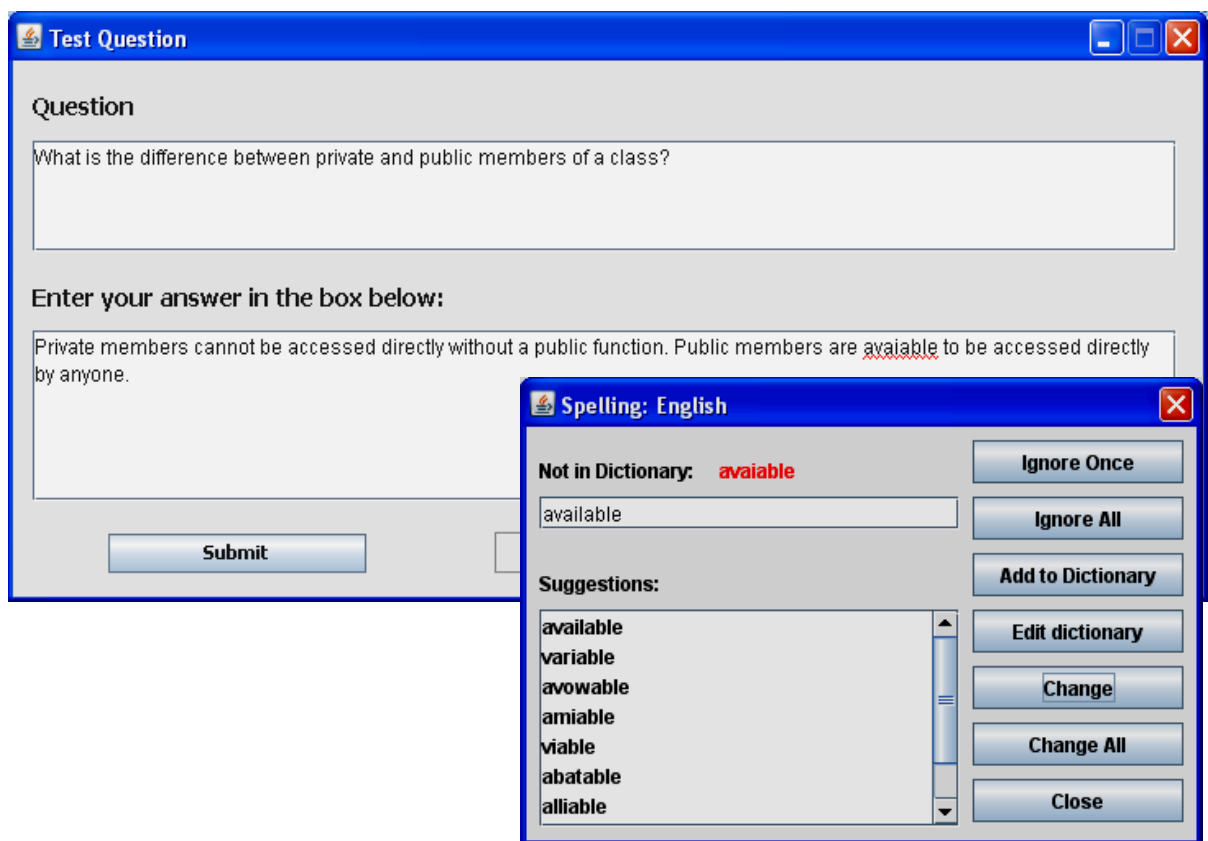


Figure 6. “Student’s answer entry” screen along with “spelling correction” dialog

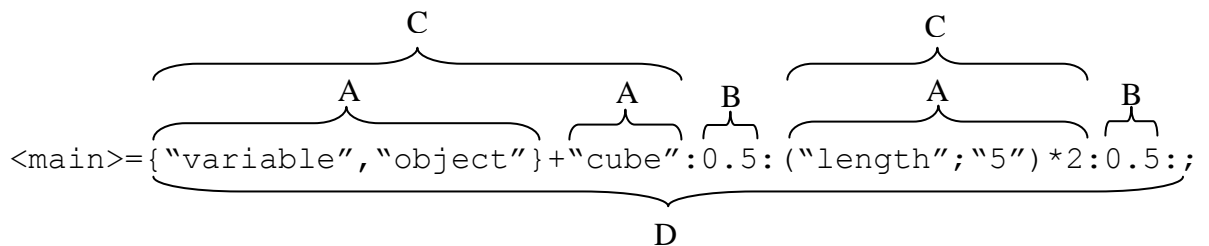
4.1.2 The Linguistic Features Analyzer

To develop the required linguistic features analysis capability, the Stanford Parser [32] and a self-developed Noun Phrase and Verb Group (NP & VG) chunker were exploited. The Stanford Parser is a program that determines the grammatical structure of sentences. It is

a Treebank-trained statistical parser developed by Klien and Manning at Stanford University and is capable of generating parses with high accuracy [35]. Key reasons for choosing the Stanford Parser were: (1) it is written in Java and since the rest of the system's software is also written in Java, the parser is easy to integrate with the system, and (2) the parser is highly accurate. The parser can read plain text input and can output various analysis formats, including part-of-speech tagged text, phrase structure trees and a grammatical relations (typed dependency) format. **In the case of the author's system, the Stanford Parser is used only to get the part-of-speech tagged text output. The tagged text output is then chunked into noun phrases and verb groups by a self-developed NP & VG chunker.**

4.1.3 Overall Algorithm for the Structure Matcher and the Marks Calculator

The "structure matcher" and "marks calculator" were designed during the work undertaken for this thesis. Their main function is to compare the structure of the student's answer text with the structure specified in the related QAL-based specification and based on the result of such a comparison, the student's marks for the answer are computed. Before the sub-components' algorithm is presented, consider the following QAL possibility to understand the notation used in the algorithm's pseudo code:



A=sub-part pattern

B=MAIN_PATTERN_PART_MARKS

C=main-pattern part

D=main-pattern

The above main-pattern has two main-pattern parts. The first main-pattern part has two sub-part patterns. Each main-pattern part has an associated MAIN_PATTERN_PART_MARKS.

The pseudo code for the overall algorithm used in the “structure matcher” and “marks calculator” is given below. A combined algorithm is given because the two components are highly integrated:

```
TOTAL_STUDENT_MARKS_FOR_THIS_ANSWER=0;
FOR each region in the regions specification BEGIN
  MAXIMUM_MARKS_OBTAINED_THIS_REGION_ALL_SENTENCES=0;
  FOR each sentence in the student's answer text BEGIN
    MAXIMUM_MARKS_OBTAINED_THIS_REGION_THIS_SENTENCE=0;
    FOR each possibility of the region BEGIN
      CURRENT_POSSIBILITY_MARKS=0;
      FOR each main-pattern part BEGIN
        PATTERN_MATCHED=true;
        FOR each sub-part of the main-pattern part BEGIN
          IF the sub-part pattern is NOT matched in student's
          answer sentence THEN
            PATTERN_MATCHED=false;
            BREAK this FOR loop;
          END IF
        END FOR
      END FOR
      IF PATTERN_MATCHED=true THEN
        CURRENT_POSSIBILITY_MARKS=
        CURRENT_POSSIBILITY_MARKS+MAIN_PATTERN_PART_MARKS;
      END IF
    END FOR
    IF (MAXIMUM_MARKS_OBTAINED_THIS_REGION_THIS_SENTENCE <
    CURRENT_POSSIBILITY_MARKS) THEN
      MAXIMUM_MARKS_OBTAINED_THIS_REGION_THIS_SENTENCE=
      CURRENT_POSSIBILITY_MARKS;
    END IF
  END FOR
END FOR
```

```

IF      (MAXIMUM_MARKS_OBTAINED_THIS_REGION_ALL_SENTENCES    <
        MAXIMUM_MARKS_OBTAINED_THIS_REGION_THIS_SENTENCE) THEN
        MAXIMUM_MARKS_OBTAINED_THIS_REGION_ALL_SENTENCES=
        MAXIMUM_MARKS_OBTAINED_THIS_REGION_THIS_SENTENCE;
END IF
END FOR
TOTAL_STUDENT_MARKS_FOR_THIS_ANSWER=
TOTAL_STUDENT_MARKS_FOR_THIS_ANSWER+
MAXIMUM_MARKS_OBTAINED_THIS_REGION_ALL_SENTENCES;
END FOR

```

The “sub-part pattern” matching with the student’s answer sentence is a very important step in the above algorithm because the decision about student’s marks is based on the result of this matching. This step is quite involved (as well as important). The novel algorithms developed to perform this structure matching are presented in the next section.

4.1.4 Structure Matching Algorithms for Various QAL Constructs

Structure matching algorithms used in the “structure matcher”, for matching various QAL constructs with the student’s answer text, are described in detail in this section. In the context of the algorithm presented in the previous section, a “sub-part pattern” corresponds to a QAL construct. A QAL construct may be any one of the following: (1) a text string, (2) a “noun phrase required” construct, (3) a “verb group required” construct, (4) a “no noun phrase / no verb group / no word” condition, (5) an “alternative options” construct, (6) an “allowed permutation”, (7) a “not” condition, or (8) a call to a sub-pattern. The “linguistic features analyzer” converts a student’s answer sentence from a simple text string to a Part Of Speech (POS) tagged and Noun Phrase and Verb Group (NP & VG) chunked text. As an example, consider the following student’s answer sentence:

“Arrays can only hold multiple data items of the same type, but structures can hold multiple data items of different data types”.

The above sentence in the POS tagged and NP & VG chunked form is given below:

[Arrays/NNS]/NP can/MD only/RB [hold/VB]/VG [multiple/JJ data/NN items/NNS]/NP of/IN [the/DT same/JJ type/NN]/NP ./, but/CC [structures/NNS]/NP can/MD [hold/VB]/VG [multiple/JJ data/NN items/NNS]/NP of/IN [different/JJ data/NN types/NNS]/NP ./.

The noun phrases and verb groups in the above text are highlighted. There are some words that do not fall into the category of either a noun phrase or a verb group. IndusMarker creates a sequential list containing all the noun phrases, verb groups and words (in the same order as they appear in the student's answer sentence). The sequential list is called ANSWER_SENTENCE_CONSTITUENTS. An entire main pattern part has to match in the student's answer sentence (i.e. ANSWER_SENTENCE_CONSTITUENTS), only then the associated MAIN_PATTERN_PART_MARKS are added to the CURRENT_POSSIBILITY_MARKS. A main pattern part consists of one or more sub-part patterns. These sub-part patterns have to be processed in the sequence they are specified and they should also appear in the student's answer sentence in the same sequence (i.e. order is important). For this purpose, a pointer called CONSTITUENTS_POINTER is maintained. The purpose of the CONSTITUENTS_POINTER is to keep track of the position in the ANSWER_SENTENCE_CONSTITUENTS from where the next sub-part pattern matching should start. The search for the next sub-part pattern begins from the location (or position) just next to the location where the search for the previous sub-part pattern ended. So, if the search for a previous sub-part pattern ended at location *i* of the ANSWER_SENTENCE_CONSTITUENTS list, the search for the next sub-part pattern starts from the location *i*+1. Once the matching process for an entire main pattern part (i.e. all sub-part patterns included in the main pattern part) has finished, the CONSTITUENTS_POINTER is reset to the position of the first element of the ANSWER_SENTENCE_CONSTITUENTS so that the matching process for the next main pattern part can start.

The matching algorithms used to match various forms of sub-part patterns are given below. In each of the matching algorithms presented below, the result of the matching process is indicated through the value of SUB_PART_PATTERN_MATCHED at the end of algorithm execution. If the value of SUB_PART_PATTERN_MATCHED at the end of algorithm execution is TRUE, then the sub-part pattern has been matched in the student's

answer sentence. Otherwise, the sub-part pattern has not been matched in the student's answer sentence.

4.1.4.1 “Text String” Matching Algorithm

The “text string” matching algorithm's pseudo code is given below:

```
SUB_PART_PATTERN_MATCHED=FALSE;
FOR (each element of ANSWER_SENTENCE_CONSTITUENTS from the
location CONSTITUENTS_POINTER) BEGIN
    IF (current element contains the required text string)
    THEN
        SUB_PART_PATTERN_MATCHED=TRUE;
        CONSTITUENTS_POINTER=(location of the current
        element in ANSWER_SENTENCE_CONSTITUENTS + 1);
        BREAK this FOR loop;
    END IF
END FOR
```

4.1.4.2 “Noun Phrase Required” Matching Algorithm

The list of text strings contained in the enclosed brackets of the “noun phrase required” construct is referred to as TEXT_STRINGS_LIST in the following pseudo code for the “noun phrase required” matching algorithm:

```
SUB_PART_PATTERN_MATCHED=FALSE;
FOR (each element of TEXT_STRINGS_LIST) BEGIN
    FOR (each element of ANSWER_SENTENCE_CONSTITUENTS from
    the location CONSTITUENTS_POINTER) BEGIN
        IF (current element of ANSWER_SENTENCE_CONSTITUENTS
        is a noun phrase containing current element of
        TEXT_STRINGS_LIST) THEN
            SUB_PART_PATTERN_MATCHED=TRUE;
```

```

        CONSTITUENTS_POINTER=(location of the current
        element in ANSWER_SENTENCE_CONSTITUENTS + 1);
        BREAK this FOR loop;
    END IF
END FOR
IF (SUB_PART_PATTERN_MATCHED=TRUE) THEN
    BREAK this FOR loop;
END IF
END FOR

```

4.1.4.3 “Verb Group Required” Matching Algorithm

The algorithm for the “verb group required” matching is quite similar to the one used for “noun phrase required” matching. The only difference is that instead of considering noun phrases in ANSWER_SENTENCE_CONSTITUENTS, the algorithm for “verb group required” matching considers verb groups in ANSWER_SENTENCE_CONSTITUENTS. The rest of the algorithm is pretty much the same. The list of text strings contained in the enclosed brackets of the “verb group required” construct is referred to as TEXT_STRINGS_LIST in the following pseudo code:

```

SUB_PART_PATTERN_MATCHED=FALSE;
FOR (each element of TEXT_STRINGS_LIST) BEGIN
    FOR (each element of ANSWER_SENTENCE_CONSTITUENTS from
    the location CONSTITUENTS_POINTER) BEGIN
        IF (current element of ANSWER_SENTENCE_CONSTITUENTS
        is a verb group containing current element of
        TEXT_STRINGS_LIST) THEN
            SUB_PART_PATTERN_MATCHED=TRUE;
            CONSTITUENTS_POINTER=(location of the current
            element in ANSWER_SENTENCE_CONSTITUENTS + 1);
            BREAK this FOR loop;
        END IF
    END IF
END FOR

```



```

END FOR
  IF (SUB_PART_PATTERN_MATCHED=TRUE) THEN
    BREAK this FOR loop;
  END IF
END FOR

```

4.1.4.4 “No Noun Phrase / No Verb Group / No Word” Condition Matching Algorithm

Before the “no noun phrase / no verb group / no word” condition matching algorithm is presented, consider the following QAL possibility to understand the notation used in the algorithm’s pseudo code:

<main>=“pointer”+ [NO_WORD]+“array”:1:;

A=previous sub-part pattern

B=condition

C=next sub-part pattern

Two new variables called START_POSITION and END_POSITION are used. The START_POSITION is the position just after the position where the previous sub-part pattern has finished in the ANSWER_SENTENCE_CONSTITUENTS. The END_POSITION is the position just before the position where the next sub-part pattern starts in the ANSWER_SENTENCE_CONSTITUENTS. The condition applies on positions from START_POSITION to END_POSITION i.e. the condition needs to be true from START_POSITION to END_POSITION of the ANSWER_SENTENCE_CONSTITUENTS. The matching algorithm’s pseudo code is given below:

```

SUB_PART_PATTERN_MATCHED=TRUE;
NO_NP_CONDITION_PRESENT=FALSE;
NO_VG_CONDITION_PRESENT=FALSE;
NO_WORD_CONDITION_PRESENT=FALSE;

```

```

IF (condition contains NO_NP) THEN
    NO_NP_CONDITION_PRESENT=TRUE;
END IF
IF (condition contains NO_VG) THEN
    NO_VG_CONDITION_PRESENT=TRUE;
END IF
IF (condition contains NO_WORD) THEN
    NO_WORD_CONDITION_PRESENT=TRUE;
END IF
Compute START_POSITION and END_POSITION;
IF (NO_WORD_CONDITION_PRESENT=TRUE) THEN
    IF (START_POSITION≠END_POSITION) THEN
        SUB_PART_PATTERN_MATCHED=FALSE;
    END IF
END IF
IF (NO_NP_CONDITION_PRESENT=TRUE OR NO_VG_CONDITION_PRESENT=
TRUE) THEN
    FOR (each position from START_POSITION to END_POSITION)
    BEGIN
        IF          (NO_NP_CONDITION_PRESENT=TRUE          AND
NO_VG_CONDITION_PRESENT=TRUE) THEN
            IF (element at current position of
ANSWER_SENTENCE_CONSTITUENTS is a noun phrase
or a verb group) THEN
                SUB_PART_PATTERN_MATCHED=FALSE;
                BREAK this FOR loop;
            END IF
        END IF
        IF          (NO_NP_CONDITION_PRESENT=TRUE          AND
NO_VG_CONDITION_PRESENT=FALSE) THEN

```

```

        IF      (element      at      current      position      of
ANSWER_SENTENCE_CONSTITUENTS is a noun phrase)
        THEN
                SUB_PART_PATTERN_MATCHED=FALSE;
                BREAK this FOR loop;
        END IF
    END IF
    IF      (NO_NP_CONDITION_PRESENT=FALSE      AND
NO_VG_CONDITION_PRESENT=TRUE) THEN
        IF      (element      at      current      position      of
ANSWER_SENTENCE_CONSTITUENTS is a verb group)
        THEN
                SUB_PART_PATTERN_MATCHED=FALSE;
                BREAK this FOR loop;
        END IF
    END IF
END FOR
END IF

```

4.1.4.5 “Alternative Options” Matching Algorithm

Consider the following QAL possibility to understand the notation used in the “alternative options” matching algorithm:

$$\langle \text{main} \rangle = \left\{ \underbrace{\text{"type int"}}_A, \underbrace{\text{"type"} \& \text{"int"}}_A \right\} : 0.5 ::;$$

A=option

B=sub-option. Each sub-option is separated by a & symbol. An option can have one or more sub-options.

C="alternative options" construct

Two new lists called OPTION_LIST and SUB_OPTION_LIST are used in the algorithm's pseudo code. For the "alternative options" construct in the above QAL possibility, the OPTION_LIST consists of two elements (i.e. "type int" and "type"&"int"). Each element in the OPTION_LIST has its own SUB_OPTION_LIST. The SUB_OPTION_LIST for the first element in the OPTION_LIST consists of only one element (i.e. "type int"). The SUB_OPTION_LIST for the second element in the OPTION_LIST consists of two elements (i.e. "type" and "int"). In other words, a SUB_OPTION_LIST contains all the sub-options of an option. The matching algorithm's pseudo code is given below:

```
SUB_PART_PATTERN_MATCHED=FALSE;
SUB_OPTION_FOUND=FALSE;
FOR (each element of OPTION_LIST) BEGIN
    STARTING_POINTER=CONSTITUENTS_POINTER;
    FOR (each element of SUB_OPTION_LIST8) BEGIN
        SUB_OPTION_FOUND=FALSE;
        FOR (each element of ANSWER_SENTENCE_CONSTITUENTS
            from the location STARTING_POINTER) BEGIN
            IF (current element of SUB_OPTION_LIST is
                present as a sub-string in the current element
                of ANSWER_SENTENCE_CONSTITUENTS) THEN
                SUB_OPTION_FOUND=TRUE;
                STARTING_POINTER=(location of current
                    element in
                    ANSWER_SENTENCE_CONSTITUENTS+1);
                BREAK this FOR loop;
            END IF
        END FOR
    END FOR
```

⁸ The SUB_OPTION_LIST associated with the current element of the OPTION_LIST.

```

        IF (SUB_OPTION_FOUND=FALSE) THEN
            BREAK this FOR loop;
        END IF
    END FOR
    IF SUB_OPTION_FOUND=TRUE THEN
        SUB_PART_PATTERN_MATCHED=TRUE;
        CONSTITUENTS_POINTER=STARTING_POINTER;
        BREAK this FOR loop;
    END IF
END FOR

```

4.1.4.6 “Allowed Permutation” Matching Algorithm

Consider the following QAL possibility to understand the notation used in the “allowed permutation” matching algorithm:

option
MinNum
⏟
┆
 <main>= (“base”; “constructor”) *2:0.5:;

OPTION_LIST is the list of all options in the “allowed permutation” construct. The pseudo code for the “allowed permutation” matching algorithm is given below:

```

SUB_PART_PATTERN_MATCHED=FALSE;
HIGHEST_LOCATION_REACHED=CONSTITUENTS_POINTER;
NUMBER_OF_MATCHES=0;
FOR (each element of OPTION_LIST) BEGIN
    FOR (each element of ANSWER_SENTENCE_CONSTITUENTS from
        the location CONSTITUENTS_POINTER) BEGIN
        IF (current element of OPTION_LIST is present as a
            sub-string in the current element of
            ANSWER_SENTENCE_CONSTITUENTS) THEN
            NUMBER_OF_MATCHES=NUMBER_OF_MATCHES+1;
    END FOR
END FOR

```

```

        IF (HIGHEST_LOCATION_REACHED < location of the
        current ANSWER_SENTENCE_CONSTITUENTS element)
        THEN
            HIGHEST_LOCATION_REACHED=location of the
            current ANSWER_SENTENCE_CONSTITUENTS
            element;
        END IF
        BREAK this FOR loop;
    END IF
END FOR
IF (NUMBER_OF_MATCHES=MinNum) THEN
    SUB_PART_PATTERN_MATCHED=TRUE;
    CONSTITUENTS_POINTER=HIGHEST_LOCATION_REACHED+1;
    BREAK this FOR loop;
END IF
END FOR

```

4.1.4.7 “Not” Condition Matching Algorithm

Consider the following QAL possibility to understand the notation used in the algorithm’s pseudo code:

$$\langle \text{main} \rangle = \underbrace{\text{"P"}}_A + \underbrace{\text{NOT("R" | "S")}}_B + \underbrace{\text{"Q"}}_C : 0.5 : ;$$

A=previous sub-part pattern

B=“not” condition

C=next sub-part pattern

NOT_TEXT_STRING_LIST is the list of all the text strings contained in a “not” condition. Like “no noun phrase / no verb group / no word” condition matching algorithm, the two variables called START_POSITION and END_POSITION are also used in the “not” condition matching algorithm. The START_POSITION is the position just after the

position where the previous sub-part pattern has finished in the ANSWER_SENTENCE_CONSTITUENTS. The END_POSITION is the position just before the position where the next sub-part pattern starts in the ANSWER_SENTENCE_CONSTITUENTS. The “not” condition applies on positions from START_POSITION to END_POSITION i.e. the “not” condition needs to be true from START_POSITION to END_POSITION of the ANSWER_SENTENCE_CONSTITUENTS. The matching algorithm’s pseudo code is given below:

```

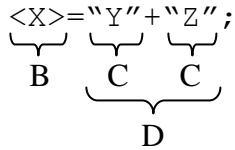
SUB_PART_PATTERN_MATCHED=TRUE;
Compute START_POSITION and END_POSITION;
FOR (each element of NOT_TEXT_STRING_LIST) BEGIN
    FOR (each element of ANSWER_SENTENCE_CONSTITUENTS from
        START_POSITION to END_POSITION) BEGIN
        IF (current element of NOT_TEXT_STRING_LIST is
            present as a sub-string in the current element of
            ANSWER_SENTENCE_CONSTITUENTS) THEN
            SUB_PART_PATTERN_MATCHED=FALSE;
            BREAK this FOR loop;
        END IF
    END FOR
    IF (SUB_PART_PATTERN_MATCHED=FALSE) THEN
        BREAK this FOR loop;
    END IF
END FOR

```

4.1.4.8 “Sub-pattern” Matching Algorithm

A sub-part pattern of a main pattern part can be a call to a sub-pattern. Consider the following QAL possibility to understand this situation:

<main>="W"+^A<X>:0.5;;



- A=call to a sub-pattern
- B=sub-pattern identifier
- C=sub-pattern part
- D=sub-pattern

There is a call to the sub-pattern <X> in the main-pattern. The sub-pattern <X> itself consists of two sub-pattern parts (i.e. “Y” and “Z”). A sub-pattern part may be any one of the following: (1) a text string, (2) a “noun phrase required” construct, (3) a “verb group required” construct, (4) a “no noun phrase / no verb group / no word” condition, (5) an “alternative options” construct, (6) an “allowed permutation”, or (7) a “not” condition. Therefore, appropriate matching algorithm has to be called depending on the type/form of sub-pattern part. For example, if a sub-pattern part is a “noun phrase required” construct, then the “noun phrase required” matching algorithm is called.

```

SUB_PART_PATTERN_MATCHED=TRUE;
FOR (each sub-pattern part) BEGIN
    Call the appropriate matching algorithm for the current
    sub-pattern part;
    IF (current sub-pattern part is NOT matched) THEN
        SUB_PART_PATTERN_MATCHED=FALSE;
        BREAK this FOR loop;
    END IF
END FOR

```

4.2 Design and Development of the QAML Structure Editor's Sub-Components

As already stated in Section 3.3.2 and depicted in Figure 5, the “QAML structure editor” consists of three main components: (1) the Validator, (2) the Automatic QAML

Generator, and (3) the Transformer. The Graphical User Interface (GUI) of the “QAML structure editor” enables users of the system to easily exploit the necessary functionalities. Figure 7 (on the next page) is a screen shot of the “QAML structure editor”. Since the QAML-based specifications are divided into two parts (i.e. the “regions specifications” and the “possibility specifications”), the GUI of the “QAML structure editor” consists of two screens: one for specifying the “regions specifications” and the other for specifying the “possibility specifications”. The screen in Figure 7 is the GUI for the “possibility specification”. The GUI for the “regions specification” is similar to the GUI for the “possibility specification”. The QAML-based “possibility specification” is represented as a tree structure in the GUI. The components of the structure editor are implemented using various XML-related technologies. Implementation details of the main “QAML Structure Editor” components are presented in Sections 4.2.1, 4.2.2 and 4.2.3.

4.2.1 The Validator

The Validator’s functionality is implemented using the Simple API for XML (SAX) [36]. SAX enables Java programs to parse and validate XML documents. SAX uses an event-driven approach. A SAX parser doesn’t wait until the document is completely loaded. Instead, as it traverses the document, it reports back whatever it finds.

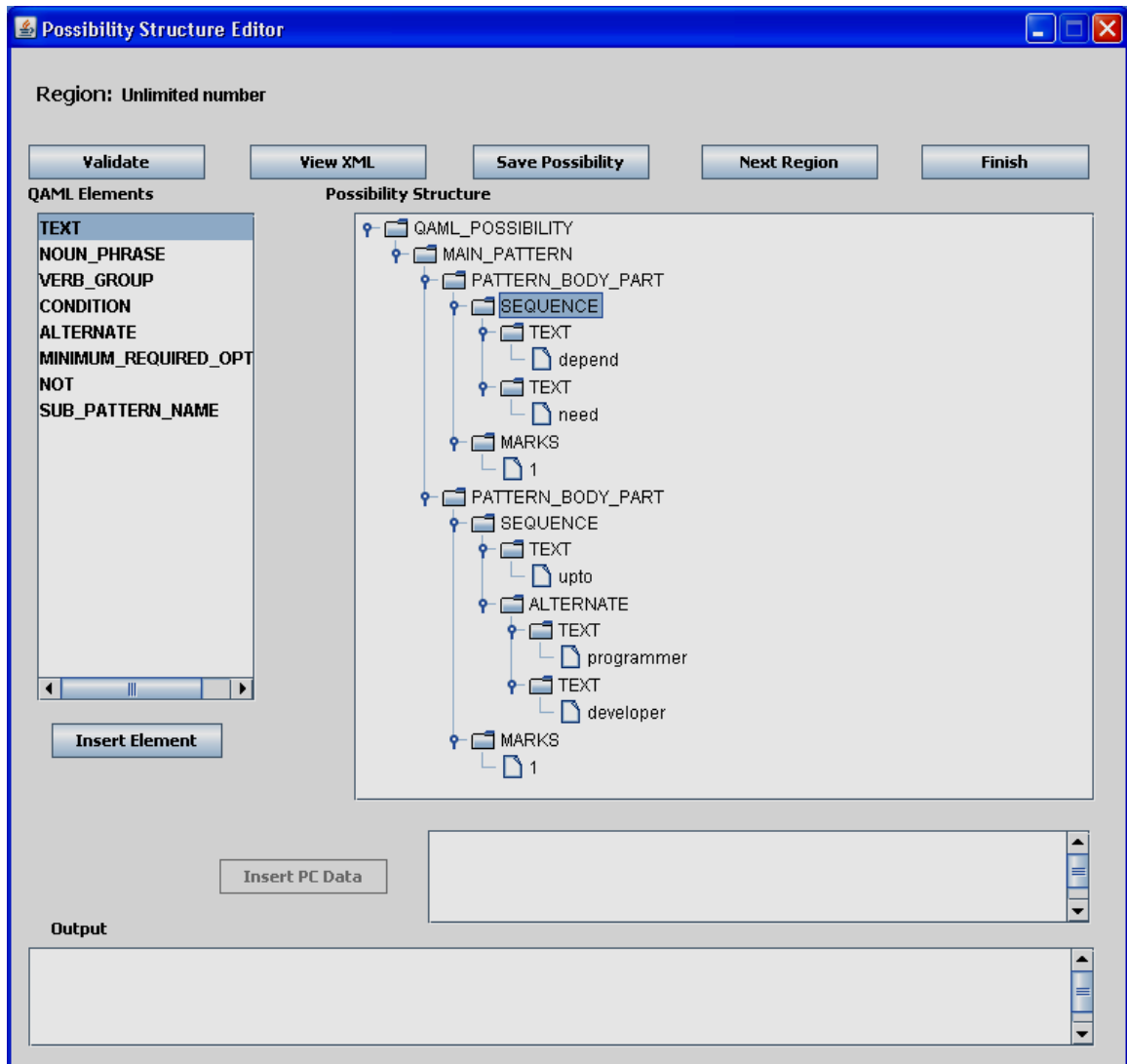


Figure 7. Screen shot of the “possibility specification” structure editor

Selecting and downloading an appropriate SAX parser is an important task in using SAX. The author has used the Apache Xerces parser. The parser comes bundled with SAX 2.0 classes. SAX provides an interface called `org.xml.sax.XMLReader` that all SAX-compliant XML parsers should implement. The Apache Xerces parser’s class that implements the `org.xml.sax.XMLReader` interface is called `org.apache.xerces.parsers.SAXParser`.

The code to instantiate a parser using `org.apache.xerces.parsers.SAXParser` class is given below:

```
XMLReader parser=XMLReaderFactory.createXMLReader(  
"org.apache.xerces.parsers.SAXParser");
```

SAX provides an `ErrorHandler` interface that can be implemented to treat various error conditions that may arise during parsing. The system required a class to be created to implement the `ErrorHandler` interface defined by SAX. The resulting custom error handler was named `MyErrorHandler`. The `MyErrorHandler` instance is registered with the SAX parser using the `setErrorHandler()` method of the `XMLReader` interface:

```
parser.setErrorHandler(errorHandler);
```

The `ErrorHandler` interface has three methods: `warning()`, `error()` and `fatalError()`. This interface allows custom behavior to be attached to the three types of problem conditions that can occur within the lifecycle of XML parsing. Each receives the `SAXParseException` indicating what problem initiated the callback [36]. As the names of the methods indicate, the three types of problem conditions are warnings, errors and fatal errors. There are almost no warnings that can arise as a result of validation being requested. Invalidity in XML documents was considered by the W3C to be important enough to always warrant the generation of an error [36]. Almost all SAX problems received when validating XML are non-fatal errors. This is generated whenever XML constraints are violated. In the case of the system developed here, these constraints are specified in the QAML DTDs. Fatal errors are typically related to a document not being well-formed and are not related to the validity of a document. A document that violates its DTD's constraints will never generate a fatal error.

SAX 2.0 includes the methods needed for setting properties and features in the `XMLReader` interface. The `setFeature()` method is used to turn on validation by supplying the URI specific to setting validation. The following is the code to turn on validation:

```
parser.setFeature("http://xml.org/sax/features/validation",  
true);
```

The parser is now loaded and ready for use. It can now be instructed to parse the QAML document. This is conveniently handled by the `parse()` method of `org.xml.sax.XMLReader`. This method accepts the URI of the QAML document which in this case is the full path to the QAML document. The following is the program statement to parse the QAML document:

```
parser.parse(uri);
```

4.2.2 The Automatic QAML Generator

The “automatic QAML generator” enables users of the structure editor to dynamically view the current status of the QAML documents. The XML Document Object Model (DOM) is used to implement this component. The principle behind programming with DOM is simple. The first stage involves using a parser to translate the XML document into an in-memory tree of objects representing every element and attribute. The methods in these objects’ interfaces can be used to navigate around the document, extract information and modify these objects’ content. The in-memory object hierarchy can also be converted back to XML [37].

In order to use DOM, the DOM parser needs to be first accessed, and this is done by code that is entirely proprietary to Xerces. The parser is implemented as a `DocumentBuilder` object. This is because what it actually does is build a document object model from the incoming data source, via the method `parse()`, which returns a `Document` object [37]. The code to set up the DOM parser and parse a QAML document to produce an in-memory object hierarchy is given below:

```
DocumentBuilderFactory
factoryObj=DocumentBuilderFactory.newInstance();
DocumentBuilder domBuilderObj=factoryObj.newDocumentBuilder();
FileReader inFileObj=new FileReader("QAML_POSSIBILITY.xml");
Document domObj=domBuilderObj.parse(new
InputSource(inFileObj));
```

The `domObj` (which is an instance of the `Document` object) now contains the in-memory object hierarchy. The desired functionality is that the structure and content of the GUI tree on the structure editor screen should be the same as the in-memory DOM-based tree. The in-memory tree may have different structure and/or content from the GUI tree. In order to make sure that the two trees are synchronized and the updated QAML document is displayed, the following steps are taken: (i) the child nodes of the in-memory tree root node are deleted, (ii) the in-memory tree is re-populated based on the current structure and content of the GUI tree, and (iii) the in-memory tree is then written back to the QAML document.

The code to accomplish the first step is given below. The code given below is for the desired functionality implemented in the “possibility specification” part of the structure editor. The code used to implement the same functionality in the “regions specification” part of the structure editor is quite similar to the following code:

```
NodeList
qamlPossibilityList=domObj.getElementsByTagName("QAML_POSSIBIL
ITY");
Element qamlPossibilityObj=(Element)
qamlPossibilityList.item(0);
if(qamlPossibilityObj.hasChildNodes()){
    NodeList
    mainPatternList=qamlPossibilityObj.getElementsByTagName("
MAIN_PATTERN");
    while(mainPatternList.getLength()!=0){
        Element mainPatternObj=(Element)
        mainPatternList.item(0);
        qamlPossibilityObj.removeChild(mainPatternObj);
    }
}
if(qamlPossibilityObj.hasChildNodes()){
    NodeList
    subPatternList=qamlPossibilityObj.getElementsByTagName("S
UB_PATTERN");
```

```

while(subPatternList.getLength()!=0){
    Element subPatternObj=(Element)
    subPatternList.item(0);
    qamlPossibilityObj.removeChild(subPatternObj);
}
}

```

First, it locates all instances of <QAML_POSSIBILITY> elements and put them in a NodeList. Since the <QAML_POSSIBILITY> is the root node of the in-memory tree (as well as the GUI tree), there will be only one instance of <QAML_POSSIBILITY> element and this is located by supplying 0 to the item() method of the NodeList. Once the <QAML_POSSIBILITY> element has been located, the rest of the above code deals with the removal of child nodes of the <QAML_POSSIBILITY> element. After the execution of the above code, the in-memory tree will be ready for the re-population.

The code for the second step (which is about re-population of the in-memory tree based on the current structure and content of the GUI tree) is quite involved and therefore only a summarized algorithm in the form of pseudo code is presented below:

```

Initialize QUEUE with the root node of the GUI tree.
WHILE QUEUE is NOT EMPTY
    Remove the first element of the QUEUE and store it in a
    variable called CURRENT_NODE.
    Create a new in-memory tree node using the content of the
    CURRENT_NODE and assign it to PARENT_XML_ELEMENT.
    FOR each child of the CURRENT_NODE
        Create a new in-memory tree node using the content
        of the current child of the CURRENT_NODE and assign
        it to CURRENT_XML_ELEMENT.
        Add the current child of the CURRENT_NODE to the end
        of the QUEUE.

```

```
Append the CURRENT_XML_ELEMENT to the
PARENT_XML_ELEMENT. (In effect, this step leads to
the gradual re-population of the in-memory tree).
END FOR
END WHILE
```

Once the in-memory tree has been updated, it needs to be written back to the QAML document file. This is done via a class unique to Xerces called `XMLSerializer`. When an `XMLSerializer` object is created its output format is also specified. This output format defines how the QAML is to be written to the output file. The `setOutputStream()` method of the `XMLSerializer` object is used to point the output stream to a standard Java `FileWriter` object representing the QAML document file. The rest of the task is performed by calling the `serialize()` method of the `XMLSerializer` object. The Java code for all this is given below:

```
OutputFormat of = new OutputFormat("XML", "ISO-8859-1", true);
of.setIndent(1);
of.setIndenting(true);
FileWriter outFileObj=new FileWriter("QAML_POSSIBILITY.xml");
XMLSerializer serializerObj=new XMLSerializer(of);
serializerObj.setOutputStream(outFileObj);
serializerObj.serialize(domObj);
```

After the updated in-memory tree has been written back to the QAML document file, the contents of the QAML document file are then displayed in a separate GUI screen. An example screen shot is depicted in Figure 8 (on the next page). This is how the “automatic QAML generator” component is implemented.

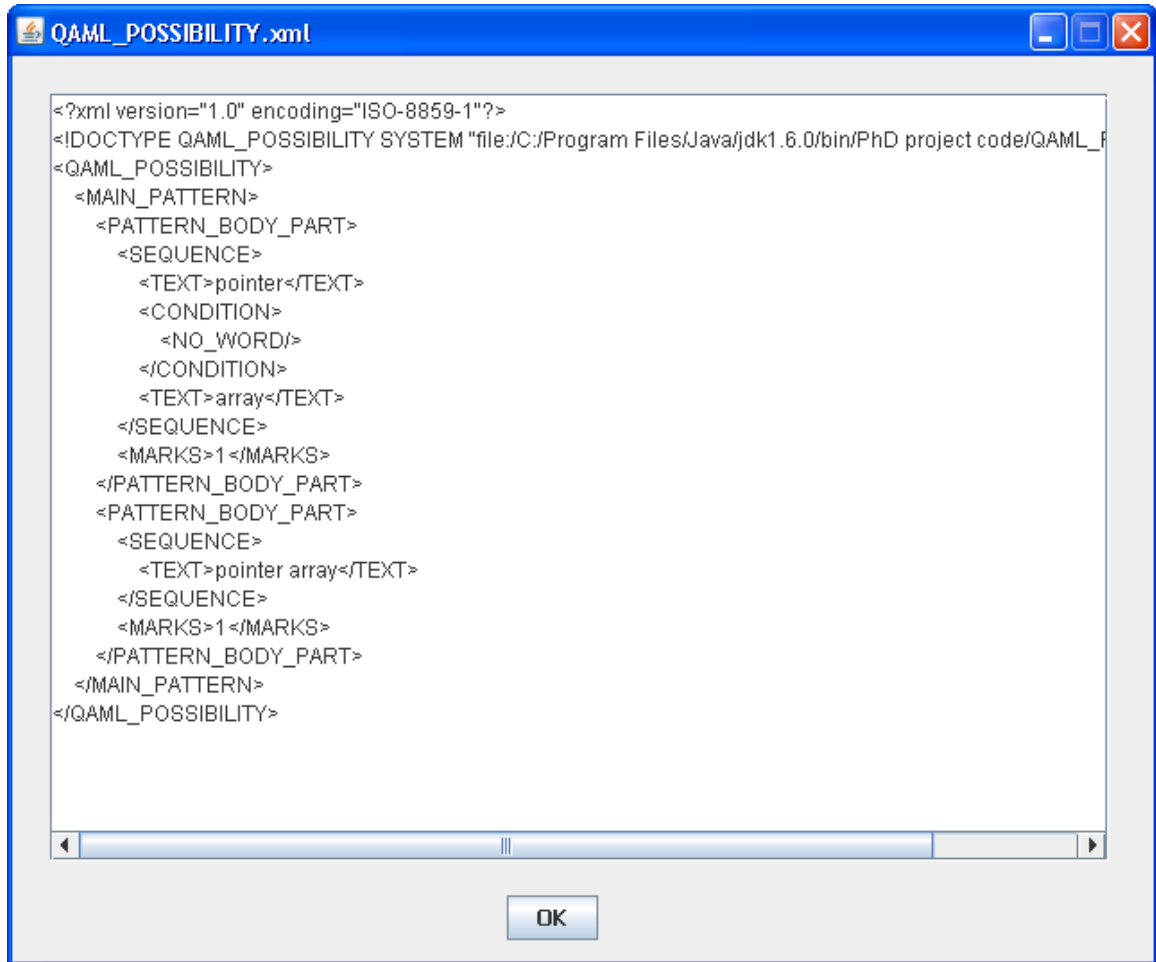


Figure 8. The contents of the “QAML_POSSIBILITY.xml” file displayed via a GUI screen

4.2.3 The Transformer

The QAML-based specifications (both “regions” and “possibility”) need to be stored in a database. **An important point to remember here is that IndusMarker was initially built for Question Answer Language (QAL) and not for QAML. The system had to be adapted for this important change.** A design decision was made that there was no need to change the “answer text analyzer” component. The reason was that the logic and working of the “answer text analyzer” had already been tested with QAL and if somehow the QAML-based specification was transformed to the QAL-based equivalent and this QAL-based equivalent was stored in the system database (instead of the QAML-based specification), then no modification to the “answer text analyzer” component was required. The

transformation of the QAML-based specification to an equivalent QAL-based specification is achieved using the XSL/XSLT and the XPath technologies. The QAL-based specification is then stored in a relational database.

XSL is a language that transforms a document from one format to another [36]. In the case of the “transformer” developed, the XSL style sheet is used to transform a QAML-based specification to an equivalent QAL-based specification. The Apache Xalan XSLT processor takes in the XSL style sheet and the QAML-based document as input and produces an HTML document containing a QAL-based specification. Since the QAL-based specification is in the HTML document, it needs to be extracted and stored in the relational database of the system. This task is carried out by a self-developed “QAL-based Specification Extractor” (a sub-component of the “transformer”).

5. IndusMarker's Evaluation

There were four important objectives for the IndusMarker's evaluation:

1. Verify that the marking algorithm developed is capable of achieving high human-system agreement rates for certain types of short-answer questions. The types of short-answer questions suitable for IndusMarker are defined in Section 5.3.1 for object-oriented programming tests and in Section 5.4.1 for science tests.
2. Analyze errors made by IndusMarker during marking.
3. Check whether the task of "required structure formulation and validation" can be easily carried out in manageable period of time.
4. Verify the feasibility of using IndusMarker to conduct practice tests and as a tool to improve teaching and learning.

IndusMarker was evaluated at two different educational institutions in Pakistan. One was a private school called the City School which offers primary and secondary level education. The other was a public-sector university called Bahria University which offers degree courses in engineering and management sciences. The two institutions have been chosen so that IndusMarker may be evaluated in two different, un-related subjects and at two different levels of education. Before presenting the system evaluation at the two institutions, it is important to consider the kind of knowledge that may be tested through IndusMarker.

5.1 Evaluation Method and Criteria

The author has used empirical methods (i.e. experimental evaluation) to evaluate IndusMarker. Three important evaluation criteria are:

1. IndusMarker should be capable of marking the allowed short-answer question types (listed in Section 5.3.1 and Section 5.4.1) with high human-system agreement rates.
2. Time taken to formulate and validate the required structures should be manageable.
3. The use of IndusMarker to conduct short-answer practice tests should have a positive impact on students' performance in the final exam.

In order to verify that IndusMarker passes these criteria, a systematic strategy is adopted. A number of tests consisting of questions of the allowed short-answer question types have been created and then conducted at two different educational institutions. These tests are both manually and automatically marked (i.e. marked by IndusMarker) so that human-system agreement rates for the various allowed question types can be computed. If the human-system agreement rate for the allowed question types is reasonably high, then IndusMarker is deemed to have passed the first evaluation criteria. The “required structure” formulation and validation time is recorded and if the time taken is manageable, then IndusMarker is deemed to have also passed the second evaluation criteria. Details of students’ performance in the final exam are collected for the term when IndusMarker was used to conduct practice tests and for the previous three terms when it was not used. If the students’ final exam performance is superior in the term when IndusMarker was used compared with the other three previous terms, then IndusMarker is deemed to have passed the third evaluation criteria as well.

5.2 Knowledge Targets and the Design of Short-Answer Questions

Until recently Bloom’s taxonomy [38] provided a definition of *knowledge* for many educators. In this scheme, knowledge is the first, and “lowest,” level of categories in the *cognitive domain*, in which knowledge is defined as “remembering something”. All that is required is that the students recall or recognize facts, definitions, terms, concepts or other information.

The contemporary view of knowledge is that remembering is only part of what occurs when students learn. It is also necessary to think about how the knowledge is represented in the mind of the student⁹. *Knowledge representation* is how information is constructed and stored in long-term and working memory [39]. There are different types of knowledge representations. The type relevant to the author’s research is *declarative knowledge*. Declarative knowledge is information that is retained about something, and, hence, knowing that it exists [14]. At the lowest level, declarative knowledge is similar to Bloom’s first level, i.e. remembering or recognizing specific facts about persons, places,

⁹ The notion of a “representation” of knowledge in the mind of a student is, to say the least, problematic, i.e. we have difficulty seeing into the mind of a student!

events or content in a subject area. The knowledge is represented by simple association or discrimination, such as rote memory. At the highest level, declarative knowledge consists of concepts, ideas and generalizations that are more fully understood and applied. This type of knowledge involves understanding in the form of comprehension or application, the next two levels in Bloom's taxonomy. In other words, declarative knowledge can exist as recall or understanding, depending on the intent of the instruction and how the information is learned. The nature of the representation moves from rote memorization and association of facts to generalized understanding and usage.

Although IndusMarker is primarily designed to assess factual knowledge, it can also be used to assess answers to some forms of applied or comprehension questions. Examples of questions whose answers require understanding that IndusMarker is capable of marking with high accuracy rate are given below:

Example #1:

Consider the following piece of C++ code:

```
#include <iostream>
using namespace std;
class A {
    int data;
public:
    void f(int arg) { data = arg; }
    int g() { return data; }
};
class B : public A { };
int main() {
    B obj; obj.f(20);
    cout << obj.g() << endl;
}
```

- i. *How many data members are contained in class B?*
- ii. *How many member functions are contained in class B?*

iii. *Explain the meaning of the following code fragment:*

```
class B : public A
```

iv. *What will be the output of the program?*

Example #2:

The battery in a torch stores 100 joules of energy and 30 joules are out as light. How much energy is wasted as heat?

Example #3:

If a base class contains a member function `basefunc()`, and a derived class does not contain a function with this name, can an object of the derived class access `basefunc()`?

It is important to understand that if short-answer questions requiring understanding are carefully designed, i.e. if the number of possible answers for the question designed is finite and the “required structure” for possible answers can be easily pre-determined, then short-answer questions requiring understanding can also be assessed through IndusMarker.

5.3 IndusMarker’s Evaluation at Bahria University

Bahria University (BU) is a federally chartered university in Pakistan that is accredited by both Higher Education Commission, Pakistan and Pakistan Engineering Council [40], [41], [42]. IndusMarker was evaluated at BU using Object-Oriented Programming (OOP) tests. The OOP tests were designed keeping the allowed short-answer question types in mind.

Since the release of the Java language more than ten years ago, OOP has become widely used to introduce programming skills to computer science students [43]. The OOP course taught at BU aims to introduce programming and object-oriented concepts to undergraduate students using C++ [44]. The course involves three hours of lectures and three hours of laboratory work in each week of its 16 week duration. The goal of the course is to teach students how to develop object-oriented programs, rather than teach details of the C++ language, i.e. the intention is to use the C++ language merely as a tool for mastering object-oriented programming.

The course is part of many degree programs at BU including Bachelor of Engineering (Electrical), Bachelor of Software Engineering, Bachelor of Computer Engineering and Bachelor of Science (Computer Science). BU has two campuses: one in Karachi and the other one in Islamabad. A typical cohort of students taking the OOP course at a BU campus ranges from 200 to 250 students. English is the medium of instruction and computer-based classrooms (i.e. classrooms where each student is provided with a computer) are available. All computers in these computer-based classrooms have a fast internet connection.

IndusMarker cannot currently mark all types of short-answer questions with a high degree of accuracy. The limitations of the system must be defined and one means of doing so is to clearly define the types of short-answer questions that the system is designed to process.

5.3.1 Allowed Short-Answer Question Types for OOP Tests

The following is the list of OOP short-answer question types that are expected to be marked by the system with a satisfactory degree of accuracy (the question types are listed in increasing order of complexity):

1. “True” or “false” question: This type of question requires student to state whether a particular statement is “true” or “false”. An example of such a question is given below:

State whether the following statements are true or false:

- *A class is an instance of an object.*
 - *The destructor of a class never has any arguments.*
2. Sentence completion: This type of question requires student to supply the missing words in an incomplete sentence. Two examples of such questions are:
 - *The wrapping up of data and member function into a single unit is called _____.*
 - *If there is a pointer p to objects of a base class, and it contains the address of an object of a derived class, and both classes contain a non-virtual member function, ding(), then the statement p->ding(); will cause the version*

of `ding()` in the _____ class to be executed.

3. Single term generation: This type of question requires students to generate a single term. The student's answer may be longer than the required term, but it is the required term that the system is looking for in the answer. Two examples of such questions are:

- *What is the name of a special member function that has no return type and has the same name as the name of the class?*
- *Data abstraction and inheritance are key features of Object-Oriented Programming (OOP). Name one more key feature of OOP.*

4. "Quantity" required: This type of question normally starts with the words "how many" and requires students to specify some quantity. An example of a "quantity required" question is given below:

Consider the following piece of code (of Java programming language):

```
int x=3;
for(int i=0;i<x;i++)
    System.out.println("AAA");
```

How many times the above for loop will iterate?

5. "Numerical value" generation: This type of question requires the generation of a numerical value. An example of such a question is given below:

Consider the following C++ statement:

```
int primes[] = {1,2,3,5,7,11,13};
```

What is the size of the array `primes` declared in the above C++ statement?

6. "Location" required: This type of question requires students to identify a particular location, e.g. a part of a text. An example of such a question is given below:

- *Where do C++ programs begin to execute?*
- *In C++, where do we place the `virtual` keyword to indicate that a function is virtual?*

7. “Program statement output” required: This type of question requires students to generate the output of a print statement contained in a program. Through this the examiner can test student’s understanding of the program. An example of such a question is given below:

Consider the following C++ piece of code:

```
#include <iostream>
using namespace std;
int main(){
    int i=10;
    int *m=&i;
    int n=*m+*m;
    cout << m << "\n";
    cout << n << "\n";
}
```

What will get printed on the console due to the second cout statement?

The student taking the test can only answer the question if s/he has good understanding of the “address of” (&) and “value of” (*) operators.

8. Single phrase generation: This type of question should be answerable through a single phrase although the answer supplied by a student may be longer. Examples of such questions are:

Consider the following C++ statement:

```
float annual_temp[100];
  ↑       ↑       ↑
  X       Y       Z
```

The above C++ statement is a declaration of an array and three of its syntactical parts are highlighted and named as X, Y and Z.

- *What does part X represent?*
- *What does part Y represent?*
- *What does part Z represent?*

9. “Example” required: This type of question requires students to provide examples of a given term, situation etc. The number of possible examples must be finite and easily predictable in advance because it is impossible to correctly mark valid, unpredictable examples. Examples of questions of this type are:
- *Give an example of a relational operator?*
 - *Give an example of an arithmetic unary operator?*
10. List: This type of question requires students to specify reasons, constructs, items, entities etc. that fall under a particular category or satisfy specific conditions. Examples of questions of this type are:
- *List three types of loops available to a C++ programmer.*
 - *Every software object has two characteristics. Name these two characteristics.*
11. Short explanation/description: This type of question requires students to provide a short explanation or description of some statement(s), process, or logic behind a piece of code etc. The explanation or description should ideally be no more than two sentences long. Examples of questions of this type are given below:
- *Describe the idea of “function overloading” in a single sentence.*
 - *Explain the meaning of the following C++ statement: `int *m[10];`*
12. “Situation” or “context” required: This type of question normally starts with “when” and requires students to specify the situation or context in which a particular condition is valid or an event occurs. Two examples of such questions are:
- *When does C++ create a default constructor?*
 - *When do we use the keyword `virtual` in C++?*
13. Definition: This type of question requires students to provide a definition of a specified term. The term must be definable in one or two sentences, however, a student can provide a longer definition. Examples of such questions are:
- *What is a ternary operator?*
 - *What is a variable?*

14. Contrast: This type of question requires students to identify differences between two things. Two examples of such questions are:

- *What is the main difference between a while and a do while statement?*
- *What is the difference between private and public members of a class?*

When designing this type of question, it must be remembered that the number of possible differences should be small (ideally not more than two or three in number). This restriction on the number of differences is necessary in order that the possible difference(s) can be easily predicted by the examiner. The examiner can then specify the “required structure” based on his/her knowledge of possible difference(s).

15. Compare: This type of question requires students to state similarity between two things. Two examples of such questions are:

- *What is the most important similarity between C++ and Java programming languages?*
- *What is the similarity between a base class object and a derived class object?*

The number of possible similarities must again be small in number and easily predictable by the examiner in advance.

16. Composite questions: A question that consists of more than one part with each part itself a question of one of the question types 1-15 above. Two examples of such questions are:

- *What is a ternary operator? Give an example of a ternary operator.*
- *What is the purpose of delete operator? What is called when the delete operator is used for a C++ class object?*

IndusMarker’s evaluation at BU was divided in to two phases: (1) the OOP tests creation and the associated “required structures” formulation and validation at BU Karachi campus, and (2) use of the stored OOP tests and the associated “required structures” to conduct practice tests at BU Islamabad campus.

5.3.2 The OOP Tests Creation and the Required Structures Formulation and Validation Phase

The first task was to design OOP tests suitable for IndusMarker to mark. The lecturers involved were informed about the types of short-answer questions that IndusMarker can mark. Six OOP tests were designed so that the performance of IndusMarker can be satisfactorily evaluated on all types of short-answer questions. 225 students of BU Karachi campus undertook each of the six OOP tests. Two lecturers and six teaching assistants of BU Karachi campus took part in the evaluation process. Teaching assistants carried out manual marking of students' answers while lecturers performed the "required structure" formulation and validation. Both lecturers and teaching assistants had reasonably good knowledge of OOP concepts. The lecturers were provided with guidelines for how to write the "required structures" in QAML using the system's structure editor.

The system was made available on computers of the BU's network. The students' answers were collected through the system and first marked manually by teaching assistants. The students' answers for each question were divided in to two parts: 25 students' answers were kept for the "required structure" formulation and the remaining 200 students' answers were kept for the "required structure" validation. To define the required structure for a question, a lecturer analyzed the structure and content of the model answer and 25 students' answers. Once the "required structure" for a question had been developed using the system's structure editor, the "required structure" was tested using the remaining 200 students' answers.

Marks computed by IndusMarker for a student's answer are compared with marks assigned by the human marker. If the marks assigned by IndusMarker and that assigned by human marker are the same, then IndusMarker's judgment is considered to be correct otherwise it is considered as an incorrect judgment. Human-system agreement rate is calculated based on the ratio of the number of correct judgments to the total number of judgments:

$$r = \frac{c}{t} \times 100$$

where r = human-system agreement rate, c = number of correct judgments and t = total number of judgments.

Questions used in the six OOP tests are presented in the following six sub-sections (i.e. from Section 5.3.2.1 to Section 5.3.2.6) along with the question type and human-system agreement rate for each question. The questions used were based upon a number of OOP topics such as function overloading, dynamic memory allocation, inheritance, polymorphism (virtual functions) etc. Analysis of evaluation results is presented in Section 5.3.2.7.

5.3.2.1 First OOP Test

Questions used in the first OOP test, along with the question type and human-system agreement rate for each question, are presented below:

1. A _____ is an item of data named by an identifier. (Question type: Sentence completion, Human-system agreement rate: 100%)
2. Where do C++ programs begin to execute? (Question type: “Location” required, Human-system agreement rate: 100%)
3. Consider the following piece of code:

```
int x=3;
for(int i=0;i<x;i++)
    cout << "AAA";
```

 - i. How many times the above for loop will iterate? (Question type: “Quantity” required, Human-system agreement rate: 100%)
 - ii. If the for loop condition $i < x$ is changed to $i \leq x$, then what will be the effect? (Question type: Single phrase generation, Human-system agreement rate: 100%)
4. What is the main difference between a while and a do while statement? (Question type: Contrast, Human-system agreement rate: 84.5%)
5. What is the main difference between structures and arrays? (Question type: Contrast, Human-system agreement rate: 86.5%)
6. Consider the following C++ piece of code:

```
#include <iostream>
using namespace std;
```

```

int main(){
    int i=10;
    int *m=&i;
    int n=*m+*m;
    cout << m << "\n";
    cout << n << "\n";
}

```

- i. *What will get printed on the console as a result of the first cout statement?*
(Question type: “Program statement output” required, Human-system agreement rate: 100%)
 - ii. *What will get printed on the console due to the second cout statement?*
(Question type: “Program statement output” required, Human-system agreement rate: 96.5%)
7. *Explain the meaning of the following C++ statement: int *m[10];* (Question type: Short explanation/description, Human-system agreement rate: 96.5%)

5.3.2.2 Second OOP Test

Questions used in the second OOP test, along with the question type and human-system agreement rate for each question, are presented below:

1. *The wrapping up of data and member function into a single unit is called _____.* (Question type: Sentence completion, Human-system agreement rate: 100%)
2. *Consider the following piece of code:*

```

class Y{
    int a;
public:
    Y();
    ~Y();
};

```

Explain the meaning of the following statements in the above code:

- i. $Y()$; (Question type: Short explanation/description, Human-system agreement rate: 94%)
 - ii. $\sim Y()$; (Question type: Short explanation/description, Human-system agreement rate: 97.5%)
3. Describe the idea of “function overloading” in a single sentence. (Question type: Short explanation/description, Human-system agreement rate: 100%)
4. Consider the following piece of code:

```

class Cube{
    int sideLength;
    public:
    Cube(int initialSideLength);
    ~Cube();
    int volume();
    int surfaceArea();
    void printSideLength();
};
Cube::Cube(int initialSideLength) {
    sideLength=initialSideLength;
}
Cube::~~Cube(){}
void Cube::printSideLength(){
    cout << "Side Length = " << sideLength << endl;
}
int Cube::volume(){
    return (sideLength*sideLength*sideLength);
}
int Cube::surfaceArea(){
    return (6*sideLength*sideLength);
}
int main(){
    Cube c(5);

```

```

    c.printSideLength();
    cout << "Surface Area= " << c.surfaceArea() <<
endl;
    cout << "Volume= " << c.volume() << endl;
}

```

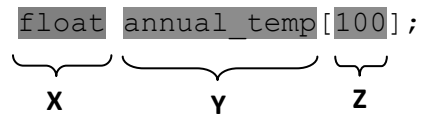
- a. *How many member functions are there in the Cube class (excluding the constructor and destructor)?* (Question type: "Quantity" required, Human-system agreement rate: 100%)
- b. *How many data members are there in the Cube class?* (Question type: "Quantity" required, Human-system agreement rate: 100%)
- c. *Explain the meaning of the following statement (present in the main method of the code under consideration):* `Cube c(5);` (Question type: Short explanation/description, Human-system agreement rate: 88.5%)
- d. *What output will be produced by the following cout statements in the code under consideration:*
 - i. `cout << "Surface Area= " << c.surfaceArea() << endl;` (Question type: "Program statement output" required, Human-system agreement rate: 100%)
 - ii. `cout << "Volume= " << c.volume() << endl;` (Question type: "Program statement output" required, Human-system agreement rate: 100%)
5. *Is it allowed to overload functions on return values?* (Question type: "True" or "false" question, Human-system agreement rate: 100%)
6. *This question is about object-oriented concepts:*
 - i. *Software objects are modeled after _____.* (Question type: Sentence completion, Human-system agreement rate: 100%)
 - ii. *Every software object has two characteristics. Name these two characteristics.* (Question type: List, Human-system agreement rate: 91.5%)
7. *What is the difference between a constructor and a function?* (Question type: Contrast, Human-system agreement rate: 91%)

5.3.2.3 Third OOP Test

Questions used in the third OOP test, along with the question type and human-system agreement rate for each question, are presented below:

1. Consider the following C++ statement:

```
float annual_temp[100];
```



The above C++ statement is a declaration of an array and three of its syntactical parts are highlighted and named as X, Y and Z.

- i. What does part X represent? (Question type: Single phrase generation, Human-system agreement rate: 100%)
 - ii. What does part Y represent? (Question type: Single phrase generation, Human-system agreement rate: 82.5%)
 - iii. What does part Z represent? (Question type: Single phrase generation, Human-system agreement rate: 89.5%)
2. Consider the following C++ statement:

```
int primes[] = {1, 2, 3, 5, 7, 11, 13};
```

What is the size of the array primes declared in the above C++ statement?
(Question type: “Numerical value” generation, Human-system agreement rate: 96.5%)
 3. List three types of loops available to a C++ programmer. (Question type: List, Human-system agreement rate: 92.5%)
 4. State whether the following statements are true or false.
 - i. A class is an instance of an object. (Question type: “True” or “false” question, Human-system agreement rate: 100%)
 - ii. An object is the definition of a class. (Question type: “True” or “false” question, Human-system agreement rate: 100%)
 - iii. The wrapping up of data and member function into a single unit is called encapsulation. (Question type: “True” or “false” question, Human-system agreement rate: 100%)

- iv. *The destructor of a class never has any arguments.* (Question type: “True” or “false” question, Human-system agreement rate: 100%)
5. *How many constructors can be created for a class?* (Question type: “Quantity” required, Human-system agreement rate: 96%)
6. *What is the difference between a data member and a local variable inside a member function?* (Question type: Contrast, Human-system agreement rate: 83.5%)
7. *What is a ternary operator? Give an example of a ternary operator.* (Question type: Composite questions, Human-system agreement rate: 91%)
8. *Consider the following C++ program:*

```
#include <iostream>
using namespace std;
void f(int a){
    cout << "a = " << a << endl;
    a = 10;
    cout << "a = " << a << endl;
}
int main() {
    int x = 20;
    cout << "x = " << x << endl;
    f(x);
    cout << "x = " << x << endl;
}
```

- i. *What will get printed as a result of the first cout statement in the main function?* (Question type: “Program statement output” required, Human-system agreement rate: 100%)
- ii. *What will get printed as a result of the first cout statement in the f function?* (Question type: “Program statement output” required, Human-system agreement rate: 94.5%)

- iii. *What will get printed as a result of the second cout statement in the f function? (Question type: “Program statement output” required, Human-system agreement rate: 97.5%)*
- iv. *What will get printed as a result of the second cout statement in the main function? (Question type: “Program statement output” required, Human-system agreement rate: 97%)*

9. *What will be the output of the following C++ program:*

```
#include <iostream>
using namespace std;
void f(int* p) {
    cout << "*p = " << *p << endl;
    *p = 50;
}
int main() {
    int x = 40;
    cout << "x = " << x << endl;
    f(&x);
    cout << "x = " << x << endl;
}
```

- i. *What will get printed as a result of the first cout statement in the main function? (Question type: “Program statement output” required, Human-system agreement rate: 100%)*
- ii. *What will get printed as a result of the cout statement in the f function? (Question type: “Program statement output” required, Human-system agreement rate: 100%)*
- iii. *What will get printed as a result of the second cout statement in the main function? (Question type: “Program statement output” required, Human-system agreement rate: 100%)*

10. *How are the C++ classes and structures similar? (Question type: Compare, Human-system agreement rate: 92%)*

11. *What is the difference between private and public members of a class?* (Question type: Contrast, Human-system agreement rate: 75%)

5.3.2.4 Fourth OOP Test

Questions used in the fourth OOP test, along with the question type and human-system agreement rate for each question, are presented below:

1. *Consider the following piece of C++ code:*

```
#include <iostream>
using namespace std;
class A {
    int data;
    public:
    void f(int arg) { data = arg; }
    int g() { return data; }
};
class B : public A { };
int main() {
    B obj; obj.f(20);
    cout << obj.g() << endl;
}
```

- i. *How many data members are contained in class B?* (Question type: “Quantity” required, Human-system agreement rate: 100%)
- ii. *How many member functions are contained in class B?* (Question type: “Quantity” required, Human-system agreement rate: 100%)
- iii. *Explain the meaning of the following code fragment:*
- ```
class B : public A
```
- (Question type: Short explanation/description, Human-system agreement rate: 77.5%)
- iv. *What will be the output of the program?* (Question type: “Program statement output” required, Human-system agreement rate: 100%)

2. Complete the following sentences:
  - i. \_\_\_\_\_ is the process of creating new classes, called derived classes, from existing or base classes. (Question type: Sentence completion, Human-system agreement rate: 100%)
  - ii. To be accessed from a member function of the derived class, data or functions in the base class must be public or \_\_\_\_\_. (Question type: Sentence completion, Human-system agreement rate: 98%)
3. If a base class contains a member function `basefunc()`, and a derived class does not contain a function with this name, can an object of the derived class access `basefunc()`? (Question type: “True” or “false” question, Human-system agreement rate: 100%)
4. If a base class and a derived class each include a member function with the same name, which member function will be called by an object of the derived class? (Question type: Single term generation, Human-system agreement rate: 97.5%)
5. Consider the following piece of code:

```

class Animal{
 float weight;
 public:
 Animal(float);
};
Animal::Animal(float w){
 weight=w;
}
class Cat : public Animal{
 public:
 Cat(float);
};
Cat::Cat(float w) : Animal(w){ }

```

**X**

What does part X represent in the above code? (Question type: Single phrase generation, Human-system agreement rate: 97%)

6. State whether the following statements are true or false:
  - i. Adding a derived class to a base class requires fundamental changes to the base class. (Question type: “True” or “false” question, Human-system agreement rate: 100%)
  - ii. A class D can be derived from a class C, which is derived from a class B, which is derived from a class A. (Question type: “True” or “false” question, Human-system agreement rate: 100%)

### 5.3.2.5 Fifth OOP Test

Questions used in the fifth OOP test, along with the question type and human-system agreement rate for each question, are presented below:

1. In C++, what keyword tells the compiler that it should not perform early binding? (Question type: Single term generation, Human-system agreement rate: 97.5%)
2. Define (in a single sentence):
  - i. Binding (Question type: Definition, Human-system agreement rate: 100%)
  - ii. Early binding (Question type: Definition, Human-system agreement rate: 92.5%)
  - iii. Late binding (Question type: Definition, Human-system agreement rate: 88.5%)
3. What term is used to refer to the redefinition of a function in a derived class? (Question type: Single term generation, Human-system agreement rate: 100%)
4. Consider the following C++ capability (that can be achieved if certain conditions are met):

*“Completely different functions are executed by the same function call”.*

What is this C++ capability called? (Question type: Single term generation, Human-system agreement rate: 100%)
5. In C++, where do we place the virtual keyword to indicate that a function is virtual? (Question type: “Location” required, Human-system agreement rate: 100%)

6. *Data abstraction and inheritance are key features of Object-Oriented Programming (OOP). Name one more key feature of OOP. (Question type: Single term generation, Human-system agreement rate: 100%)*
7. *If there is a pointer p to objects of a base class, and it contains the address of an object of a derived class, and both classes contain a non-virtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the \_\_\_\_\_ class to be executed. (Question type: Sentence completion, Human-system agreement rate: 100%)*
8. *Consider the following piece of code:*

```
class Base{
 public:
 void show(){
 cout << "Base\n";
 }
};

class Derv1 : public Base{
 public:
 void show(){
 cout << "Derv1\n";
 }
};

class Derv2 : public Base{
 public:
 void show(){
 cout << "Derv2\n";
 }
};
```

*The function show() is defined in the Base, Derv1 and Derv2 classes. If we want to declare the show() function as virtual, which class's show() function*

*must be declared to be so?* (Question type: “Location” required, Human-system agreement rate: 97.5%)

### 5.3.2.6 Sixth OOP Test

Questions used in the sixth OOP test, along with the question type and human-system agreement rate for each question, are presented below:

1. *When do we set data member or member function of a class as “Public”?* (Question type: “Situation” or “context” required, Human-system agreement rate: 80.5%)
2. *Which C++ operator provides dynamic memory allocation? When this operator is used in an expression to create an object, what does that expression return? If the object is an array, then what does that expression return?* (Question type: Composite questions, Human-system agreement rate: 94.5%)
3. *Consider the following two C++ statements:*

```
int array1[20];
float array2[25];
```

*What is the similarity between the two C++ statements?* (Question type: Compare, Human-system agreement rate: 95.5%)

4. *Give an example of an arithmetic binary operator?* (Question type: “Example” required, Human-system agreement rate: 98%)
5. *Give an example of an arithmetic unary operator?* (Question type: “Example” required, Human-system agreement rate: 100%)
6. *Give an example of a relational operator?* (Question type: “Example” required, Human-system agreement rate: 100%)
7. *What is the similarity between classes and structures in C++?* (Question type: Compare, Human-system agreement rate: 94.5%)
8. *Consider the following piece of C++ code:*

```
class Person{
 public:
 Person() {
 age=1;
```

```

 weight=5;
 }
 ~Person() {}
 void setAge(int age) {
 this.age=age;
 }
 int getAge() {
 return age;
 }
private:
 int age;
 int weight;
};

int main() {
 Person people[5];
 int i;
 for(i=0;i<5;i++)
 people[i].setAge(2*i+1);
 return 0;
}

```

- i. *How many Person objects are there in the people array? (Question type: “Quantity” required, Human-system agreement rate: 100%)*
- ii. *After the execution of for loop in the main method, what will be the value of the age attribute in the Person object at index 0 of the people array? (Question type: “Numerical value” generation, Human-system agreement rate: 100%)*
- iii. *What will be the value of the age attribute in the Person object at index 4 of the people array (after the execution of for loop in the main method)? (Question type: “Numerical value” generation, Human-system agreement rate: 100%)*



9. *What is the purpose of delete operator? What is called when the delete operator is used for a C++ class object?* (Question type: Composite questions, Human-system agreement rate: 90.5%)
10. *When do we use brackets after the delete operator?* (Question type: “Situation” or “context” required, Human-system agreement rate: 100%)
11. *When do we use the keyword virtual in C++?* (Question type: “Situation” or “context” required, Human-system agreement rate: 100%)
12. *What is the similarity between a base class object and a derived class object?* (Question type: Compare, Human-system agreement rate: 94%)

### 5.3.2.7 Analysis of Evaluation Results

Table 4 (on the next page) summarizes the structure testing results for all the allowed short-answer question types. The table shows the number of questions used, the average answer length and the average human-system agreement rate for each question type.

In order to interpret data in Table 4, it is important to understand how the values presented have been calculated. The distinction between a question and a question type is significant. Multiple questions were used for a particular question type. For example, 5 questions were used for the “single term generation” question type. To calculate the average answer length (in words) for a question type, the author used the following formula:

$$y = \frac{\sum_{i=1}^q w_i}{q \times a}$$

where  $y$  = average answer length (in words),  $w_i$  = total number of words in all the answers for  $i$ th question,  $q$  = total number of questions used for the question type, and  $a$  = total number of answers per question.

**Table 4. Summary of the system's performance on all the allowed short-answer question types**

| S. No. | Question type                       | Number of questions used | Average answer length (words) | Average human-system agreement rate |
|--------|-------------------------------------|--------------------------|-------------------------------|-------------------------------------|
| 1      | "True" / "False" question           | 8                        | 1.2                           | 100%                                |
| 2      | Sentence completion                 | 6                        | 1.5                           | 99.67%                              |
| 3      | Single term generation              | 5                        | 1.9                           | 99%                                 |
| 4      | "Quantity" required                 | 7                        | 3.1                           | 99.42%                              |
| 5      | "Numerical value" generation        | 3                        | 2.3                           | 98.83%                              |
| 6      | "Location" required                 | 3                        | 3.6                           | 99.16%                              |
| 7      | "Program statement output" required | 12                       | 3.9                           | 98.79%                              |
| 8      | Single phrase generation            | 5                        | 7.3                           | 93.80%                              |
| 9      | "Example" required                  | 3                        | 5.6                           | 99.33%                              |
| 10     | List                                | 2                        | 7.1                           | 92%                                 |
| 11     | Short Explanation / Description     | 6                        | 11.6                          | 92.33%                              |
| 12     | "Situation" or "context" required   | 3                        | 11.2                          | 93.50%                              |
| 13     | Definition                          | 3                        | 10.8                          | 93.67%                              |
| 14     | Contrast                            | 5                        | 20.4                          | 84.10%                              |
| 15     | Compare                             | 4                        | 15.6                          | 94%                                 |
| 16     | Composite questions                 | 3                        | 14.9                          | 92%                                 |

The average human-system agreement rate for a question type is calculated using the following formula:

$$z = \frac{\sum_{i=1}^q r_i}{q}$$

where  $z$  = average human-system agreement rate,  $r_i$  = human-system agreement rate for the  $i^{\text{th}}$  question, and  $q$  = total number of questions used for the question type.

Table 4 demonstrates that if an OOP short-answer question test is carefully designed, high human-system agreement rates can be achieved. The average human-system agreement

rate tends to decrease as the complexity of the short-answer question type increases. This trend is not valid across all question types. For example, “compare” questions are deemed to be more complex than the “single phrase generation” questions by the author but the average human-system agreement rate is higher for the “compare” questions than that for the “single phrase generation” questions. “Contrast” questions have the lowest average human-system agreement rate while “true” / “false” questions have the highest average human-system agreement rate. Another important pattern that may be deduced from the data in the table is that (in general) as the average answer length increases the average human-system agreement rate decreases but again this is not true in every case. The “contrast” question type has the highest average answer length and the lowest average human-system agreement rate. But the “example required” question type has both higher average answer length and higher average human-system agreement rate than the “single term generation” question type.

The errors of IndusMarker were analyzed and they fall into two categories: *misses* and *false positives*. A miss occurs when a response gets lower marks than it deserves. A false positive occurs when the system assigns more marks to a response than it deserves. In the case of IndusMarker’s evaluation, the number of misses was much higher than the number of false positives. Around 69% of all the errors were misses while only 31% of the errors were false positives. The relatively higher ratio of misses is due to the fact that it is very difficult to anticipate all the possible paraphrases for an answer. If some correct possibility is omitted by the person specifying the required structure, then the occurrence of that possibility in students’ answers will lead to misses.

There are two reasons for the system’s false positives. The first occurs when a student does not know when to stop typing – beginning with a correct answer but going on to say something that is clearly wrong. It is impossible to predict all the wrong possibilities in advance. So the “required structure” specification normally contains the structure of correct possibilities only. The system assigns marks when it finds the correct possibility that it is looking for in student’s answer text. It does not normally search for wrong parts of the students’ answers. The second reason for false positives is that sometimes the student

happens to use the correct language – but that the language is used in such a manner that it does not, in fact, convey the correct concept.

If the human-system agreement rate for a question is not 100%, the “required structure” for that question may be modified by analyzing the structure and content of those students’ answers where there is discrepancy between human and system markings. Once the required structure for a question has been finalized, it is stored for future use.

**Table 5. Time taken to formulate and validate the required structures for each of the 6 OOP tests**

| <b>Test</b>     | <b>Time taken</b>  |
|-----------------|--------------------|
| First OOP test  | 1 hour 50 minutes  |
| Second OOP test | 2 hours 10 minutes |
| Third OOP test  | 2 hours 45 minutes |
| Fourth OOP test | 1 hour 35 minutes  |
| Fifth OOP test  | 1 hour 40 minutes  |
| Sixth OOP test  | 2 hours 35 minutes |

In order to be confident about the feasibility of the system, it is also important to consider whether the lecturers find it easy to learn QAML and to use the system. The lecturers were given a detailed presentation about QAML and how to use the system. IndusMarker’s user manual, containing detailed guidelines on how to use the system and how to specify the required structures, was also provided to the lecturers. The lecturers found QAML a simple and a sufficiently expressive language through which the required structures can be expressed conveniently. According to the lecturers’ comments, the system is easy to use and the QAML structure editor is quite helpful in the task of required structure specification. Table 5 shows the time taken by the lecturers to formulate and validate the required structures for the 6 OOP tests used in the evaluation. The lecturers found the time consumed quite reasonable and manageable given that these tests will be repeated many times and their automated marking will provide useful benefits to both students and teachers. Time taken to formulate and validate the required structures for a particular OOP test

depends upon the number and complexity of the questions appearing in that test. The next section describes the use of IndusMarker to conduct the OOP practice tests.

### **5.3.3 Using IndusMarker to Conduct the OOP Practice Tests**

The OOP tests were designed to be used as low-stake, practice tests. Since IndusMarker cannot guarantee a 100% human-system agreement rate, it can not be used for high-stake tests. The approach to using the system for practice tests is only effective if the same practice test is repeated many times. The lecturers at BU indicated that in most cases the curriculum of a course does not change for several years and therefore the same practice test may be used for many terms or semesters. When a practice test is taken for the first time, the students' answers are manually marked and these manually marked students' answers are used to develop and validate the required structures. The required structures are stored in the system's database and can be used to mark future practice tests. This is similar to the situation where a lecturer spends a considerable amount of time preparing a lecture presentation in the form of PowerPoint slides and then reuses the same lecture slides for several years. In this way, time spent preparing the slides when a course is taught for the first time is compensated for if the same course material is taught for several years and/or by many different lecturers. In fact, effort expended and time spent initially results in much greater time and effort being saved later. Similarly, if some time and effort is consumed making and validating the required structures for a practice test, then this results in much greater benefit later on if the same test is repeated. Since the marks obtained in these practice tests do not contribute to the final grade of students, and the main objective is to promote learning and provision of immediate (and accurate) feedback to both students and teachers, it is expected that students will not raise serious objections even if there is some lack of trust in the system's marking accuracy.

Once the required structures for questions appearing in the six OOP tests were finalized (i.e. when the process of required structure formulation, validation and correction had ended), the required structures were stored in the system database. The required structures were then used when these tests were later given to students studying the same course at BU Islamabad campus. The tests were conducted online in computer-based classrooms and students who took these tests obtained summative feedback on their

performance in the form of marks immediately after the test. Marks obtained in each question and total marks obtained in the test were provided to each student who took the test. If a student thinks that the marks given by the system are not accurate then the student has the option to view the correct model answer so that he or she can compare his or her own answer with the correct model answer. Moreover, the system can highlight the parts of the student's answer text that have been matched with the required structure. It can also indicate to the student the regions (of the QAML regions specification) that have been found and those that have not been found in the student's answer text. In this way, the student can have a better idea of how his or her marks for a particular answer have been calculated. He or she can also better understand his or her mistakes. Figure 9 (on the next page) depicts an IndusMarker's GUI screen for carrying out student's answer text analysis. The student's answer text is displayed in a text area in the upper right side of the screen. The list of required "regions" for the answer is given in the lower part of the GUI screen. The screen enables its user (i.e. the student) to check whether a particular required "region" has been matched in the student's answer text or not. A "region" may be "fully matched", "partially matched" or "not matched" at all in the student's answer text. In order to check whether a particular "region" is matching or not, the student selects a "region" and presses the "Match Region" button. The sentence that matches the most, with the selected "region", is highlighted. Other important information such as "Matching Result", "Marks Obtained" and "Total Marks" can also be viewed on the screen.

Lecturers can also view the results of the test if they log-in to the system. Lecturers can then revise those topics where the overall students' performance is poor, or give additional tutorials to those students who are performing poorly. Students can also get an idea of their overall understanding of the course content and can increase their learning effort if their performance is poor. Both students and lecturers at the BU Islamabad campus found the system quite useful and its performance satisfactory. The system can also be viewed as a tool that promotes "deep learning" [45], [46]. In such learning, both lecturer and students actively participate in students' learning. The lecturer needs to obtain feedback on their teaching performance from students, and students need feedback from the lecturer on their learning performance. The feedback must reach the students as quickly as possible in

order to affect their learning, that is, to promote deep learning. This objective can be achieved through proper use of the system.

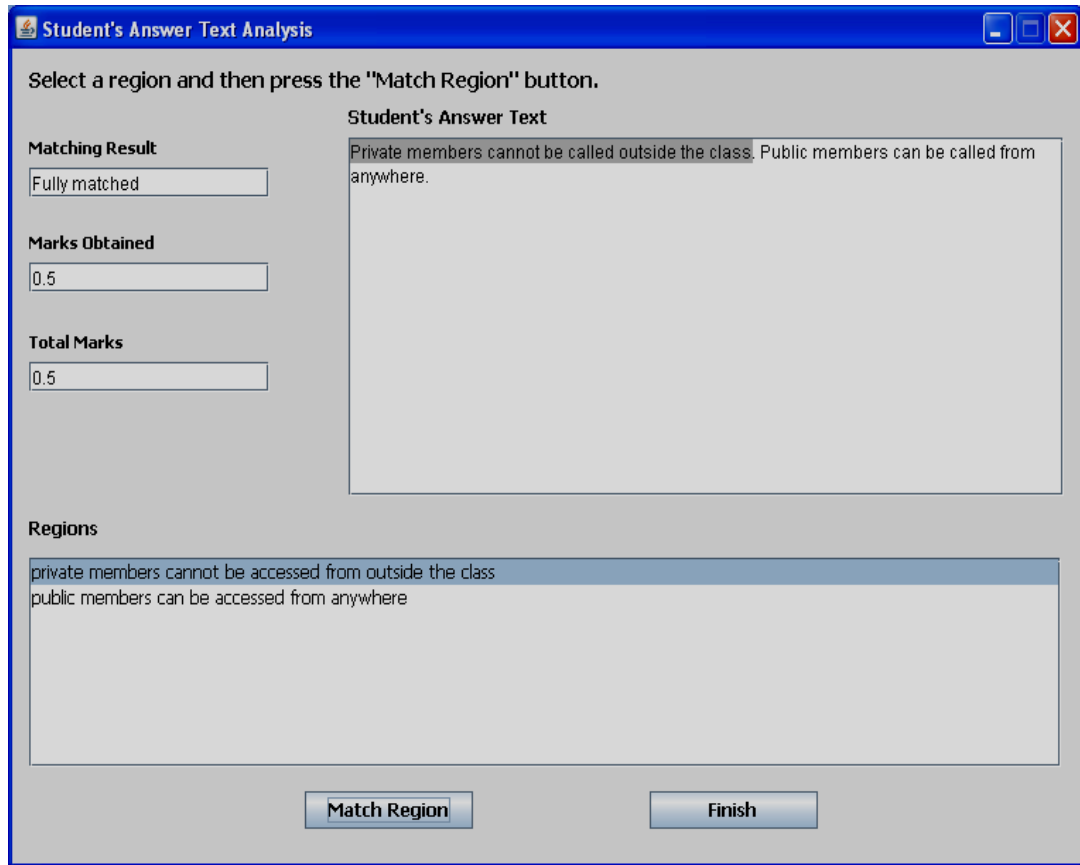


Figure 9. An IndusMarker's GUI screen for carrying out student's answer text analysis

#### ***5.4 IndusMarker's Evaluation at the City School***

IndusMarker's evaluation was also performed at the City School, Pakistan [47]. The City School is one of the largest private English medium school systems in Pakistan and was established in 1978. It now has more than 150 branches in 42 cities across Pakistan. More than 50,000 students are enrolled at the City School. Most of its students opt to take the international GCE O-level and A-level examinations at the end of grade 11 and 13 of their schooling respectively. Education at the City School is divided into four levels: junior (grades one to six), prep (grades seven to nine), O-level (grades ten and eleven) and A-level (grades twelve and thirteen). A grade corresponds to a year of schooling. A student gets promoted from a particular grade to the next by securing at least a minimum percentage of marks.

IndusMarker was evaluated using seventh grade students' answers for "science" questions. Science is taught as a subject from grade one at the City School. By the time students reach grade seven, they already possess some basic knowledge about science. The syllabus of the "science" subject for grade seven consists of a number of topics such as "acids and alkalis", "chemical reactions", "energy", "forces", "the human body" etc.

#### **5.4.1 Allowed Short-Answer Question Types for Science Tests**

Most of the types of short-answer questions used in OOP tests are reused in science tests. But some changes to the list of allowed question types were made. It was realized that including "True/False" questions again in science tests will most likely produce 100% human-system agreement rate. The analysis of such results that are easily predictable in advance will not give any useful insight into the performance of the system and therefore questions of this type have not been included. The "program statement output" question type is not applicable to science tests and therefore this question type has also been removed from the list of allowed short-answer question types for science tests. The "single phrase generation" and the "'situation' or 'context' required" question types have also been excluded from the list of allowed short-answer question types. The reason for excluding the "single phrase generation" question type is that questions of this type can easily be classified as belonging to some other question type such as "short explanation/description". The reason for excluding the "'situation' or 'context' required" question type is that questions of this type appear less frequently in science tests and when they do appear they may also be classified as belonging to the "short explanation / description" question type.

The "short explanation / description" question type is an interesting question type as many questions of varying complexity and nature can be crafted of this type. Two sub-types of "short explanation / description" have been identified: (1) "'reason' or 'justification' required" and (2) "'way of doing something' required". These two subtypes have been included to evaluate and analyze the system's performance on specialized forms of "short explanation / description". The "short explanation / description" questions that do not fall in these specialized categories are categorized as belonging to the generic category of "short explanation / description". Another question type that has been added to the list of allowed short-answer question types is "'ordering / rearrangement' required". This is a specialized



form of the “list” question type. It requires students to rearrange a list of given items or provide a list of items in a specified order. Examples of “‘ordering / rearrangement’ required” questions are:

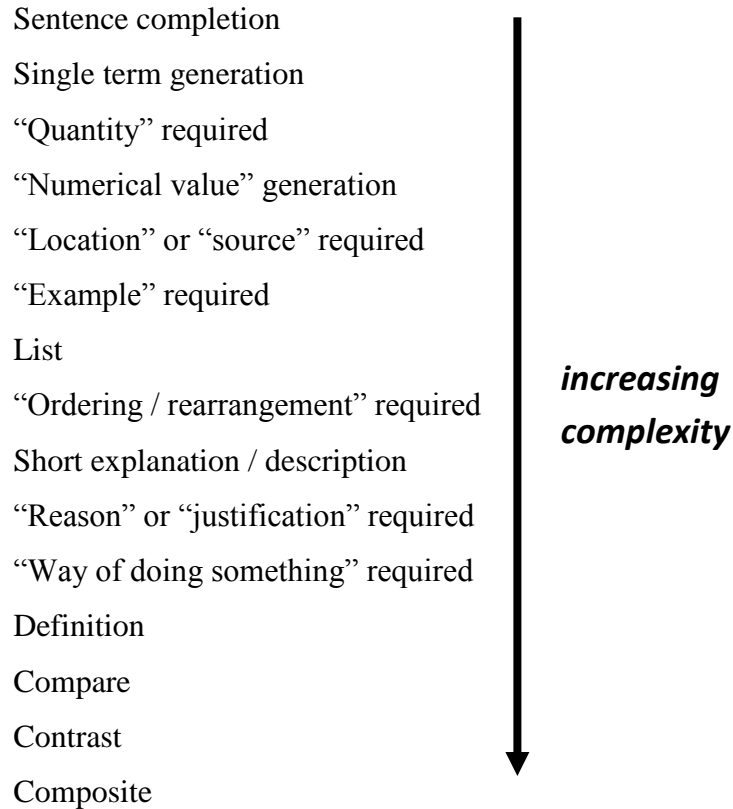
1. Rearrange the following list of planets, so that the planet closer to sun appears before the planet further away from the sun: Saturn, Jupiter, Mercury, Venus, Mars, Earth.
2. Name the colors of the spectrum, in order.

The “‘reason’ or ‘justification’ required” questions normally start with “why” while the “‘way of doing something’ required” questions normally start with “how” but it must be remembered that not every question starting with “how” belongs to this question type. Table 6 gives example questions for each of the three question types so that the difference between the three question types becomes clearer.

**Table 6. Example questions for the three question types**

| Question type                              | Example questions                                                                                                                                                                                         |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| “Short explanation / description” required | <ol style="list-style-type: none"> <li>1. What is the job of red blood cells?</li> <li>2. What happens to the particles in a solid when it dissolves in water?</li> </ol>                                 |
| “Reason” or “justification” required       | <ol style="list-style-type: none"> <li>1. Why does an astronaut weigh less on the moon than on the Earth?</li> <li>2. Why some people are not convinced that viruses are really living things?</li> </ol> |
| “Way of doing something” required          | <ol style="list-style-type: none"> <li>1. How can we increase the strength of an electromagnet?</li> <li>2. How can infectious diseases spread?</li> </ol>                                                |

The following is the list of allowed short-answer question types for science tests (the question types are arranged in ascending order of complexity):



It must be remembered that the order of complexity presented above is what the author perceived before the start of evaluation. As shown later on in Section 5.4, the order of complexity perceived does not necessarily always have a direct correlation with the human-system agreement rates achieved as a result of evaluation.

Like the IndusMarker’s evaluation at Bahria University, the evaluation at the City school was also divided in to two phases: (1) the science tests creation and the required structures formulation and validation at one branch, and (2) use of the stored science tests and the associated required structures to conduct practice tests at two other branches of the City School.

#### **5.4.2 The Science Tests Creation and the Required Structures Formulation and Validation Phase**

The first phase of evaluation was carried out at a large branch of the City School in Karachi called “PAF chapter”. An important reason for choosing the “PAF chapter” was that a number of computer-based classrooms (i.e. classrooms where each student is provided

with a computer) were available. The computer-based classrooms were equipped with new computers connected together to form a Local Area Network (LAN). This facility was not available at many other branches of the City School.

A typical cohort of students (i.e. students of the same grade) at the “PAF chapter” branch consists of 90 to 120 students. Students of the same grade are then further divided in to “sections”. At the time the IndusMarker software was evaluated at the “PAF chapter”, there were 109 students registered for the seventh grade. The students of the seventh grade were divided in to four sections. Each section had around 25 to 28 students. Students of each section study in a separate classroom. If students need to use a computer, they are taken to one of the computer-based classrooms that are available.

Three teachers took part in the evaluation process. Each one was assigned a specific task. One of them was assigned the task of setting up science tests containing questions of the allowed short-answer question types, another teacher was responsible for manually marking students’ answers and the third teacher was given the task of required structure formulation and validation. All three teachers were experienced science teachers as they have been teaching science at the City School for more than ten years. They have taught science to students of various grades including those of grade seven. Thus, they had a good knowledge of the science subject. A presentation was given to the three science teachers about the IndusMarker system and the way it is used. A thorough set of guidelines were provided about how to formulate and validate the required structures in QAML using the IndusMarker software.

Three science tests were designed each containing fifteen questions of the allowed short-answer question types. Overall in the three science tests, three questions were used for each question type. This is unlike OOP tests where the number of questions used for each question type was different. The test questions were on a number of science topics. Table 7 (on the next page) shows the topics covered in each of the three science tests.

The academic year at the City School starts in August and ends in May the following year. Each academic year is divided in to two semesters: the fall semester and the spring semester. The fall semester starts in August and ends in December. The spring semester starts in January and ends in May. IndusMarker’s evaluation was performed in the fall

semester of the academic year 2009-10. The three science tests were conducted in the months of October and November. It was ensured that the topics covered in each science test had already been taught to students by the time the test was conducted.

**Table 7. Topics covered in each of the three science tests**

| <b>Test</b> | <b>Topics covered</b>                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| First test  | <ul style="list-style-type: none"> <li>• Acids and alkalis</li> <li>• The human body</li> </ul>                                                               |
| Second test | <ul style="list-style-type: none"> <li>• Variety of life</li> <li>• Magnetism and electricity</li> <li>• Earth and space</li> <li>• The human body</li> </ul> |
| Third test  | <ul style="list-style-type: none"> <li>• Chemical reactions</li> <li>• Energy</li> <li>• Forces</li> <li>• Sight and sound</li> <li>• Materials</li> </ul>    |

IndusMarker system was deployed at the computer-based classrooms of the “PAF chapter” and the students’ answers were collected and stored by the system. The students’ attendance in the three tests varied slightly from 96 to 103. Students’ answers for the test questions were first manually marked by the teacher responsible for the task. The manual marking by the teacher was stored in the system. As in the OOP tests, the students’ answer data set for each question was divided in to two parts: 26 to 33 students’ answers were kept for the required structure formulation task and the remaining 70 students’ answers were kept for the required structure validation task. The number of students’ answers used for the required structure formulation task varies due to the different number of students present on each of the three test days. It was decided that the number of students’ answers for the required structure validation task should be the same as this will ease the task of result analysis later on.

To define the required structure for a question, the teacher responsible for the task analyzes the structure and content of the model answer and the manually marked students' answers that were set aside for this purpose. The marks awarded to students' answers were also considered while creating the required structures as they provided a guideline for the marking scheme used. Once the required structure for a question had been developed using the system's structure editor, it was tested using the 70 manually marked students' answers that were set aside for the required structure validation.

Questions used in the three science tests are presented in the following three subsections (i.e. from Section 5.4.2.1 to Section 5.4.2.3) along with other important information such as type and topic of each question and the structure testing result for each question. The structure testing results are presented as human-system agreement rates. Analysis of structure testing and other evaluation results is presented in Section 5.4.2.4.

#### **5.4.2.1 First Science Test**

Questions used in the first science test, along with other associated information, are presented below:

1. *What are indicators used for?* (Question type: Short explanation / description, Topic: Acids and alkalis, Human-system agreement rate: 92.85%)
2. *Why is universal indicator more useful than other indicators?* (Question type: "Reason" or "justification" required, Topic: Acids and alkalis, Human-system agreement rate: 87.14%)
3. *How do indigestion tablets work?* (Question type: "Way of doing something" required, Topic: Acids and alkalis, Human-system agreement rate: 84.28%)
4. *Give 3 ways in which your skeleton helps you.* (Question type: List, Topic: The human body, Human-system agreement rate: 92.85%)
5. *What is the chemical found in your skin that protects you from ultra-violet light?* (Question type: Single term generation, Topic: The human body, Human-system agreement rate: 100%)

6. *What is an element?* (Question type: Definition, Topic: Elements, Human-system agreement rate: 87.14%)
7. *Why is toothpaste usually slightly alkaline?* (Question type: “Reason” or “justification” required, Topic: Acids and alkalis, Human-system agreement rate: 91.42%)
8. \_\_\_\_\_ *attach bones to other bones.* (Question type: Sentence completion, Topic: The human body, Human-system agreement rate: 100%)
9. *When a muscle gets shorter and fatter, we say it \_\_\_\_\_.* (Question type: Sentence completion, Topic: The human body, Human-system agreement rate: 100%)
10. *Rewrite the following list of liquids so that the lower pH liquid comes before the higher pH liquid: lemon juice, pure water, sodium hydroxide solution, milk of magnesia ( $Mg(OH)_2$ ) solution, hydrochloric acid.* (Question type: “Ordering / rearrangement” required, Topic: Acids and alkalis, Human-system agreement rate: 94.28%)
11. *What are the four types of teeth?* (Question type: List, Topic: The human body, Human-system agreement rate: 94.28%)
12. *Where is acid added when your body is digesting food?* (Question type: “Location” or “source” required, Topic: The human body, Human-system agreement rate: 97.14%)
13. *Where is digestion completed and food passed in to your blood?* (Question type: “Location” or “source” required, Topic: The human body, Human-system agreement rate: 98.57%)
14. *Why are sleeping bags designed to trap air?* (Question type: “Reason” or “justification” required, Topic: The human body, Human-system agreement rate: 92.85%)

15. *How could you measure growth?* (Question type: “Way of doing something” required, Topic: The human body, Human-system agreement rate: 94.28%)

#### 5.4.2.2 Second Science Test

Questions used in the second science test, along with other associated information, are presented below:

1. *Sugar is broken down inside animals and plants to produce energy. What is this process called?* (Question type: Single term generation, Topic: Variety of life, Human-system agreement rate: 100%)
2. *Where do plants get the energy they need for food production?* (Question type: “Location” or “source” required, Topic: Variety of life, Human-system agreement rate: 97.14%)
3. *What is the main difference between a vertebrate and an invertebrate?* (Question type: Contrast, Topic: Variety of life, Human-system agreement rate: 91.42%)
4. *Plants make their own food in a process called \_\_\_\_\_.* (Question type: Sentence completion, Topic: Variety of life, Human-system agreement rate: 98.57%)
5. *What is the common feature of all vertebrates?* (Question type: Compare, Topic: Variety of life, Human-system agreement rate: 94.28%)
6. *What happens to a thin piece of wire when electricity is passed through it?* (Question type: Short explanation / description, Topic: Magnetism and electricity, Human-system agreement rate: 95.71%)
7. *What do we use to measure the size of an electric current?* (Question type: Single term generation, Topic: Magnetism and electricity, Human-system agreement rate: 100%)
8. *What do we call the safety device used in plugs? How does it work?* (Question type: Composite, Topic: Magnetism and electricity, Human-system agreement rate: 81.42%)

9. *What is the similarity between a bar magnet and an electromagnet?* (Question type: Compare, Topic: Magnetism and electricity, Human-system agreement rate: 97.14%)
10. *What happens in an eclipse of the sun?* (Question type: Short explanation / description, Topic: Earth and space, Human-system agreement rate: 87.14%)
11. *How many planets are there in the solar system?* (Question type: “Quantity” required, Topic: Earth and space, Human-system agreement rate: 100%)
12. *Give an example of a smaller object type in our solar system (i.e. objects apart from the sun, planets and satellites that go around planets).* (Question type: “Example” required, Topic: Earth and space, Human-system agreement rate: 98.57%)
13. *How many eggs does a woman usually release each month?* (Question type: “Quantity” required, Topic: The human body, Human-system agreement rate: 100%)
14. *Which parts of animal and plant cells common in both?* (Question type: Compare, Topic: Variety of life, Human-system agreement rate: 94.28%)
15. *What is a herbivore?* (Question type: Definition, Topic: Variety of life, Human-system agreement rate: 88.57%)

#### **5.4.2.3 Third Science Test**

Questions used in the third science test, along with other associated information, are presented below:

1. *What is a catalyst?* (Question type: Definition, Topic: Chemical reactions, Human-system agreement rate: 90%)
2. *What 3 things make up the fire triangle?* (Question type: List, Topic: Chemical reactions, Human-system agreement rate: 94.28%)
3. *What is the difference between a physical and a chemical change?* (Question type: Contrast, Topic: Chemical reactions, Human-system agreement rate: 85.71%)



4. *Which metal is used to galvanize iron? Why is it a good way to protect iron?* (Question type: Composite, Topic: Chemical reactions, Human-system agreement rate: 78.57%)
5. *What is the difference between a renewable and a non-renewable source of energy?* (Question type: Contrast, Topic: Energy, Human-system agreement rate: 87.14%)
6. *Give an example of a renewable energy source.* (Question type: “Example” required, Topic: Energy, Human-system agreement rate: 95.71%)
7. *The battery in a torch stores 100 joules of energy and 30 joules are out as light. How much energy is wasted as heat?* (Question type: “Numerical value” generation, Topic: Energy, Human-system agreement rate: 100%)
8. *What does kJ stand for? How many joules are there in 1 kJ?* (Question type: Composite, Topic: Energy, Human-system agreement rate: 97.14%)
9. *You push something with a force of 5 N and the force of friction is 2 N. What is the resultant force?* (Question type: “Numerical value” generation, Topic: Forces, Human-system agreement rate: 100%)
10. *An athlete runs 50 meters in 5 seconds. What is the athlete’s average speed, in meters per second?* (Question type: “Numerical value” generation, Topic: Forces, Human-system agreement rate: 97.14%)
11. *How many small bones are there in the ear?* (Question type: “Quantity” required, Topic: Sight and sound, Human-system agreement rate: 100%)
12. *How can you make the image in a pin-hole camera brighter?* (Question type: “Way of doing something” required, Topic: Sight and sound, Human-system agreement rate: 94.28%)
13. *Name the colors of the spectrum, in order.* (Question type: “Ordering / rearrangement” required, Topic: Sight and sound, Human-system agreement rate: 92.85%)

14. Give an example of a fossil fuel. (Question type: “Example” required, Topic: Materials, Human-system agreement rate: 97.14%)

15. Rewrite the following list of materials so that solids come first, then liquids and then gases: oxygen, oil, wood, water, iron, air. (Question type: “Ordering / rearrangement” required, Topic: Materials, Human-system agreement rate: 92.85%)

#### **5.4.2.4 Analysis of Structure Testing and other Evaluation Results**

Table 8 (on the next page) summarizes the structure testing results and shows how the performance of IndusMarker varies with the different allowed short-answer question types. The performance of IndusMarker on a specific short-answer question type is indicated through average human-system agreement rate for that question type. Higher average human-system agreement rate means better IndusMarker performance. Another piece of information presented in the table is the average answer length (in words) for each question type. The average answer length and average human-system agreement rate for the science test question types are calculated in the same way as these values were calculated for the OOP tests’ question types.

The table demonstrates that IndusMarker was able to mark the allowed question types with high average human-system agreement rates. The average human-system agreement rate tends to decrease as the complexity of the question types increases. This is of course not true in every case as instances where a more complex question type has a higher average human-system agreement rate can be easily identified. But this trend is more uniform than the one we had in OOP tests i.e. deviations from the trend are less frequent and the size of deviations is also smaller than the ones we had in OOP tests. Figure 10 (on page 132) is the graph that depicts this trend. The graph has been drawn so that the trend may be visualized. The greatest deviation from the trend is the average human-system agreement rate for the “compare” questions. The “compare” questions were perceived to be more complex than most of the other question types used. An important reason for this belief was that “compare” questions are classified as higher-order questions in Bloom’s taxonomy [48]. But the average human-system agreement rate is higher than many other question types which were perceived to be simpler by the author. This may be due to careful design of

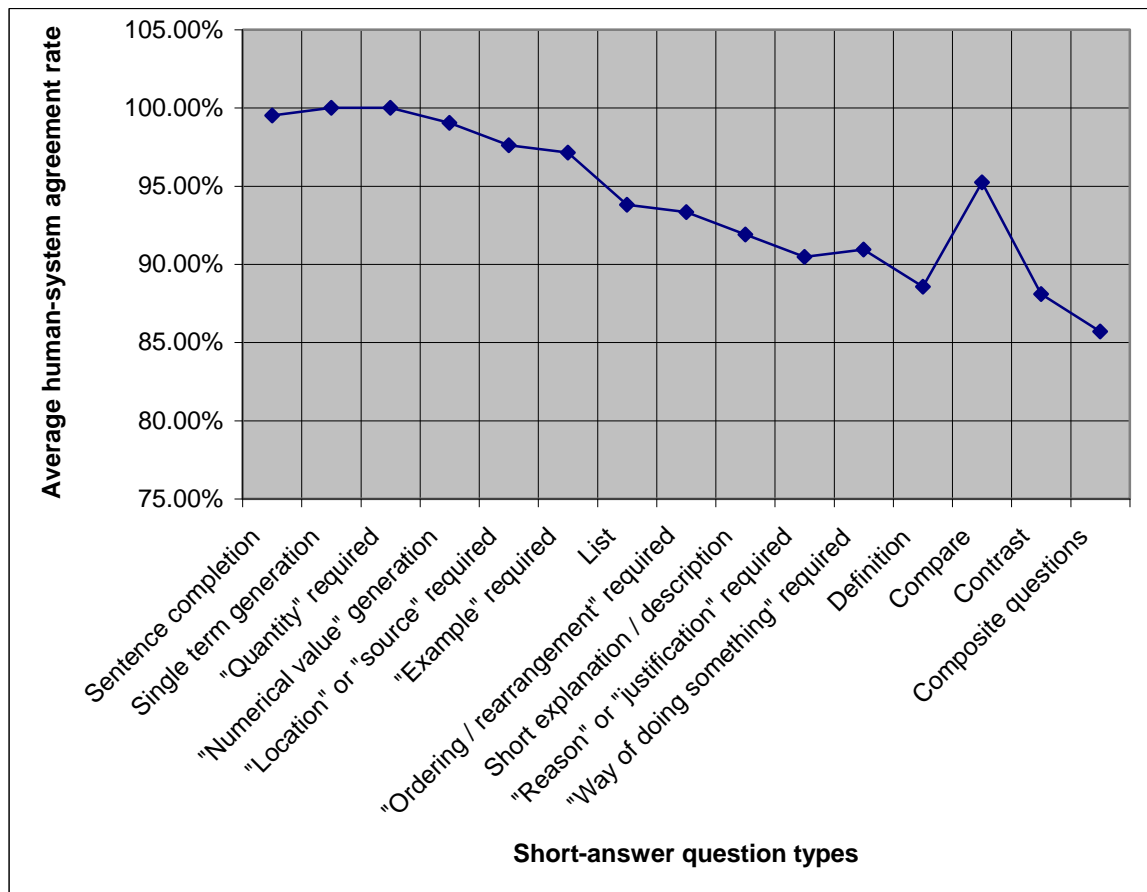
“compare” questions that restricted the scope of questions and therefore, correct answers can be easily distinguished from incorrect answers by IndusMarker.

**Table 8. Summary of the system’s performance on all the allowed short-answer question types for science tests**

| S. No. | Question type                        | Average answer length (words) | Average human-system agreement rate |
|--------|--------------------------------------|-------------------------------|-------------------------------------|
| 1      | Sentence completion                  | 1.4                           | 99.52%                              |
| 2      | Single term generation               | 2.2                           | 100%                                |
| 3      | "Quantity" required                  | 2.9                           | 100%                                |
| 4      | "Numerical value" generation         | 2.3                           | 99.04%                              |
| 5      | "Location" or "source" required      | 3.4                           | 97.61%                              |
| 6      | "Example" required                   | 2.8                           | 97.14%                              |
| 7      | List                                 | 7.3                           | 93.80%                              |
| 8      | "Ordering / rearrangement" required  | 10.5                          | 93.33%                              |
| 9      | Short explanation / description      | 8.7                           | 91.90%                              |
| 10     | "Reason" or "justification" required | 10.1                          | 90.47%                              |
| 11     | "Way of doing something" required    | 9.6                           | 90.95%                              |
| 12     | Definition                           | 13.6                          | 88.57%                              |
| 13     | Compare                              | 10.7                          | 95.23%                              |
| 14     | Contrast                             | 19.6                          | 88.09%                              |
| 15     | Composite                            | 15.9                          | 85.71%                              |

The general relationship between average answer length and average human-system agreement rate that can be deduced from the data presented in Table 8 is that as the average answer length increases the average human-system agreement rate decreases. This general relationship also existed between the two properties when IndusMarker was evaluated using the OOP tests. It is easy to notice deviations from this general pattern but still the relationship provides a good guideline about the performance of IndusMarker i.e. IndusMarker performs less accurately with longer answers. There is nothing in the design of

IndusMarker system that demands constraints on the size of student's answer. **It is important to note that, in principle, IndusMarker can process essay type answers as well.** The reason for this assertion is that IndusMarker is basically a text structure analysis tool and therefore can be applied to mark essay type answers as well (as long as essay type questions test factual knowledge). The author believes that such a use of IndusMarker is not feasible because too much time will be consumed in required structure formulation and the accuracy of IndusMarker will also likely degrade to an unacceptable level.



**Figure 10. Graph depicting how average human-system agreement rate varies with the allowed short-answer question types for the science tests. The complexity of the question types increases from left to right<sup>10</sup>**

<sup>10</sup> The order of the question types is based on the amount of processing that is expected to be required for students' answers of a particular question type.

Like the IndusMarker’s evaluation at Bahria University, the errors of IndusMarker in science tests at the City School were also analyzed and categorized as misses or false positives. 66% of the errors were categorized as misses and 34% of the errors were categorized as false positives. The reasons for the occurrence of the two types of errors are the same as those given for errors occurred during system evaluation at Bahria University.

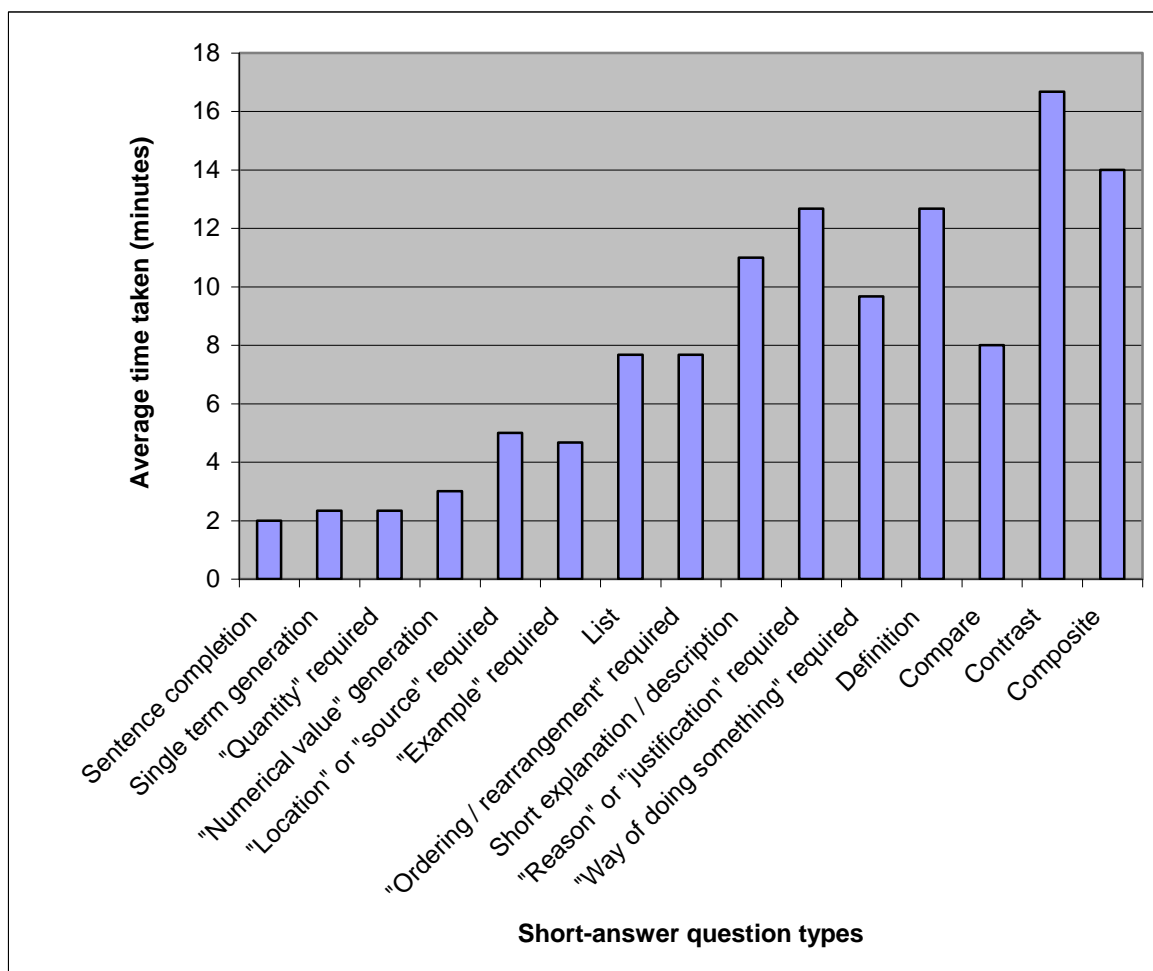
It is very important to consider how easy or difficult it was to formulate required structures in QAML using the IndusMarker’s structure editor. A good way of doing this is to measure time taken to formulate the required structure. More time taken is considered to mean that the teacher found it difficult to develop the required structure. Less time taken is considered to mean that the teacher found it easy to develop the required structure. Unlike IndusMarker’s evaluation using OOP tests, time taken to formulate the required structure for each and every question was measured rather than the total time taken for the entire test. This enabled the author to analyze how the difficulty level of required structure formulation varies with the different short-answer question types. Average time taken to formulate the required structures for each question type is calculated using the following formula:

$$a = \frac{\sum_{i=1}^n t_i}{n}$$

where  $a$  = average time taken,  $t_i$  = time taken to formulate the required structure for  $i^{\text{th}}$  question, and  $n$  = total number of questions used for the question type.

Figure 11 (on the next page) depicts how the average time taken varied with different allowed short-answer question types during IndusMarker’s evaluation using science tests. As the complexity of the question types increases, the average time taken also tends to increase. If the graphs of Figure 10 (on page 132) and Figure 11 are analyzed together, another important finding is that as the time taken to formulate the required structures increases, the average human-system agreement rate decreases. In other words, IndusMarker becomes increasingly less feasible as the complexity and length of students’ answer increases. Therefore, due to this limitation of IndusMarker, the author re-emphasizes that the use of IndusMarker should be limited to the allowed short-answer question types.

The teacher (responsible for the required structure formulation task) found the average time consumed for all the question types quite manageable. Required structure formulation for questions belonging to the “contrast” question type took the greatest amount of time. Other question types, such as “composite” and “definition”, were also considerably much more time consuming to deal with than simpler question types such as “sentence completion” and “single term generation”. The teacher found the QAML structure editor of IndusMarker quite helpful in carrying out the task. He also suggested that there is room for improvement in the design of the structure editor and improving the structure editor may make the whole process of developing required structures faster.



**Figure 11. Graph depicting how the average time taken to formulate the required structures varies with the different short-answer question types**

### **5.4.3 Using IndusMarker to Conduct the Science Practice Tests**

The required structures are adjusted based on the structure testing results. Once the required structures are finalized, they are stored in the system database for later use. The second phase of evaluation involves the use of stored required structures to conduct practice tests. These practice tests were conducted at two other branches of the City School: (1) “Darakhshan campus”, and (2) “PECHS Prep Boys-A”. Both the branches are smaller in size than the “PAF chapter” but follow exactly the same system of education as the “PAF chapter”. The syllabus for the seventh grade science subject is same at the three branches of the City School. Therefore the science tests and the associated required structures prepared at the “PAF chapter” can be effectively used as practice tests at the “Darakhshan campus” and “PECHS Prep Boys-A”.

At the time the practice tests were conducted, the number of students enrolled in the seventh grade at “Darakhshan campus” was 63 and at “PECHS Prep Boys-A” was 58. Both the branches had computer-based classrooms with computers good enough to run the IndusMarker system. When the practice tests were conducted at the two branches, both the students and teachers were satisfied with their experience of IndusMarker. According to the comments made by students and teachers, the system provided a useful means to get an idea of students’ preparedness for the final exam. Students appreciated the immediate test results generated by the system. They do not have to wait very long to get the test results. Students informed the author that usually teachers do not mark the practice tests that they take because of a lack of time. In such cases, practice tests have little benefit. The use of IndusMarker to conduct practice tests eliminates this problem as immediate feedback is provided with no effort from the teachers. Practice tests and their results also gave students opportunity to identify their weaknesses and use this information to make better study plans for the final exam.

### ***5.5 Comparison and Lessons Learnt from the Two Case Studies***

Key features of the two case studies are summarized in Table 9 below:

**Table 9. Key Features of the Two Case Studies**

|                                                          | <b>Evaluation at Bahria University</b>                                                                                                                | <b>Evaluation at the City School</b>                                                                                              |
|----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>Course Used</b>                                       | Object-Oriented Programming                                                                                                                           | 7 <sup>th</sup> grade science                                                                                                     |
| <b>Number of Questions Used</b>                          | 78                                                                                                                                                    | 45                                                                                                                                |
| <b>Data Set Size for “Answer Structure ” Formulation</b> | 25 students’ answers                                                                                                                                  | 26 to 33 students’ answers                                                                                                        |
| <b>Data Set Size for “Answer Structure” Validation</b>   | 200 students’ answers                                                                                                                                 | 70 students’ answers                                                                                                              |
| <b>People Involved</b>                                   | 2 lecturers (responsible for tests creation and “answer structure” formulation and validation)+6 teaching assistants (responsible for manual marking) | 3 teachers (1 teacher for test creation+1 teacher for manual marking+1 teacher for “answer structure” formulation and validation) |
| <b>Number of Allowed Short-Answer Question Types</b>     | 16                                                                                                                                                    | 15                                                                                                                                |
| <b>Number of Tests Created and Used</b>                  | 6                                                                                                                                                     | 3                                                                                                                                 |

Results obtained from the two case studies indicate similar trends. In both the case studies, high average human-system agreement rate was achieved for all the allowed short-answer question types. Overall, the average human-system agreement rate decreases as the complexity of the question type increases or as the average answer length increases (but there are some deviations from this general trend). Time spent in “required structures”



formulation and validation was also manageable. In both the case studies, the same evaluation process was followed. The following are the main steps of that process:

1. Test creation keeping in mind the allowed short-answer question types.
2. The test is conducted for the first time.
3. Manual marking of the test.
4. For each question:
  - i. “Answer structure” formulation using training data.
  - ii. “Answer structure” validation using unseen test data.
  - iii. (If necessary) “Answer structure” correction and validation of the corrected “answer structure” using seen test data.
5. Once “answer structures” are ready, the test may be repeated wherever and whenever the same course material is taught. The same course material may be taught during different semesters/years or at different locations (i.e. different branches/campuses).

The above process is analogous to the process of software development and software use. Software developers spend considerable time designing, writing and testing a computer program. But once the program is ready to be used, it can be used any number of times to perform the task for which it is designed. So, time and resources spent earlier on results in much greater time and resource saving later on. Another important point here is that the people using the program need not understand how the program was developed; they should only know how to use it. The same is true for “answer structure” development and use (in the case of IndusMarker). Teachers/lecturers using the “answer structures” do not need to understand how they are developed. Therefore, only those lecturers/teachers are provided the training for “answer structures” development who are responsible for that task. So, only 2 lecturers were provided this training at Bahria University and in the case of the City School only 1 teacher was trained. Other lecturers/teachers, who were just using “answer structures” to conduct practice tests, did not need the training for “answer structures” development. The whole process of teachers’ training, therefore, becomes manageable because very few teachers need to be trained for “answer structures” development.

## 6. Discussion and Conclusion

In the last chapter, IndusMarker's evaluation was presented. There are four important purposes of this chapter: (1) analyze the impact of IndusMarker on students' performance in the final exam (Section 6.1), (2) analyze IndusMarker's performance without the "linguistic features analyzer" (Section 6.2), (3) present a comparison of IndusMarker with other similar systems (Section 6.3), and (4) conclude the thesis by presenting a summary of the contributions made as a result of the author's research during this project, and also suggest some directions for future research (Section 6.4).

### *6.1 Impact of IndusMarker on Students' Performance*

From the beginning, IndusMarker was intended to be used to conduct short-answer practice tests. After completion of IndusMarker's design and implementation and also the subsequent completion of the required structures formulation and validation phase, IndusMarker was used to conduct practice tests at Bahria University (BU) Islamabad campus and the City School's "Darakhshan campus" and "PECHS Prep Boys-A" branches. It was stated in Section 5.3.3 and Section 5.4.3 that IndusMarker provided a useful means to both students and teachers to get an idea of students' preparedness for the final exam. This in turn helped both students and teachers to adjust their study or teaching plans respectively. So, what impact did IndusMarker's use actually had on students' performance in the final exams conducted at the two institutions? To answer this question, average assessment results for the courses used to validate the IndusMarker tool were obtained<sup>11</sup>. The data collected included students' results for the term when IndusMarker was used and also for the previous three terms when IndusMarker was not used to conduct practice tests. The purpose of this students' results collection is to find out whether or not the use of IndusMarker leads to improved students' performance in the final exam. If the average of students' marks is significantly greater for the term when IndusMarker was used compared with the other three terms, then IndusMarker's use can be considered to have improved students' performance in the final exam. It may be argued that students who took the course during the term when IndusMarker was used may be overall better students than the students of the previous terms.

---

<sup>11</sup> Permission to obtain students' final exam results was taken from both Bahria University and the City School.

To counter this argument, it must be realized that data is collected from three different locations (i.e. BU Islamabad campus, the City School’s “Darakhshan campus” branch and the City School’s “PECHS Prep Boys-A” branch) and students studying at each of these locations are different. Impact of IndusMarker’s usage on students’ performance in each of the three cases is analyzed separately and if the impact is positive in all three cases then the trend can be deemed to be generic.

Once the “required structures formulation, validation and storage” phase is completed for all questions of a test, the test is ready to be used as a practice test. The “required structures formulation, validation and storage” phase was completed for questions in the 6 Object-Oriented Programming (OOP) tests at the Karachi campus of BU. An OOP test was used as a practice test at the Islamabad campus soon after it was ready to be used as a practice test due to the completion of the “required structures formulation, validation and storage” phase at the Karachi campus. Table 10 gives average marks scored by students in the OOP course’s final exam at the Islamabad campus. The average marks of students are presented for the last four terms, i.e. the term when IndusMarker was used to conduct practice tests and the three terms before that term.

**Table 10. Average marks of students in the final OOP exams taken during the last four terms of the Bahria University’s Islamabad campus**

|                                                                                                        | <b>T<sub>1</sub></b> | <b>T<sub>2</sub></b> | <b>T<sub>3</sub></b> | <b>T<sub>4</sub></b> |
|--------------------------------------------------------------------------------------------------------|----------------------|----------------------|----------------------|----------------------|
| <b>Average marks of students in the final exam of the OOP course (maximum final exam marks is 100)</b> | 59.7                 | 63.8                 | 62.3                 | 69.4                 |

T<sub>1</sub>=2006 second term (IndusMarker was not used)

T<sub>2</sub>=2007 second term (IndusMarker was not used)

T<sub>3</sub>=2008 second term (IndusMarker was not used)

T<sub>4</sub>=2009 second term when IndusMarker was used to conduct practice tests.

The OOP course is taught during the second term of an academic year at BU. An academic year at BU starts from January and ends in December of the same year. So, “2006 second term” means second term of the 2006 academic year. Table 10 demonstrates that

there was a marked improvement in students' performance when IndusMarker was used to conduct practice tests before the final exam. The author was informed by the lecturers, who have taught the course during the last four terms, that there has not been any significant change in the course outline, final exam paper format and marking criteria. This indicates that the marked improvement in students' performance is mainly due to IndusMarker's usage rather than any other factor.

Similar data was also collected from the City School's "Darakhshan campus" and "PECHS Prep Boys-A" branches. The "required structures formulation, validation and storage" phase for all questions of the three science tests was completed at the "PAF chapter" branch. As with the OOP tests, these science tests were used as practice tests at the "Darakhshan campus" and the "PECHS Prep Boys-A" branches soon after they were ready to be used as practice tests. Even though science is a year-long course at grade seven, each year is divided into two semesters and topics covered in a semester are examined in the same semester. So, each semester has its own final exam. Table 11 gives average marks scored by students in the last four fall semester final exams of the grade seven "science" subject at the "Darakhshan campus" and the "PECHS Prep Boys-A" branches of the City School.

**Table 11. Average marks in grade seven "science" subject at the "Darakhshan campus" and the "PECHS Prep Boys-A" branches of the City School**

|                                                                            | <b>S<sub>1</sub></b> | <b>S<sub>2</sub></b> | <b>S<sub>3</sub></b> | <b>S<sub>4</sub></b> |
|----------------------------------------------------------------------------|----------------------|----------------------|----------------------|----------------------|
| <b>Average marks at the "Darakhshan campus" branch (maximum marks=100)</b> | 63.1                 | 64.6                 | 67.1                 | 70.7                 |
| <b>Average marks at the "PECHS Prep Boys-A" branch (maximum marks=100)</b> | 66.2                 | 67.9                 | 65.3                 | 69.8                 |

S<sub>1</sub>=2006-07 fall semester (IndusMarker was not used)

S<sub>2</sub>=2007-08 fall semester (IndusMarker was not used)

S<sub>3</sub>=2008-09 fall semester (IndusMarker was not used)

S<sub>4</sub>=2009-10 fall semester when IndusMarker was used to conduct practice tests.

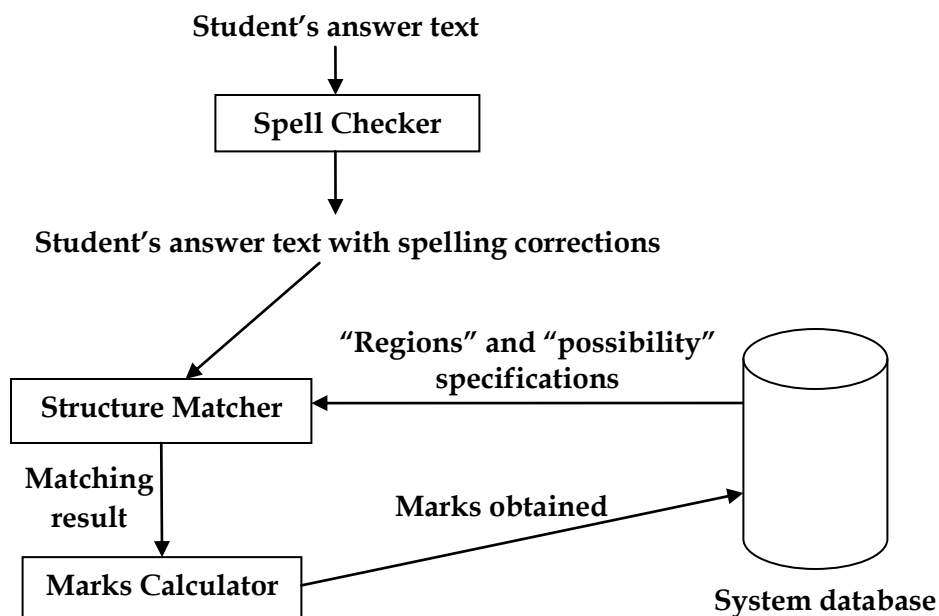
Table 11 shows that at both the City School's branches, the students performed best in the semester in which IndusMarker was used. As with the teachers at BU's Islamabad campus, the teachers at the two branches of the City School also informed the author that there has been no significant change in the "science" subject's course contents, the exam pattern and the marking criteria. So students' best performance in S<sub>4</sub> at both the branches cannot be attributed to changes in these factors. The only major difference is the use of IndusMarker in S<sub>4</sub>. In fact, IndusMarker's usage has led to superior students' performance at all three locations. This cannot be interpreted as co-incidence because IndusMarker's positive impact is consistent.

Taking mandatory practice tests using IndusMarker during term-time helps to keep students constantly revising course content. It may be argued that the same objective can also be achieved by taking paper and pencil class tests. But the main difference is that immediate feedback is provided by IndusMarker. Teachers do not normally mark students' class / practice tests "on time" and so by the time they finish marking such class / practice tests and return the marked answer scripts to students, it is usually too late to have a significant impact on students' performance in the final exam. Immediate feedback enables students to adjust their study plan on time and ultimately this leads to better performance in the final exam. This section analyzed the impact of IndusMarker on students' performance; the next section presents analysis of IndusMarker's performance without the "linguistic features analyzer".

## ***6.2 IndusMarker's Performance without "Linguistic Features Analyzer"***

As described in Section 3.3.1 and Section 4.1, the "linguistic features analyzer" is a sub-component of IndusMarker's "answer text analyzer" component. Its task is to perform some basic linguistic analysis (i.e. Part Of Speech (POS) tagging and Noun Phrase and Verb Group (NP & VG) chunking) on student's answer texts. To perform this task, a natural language parser called the Stanford Parser [32] and a self-developed Noun Phrase and Verb Group (NP & VG) chunker are exploited. During IndusMarker's evaluation, it was noticed that the processing performed by the Stanford Parser is computationally expensive and therefore consumes a lot of resources. It was decided to find out how IndusMarker would perform if the "linguistic features analyzer" component is removed. For this purpose, the

QAML had to be modified and its two linguistic-specific constructs, i.e. the “noun phrase required” construct and the “verb group required” construct, were removed from the QAML language. Figure 12 depicts the architecture of the “answer text analyzer” after removal of the “linguistic features analyzer” sub-component. IndusMarker’s code was adapted for this change.



**Figure 12. Architecture of the “answer text analyzer” component after removal of the “linguistic features analyzer” sub-component**

The students’ answers data collected at BU’s Karachi campus was re-utilized to evaluate the performance of IndusMarker without the “linguistic features analyzer” sub-component. The required structures validated and stored earlier for the same questions were re-formulated using the modified version of QAML, i.e. QAML with the linguistic-specific constructs removed. Table 12 (on the next page) demonstrates IndusMarker’s performance with and without linguistic features consideration. These results are for the same allowed short-answer question types that were used for IndusMarker’s evaluation at BU. The number of questions used and the average answer length for questions of each question type have not been included in Table 12 because the data set used was the same as that used for IndusMarker’s evaluation at BU’s Karachi campus, i.e. these quantities have already been presented in Table 4 (on page 114).

The average human-system agreement rates when IndusMarker considered linguistic features have been presented again in this table (i.e. these values have been presented before in Table 4 on page 114). These values have been included in Table 12 so that they may be compared with the corresponding values obtained when IndusMarker did not consider linguistic features. This second set of values, i.e. average human-system agreement rates without linguistic features consideration, has been calculated using the same method and formula given in Section 5.3.2.7.

**Table 12. IndusMarker’s performance with and without consideration of linguistic features**

| <b>S. No.</b> | <b>Question type</b>                | <b>Average human-system agreement rate when linguistic features were considered by IndusMarker</b> | <b>Average human-system agreement rate when linguistic features were not considered by IndusMarker</b> |
|---------------|-------------------------------------|----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| 1             | "True" / "False" question           | 100%                                                                                               | 100%                                                                                                   |
| 2             | Sentence completion                 | 99.67%                                                                                             | 99.67%                                                                                                 |
| 3             | Single term generation              | 99%                                                                                                | 99%                                                                                                    |
| 4             | "Quantity" required                 | 99.42%                                                                                             | 99.42%                                                                                                 |
| 5             | "Numerical value" generation        | 98.83%                                                                                             | 98.83%                                                                                                 |
| 6             | "Location" required                 | 99.16%                                                                                             | 99.16%                                                                                                 |
| 7             | "Program statement output" required | 98.79%                                                                                             | 98.79%                                                                                                 |
| 8             | Single phrase generation            | 93.8%                                                                                              | 92.7%                                                                                                  |
| 9             | "Example" required                  | 99.33%                                                                                             | 99.33%                                                                                                 |
| 10            | List                                | 92%                                                                                                | 91%                                                                                                    |
| 11            | Short Explanation / Description     | 92.33%                                                                                             | 91.41%                                                                                                 |
| 12            | "Situation" or "context" required   | 93.5%                                                                                              | 92.58%                                                                                                 |
| 13            | Definition                          | 93.67%                                                                                             | 93.08%                                                                                                 |
| 14            | Contrast                            | 84.10%                                                                                             | 82.05%                                                                                                 |
| 15            | Compare                             | 94%                                                                                                | 93.12%                                                                                                 |
| 16            | Composite questions                 | 92%                                                                                                | 91.16%                                                                                                 |

The corresponding average human-system agreement rates, resulting from the two IndusMarker’s architectures, are equal for the first seven and the ninth allowed short-answer question types. This indicates that for the simpler allowed short-answer question types, there is no difference at all in the performance of the two variants of IndusMarker’s architecture. For more complex question types, IndusMarker without “linguistic features analyzer” performs slightly less accurately. But the difference in accuracy is minor and not major. The major benefit of the new architecture and of the new approach of not considering linguistic features is that the processing of student’s answer text becomes much faster due to the removal of the Stanford Parser [32]. According to the author’s viewpoint, the benefit of the new approach outweighs its disadvantage. The use of the Stanford Parser slows down IndusMarker’s processing speed especially in situations where student’s answer text comprises many sentences. Another problem with the Stanford Parser is that there is no guarantee that it will produce an accurate parse of student’s answer text, and therefore relying on its output for further processing by the “structure matcher and marks calculator” component is itself problematic. The number of constructs in QAML also becomes less with this new approach and the learner of QAML will have to learn fewer QAML constructs. Having presented and analyzed IndusMarker’s performance without “linguistic features analyzer”, the next section presents a comparison of IndusMarker with other similar systems.

### ***6.3 Comparison of IndusMarker with Other Similar Systems***

The purpose of this section is to compare IndusMarker with the other three similar systems identified in chapter 2. This comparison is presented in the following sub-sections (i.e. Sections 6.3.1 to 6.3.3).

#### **6.3.1 Comparison of IndusMarker with C-rater**

A number of similarities and differences exist. The task of both systems, in general, is to compare student’s answer with a model / required structure<sup>12</sup> [49], [2]. In both systems, the “model building / required structure specification” stage is performed by hand but comparison is fully automated. The student’s marks are computed based on the result of this

---

<sup>12</sup> “Model” in the case of C-rater and “required structure” in the case of IndusMarker.



comparison. The model / required structure represents the “space” of potentially “all possible correct answers”.

C-rater does not have a formal language for model building. The model building task is performed using C-rater’s own graphical interface called Alchemist [2]. Model is built by specifying patterns in English. It is the task of C-rater to extract linguistic features from these patterns using a set of Natural Language Processing (NLP) tools [50]. In the case of IndusMarker, the required structure is specified in QAML using the QAML structure editor. Once the QAML-based required structure has been fully specified, it is stored in the IndusMarker database without extraction of linguistic features. The reason for this is that IndusMarker mainly relies on text structure analysis (i.e. analysis of occurrence and ordering of words in an answer text) and does not rely heavily on linguistic feature analysis (i.e. analysis of grammatical features in an answer text). IndusMarker has demonstrated comparable performance even after removing the “linguistic feature analyzer” component from the system. In short, linguistic feature analysis is not crucial to IndusMarker (this is unlike C-rater which relies heavily on linguistic feature analysis). QAML has only two linguistic-specific constructs: the “noun phrase required” construct and the “verb group required” construct. The linguistic-specific QAML constructs are included in QAML-based specifications just like other QAML constructs and so there is no linguistic feature extraction by IndusMarker during QAML-based required structure specification and storage. This is better in the sense that unlike C-rater, there are no complex, computationally expensive NLP tasks to perform by IndusMarker at the “required structure specification / model building” stage. It may be argued that model building task is easier in C-rater from the user’s perspective as the user does not have to write patterns in a formal language, instead he/she can easily express the patterns in English. But C-rater’s model building still needs to be done by a content expert [2] and requires considerable time and effort [50].

The matching or comparison of a student’s answer text with the “model” built or the “required structure” specified is carried out by Goldmap [24] in the case of C-rater and the “structure matcher and marks calculator” component in the case of IndusMarker [49]. In the case of C-rater, the system performs extensive pre-processing of student’s answer text before it is used by Goldmap for the comparison. This pre-processing involves normalizing the

student's response into a canonical representation [2]. To build this representation, C-rater extracts the underlying structure of the response, resolves pronoun reference, normalizes across inflected words and recognizes the use of similar terms and synonyms. A number of NLP tools such as OpenNLP parser, feature extractor, pronoun resolver, morphology analyzer, etc. are used in building the canonical representation [50]. There is also some pre-processing done in the case of IndusMarker but this pre-processing is much less complex and much less "deep" than that performed by C-rater. For example, there is no pronoun resolution and no morphology analysis in the case of IndusMarker.

C-rater's comparison module, Goldmap, is based on maximum entropy modeling [24]. Basically, given a set of attributes, constraints over these attributes, and a set of training data consisting of pairs of sentences that are both manually-annotated for match or no-match and automatically annotated according to the set of attributes, Goldmap learns a matching model. Given an unseen answer, the matching model outputs a probability on the match between the unseen answer and a model answer. In general, a threshold of 0.5 is used to determine a match. Scoring rules are then applied to obtain a score. IndusMarker's "structure matcher and marks calculator" performs matching of part-of-speech tagged and noun phrase & verb group chunked answer text with the pre-specified QAML-based required structure. Unlike C-rater, there is no learning of the matching model / technique. The matching algorithm is "hard-coded" in IndusMarker. The precise details of the algorithms used in IndusMarker's "structure matcher" and "marks calculator" components are given in Section 4.1.3 and Section 4.1.4.

Overall, the design of C-rater is more complex than IndusMarker and since the amount of NLP is significantly more in C-rater, C-rater is believed to be computationally much more expensive than IndusMarker. What about the important issue of system accuracy? Both systems have demonstrated a high-level of accuracy with the data they were evaluated with, but neither of the two systems is 100% accurate. To counter this problem of a less than 100% human-system agreement rate, the proposed use of IndusMarker is restricted to low-stake, practice tests. The main aim is to provide timely feedback to both students and teachers so that they may adjust their future study or teaching plan. In the case of C-rater, it is not clear how the problem of less than 100% human-system agreement rate

has been tackled. Unlike IndusMarker, the list of allowed short-answer question types is also not given for C-rater. So, IndusMarker’s operational domain is much more clearly defined.

### **6.3.2 Comparison of IndusMarker with the Oxford-UCLES System**

Unlike C-rater, the Oxford-UCLES system uses a formal language in which to write patterns [3]. This feature of the Oxford-UCLES system is similar to IndusMarker. QAL/QAML, however, is more “powerful”, i.e. more expressive, compared to the language used in the Oxford-UCLES system. There are no equivalent Oxford-UCLES pattern-writing language’s constructs for a number of useful QAL/QAML constructs. An example of such a QAL/QAML construct is the “allowed permutation” construct. The syntax and semantics of the Oxford-UCLES system’s pattern-writing language is also more difficult to understand and therefore, more difficult to learn compared with QAL/QAML. Sukkarieh et al. [3] states that there is a pattern-writing tool available in the Oxford-UCLES system to help users write patterns. Detailed description of the working and effectiveness of this pattern-writing tool has not been provided. IndusMarker, on the other hand, has a comprehensive “QAML structure editor” (a detailed description of which is given in Section 3.3.2 and Section 4.2).

Perhaps the most important difference between the pattern-writing languages of the two systems is that IndusMarker’s QAML exploits the utility of XML while the Oxford-UCLES system’s pattern-writing language does not exploit this utility. XML is a language for creating new markup languages or in other words, it is a meta-language [30]. The new markup languages created are called XML vocabularies [51]. QAML is an XML vocabulary. There are many benefits of defining QAML as a sub-language of XML because all the benefits of using XML are inherited by QAML. XML is currently the de facto standard format for data handling and exchange [29]. It is also platform-independent, well-supported and its format is human-readable [52].

Another important QAL/QAML concept is the use of “regions specification”. The required answer structure for longer, multi-part answers can be easily represented if the expected answer text is considered to consist of various regions. So, an important aspect of the system’s capability is the ability to process “composite questions” structured from other “composite” or “simpler” questions. In this way IndusMarker can mark longer, factual answers. This feature is lacking in the Oxford-UCLES system.

Evaluations of the two systems have also been performed differently. Evaluation of IndusMarker is more extensive and more carefully designed. Table 13 demonstrates some differences between the evaluations of the two systems.

**Table 13. Comparison of IndusMarker’s and the Oxford-UCLES system’s evaluations**

|                                                              | <b>IndusMarker’s evaluation</b>                                                                          | <b>Oxford-UCLES system’s evaluation</b>                     |
|--------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| <b>Number of courses used</b>                                | 2                                                                                                        | 1                                                           |
| <b>Title of courses used</b>                                 | <ul style="list-style-type: none"> <li>• Object-Oriented Programming (OOP)</li> <li>• Science</li> </ul> | <ul style="list-style-type: none"> <li>• Biology</li> </ul> |
| <b>Number of questions used</b>                              | 123 questions (78 OOP questions + 45 Science questions)                                                  | 9 questions                                                 |
| <b>Number of tests/exams used</b>                            | 9 tests (6 OOP tests + 3 Science tests)                                                                  | 1 biology exam                                              |
| <b>Number of institutions where evaluation was performed</b> | 2                                                                                                        | 1                                                           |
| <b>Size of the data set used in pattern-writing</b>          | 25 students’ answers (OOP tests), 26 to 33 students’ answers (Science tests)                             | 200 students’ answers                                       |
| <b>Size of test data</b>                                     | 200 students’ answers (OOP tests), 70 students’ answers (Science tests)                                  | 60 students’ answers                                        |

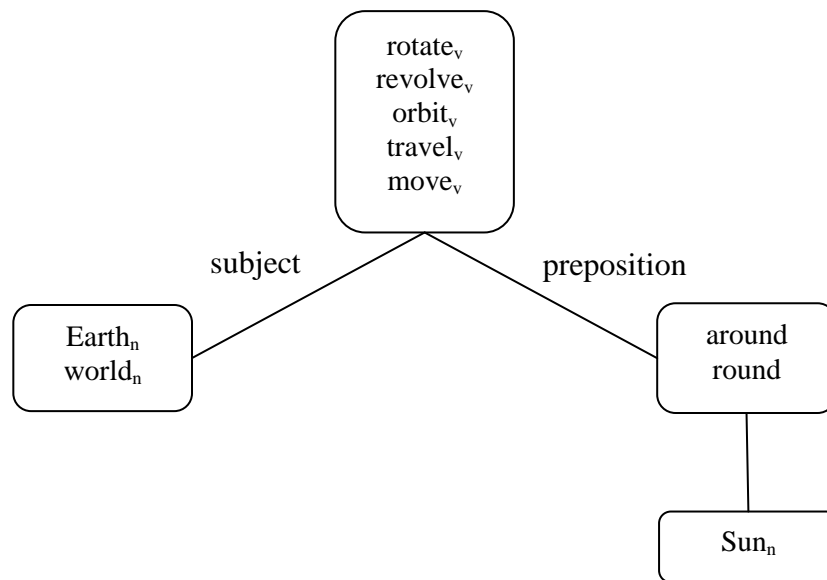
The number of questions used is much greater in the case of IndusMarker’s evaluation. The size of test data, the number of courses used and the number of institutions where evaluation was performed, are all greater for IndusMarker’s evaluation compared with

the Oxford-UCLES system’s evaluation. A more thorough evaluation of the Oxford-UCLES system could have provided a better insight into its performance and any short-comings.

The allowed short-answer question types are clearly stated in the case of IndusMarker. This means the domain and scope of IndusMarker are more clearly defined than the Oxford-UCLES system. Since questions of each allowed short-answer question type were included in IndusMarker’s evaluation, it was possible to carry out a comparative analysis of IndusMarker’s performance on the various allowed short-answer question types. This enabled better understanding of IndusMarker’s capabilities and limitations.

### 6.3.3 Comparison of IndusMarker with Automark

Like IndusMarker, Automark frames computerized marking of free-text responses as an information extraction task [6]. But there are important differences in the design of the two systems and their respective capabilities and limitations also vary. Automark searches free-text responses for pre-defined computerized mark scheme answers. These mark scheme answers are represented as syntactic-semantic templates. For example, Figure 13 illustrates a simple template for the mark scheme answer: “The Earth rotates around the Sun”.



**Figure 13. Illustration of an Automark’s mark scheme template**

The template shown can be expected to match a student's response if the response contains one of the stated verbs (rotate, revolve, orbit, travel, move) with one of the stated nouns (Earth, world) as its subject, and around / round the sun in its preposition. An important point to note here is that linguistic features such as subject and preposition of student's response are neither considered by IndusMarker's matching algorithm nor there are any corresponding constructs in QAL / QAML related to these linguistic features. IndusMarker has demonstrated reasonably good performance without considering these linguistic features of student's response. Mitchell et al. [6] did not demonstrate through appropriate evaluation how much the consideration of these additional linguistic features (such as subject and preposition of student's response), help in improving the accuracy of Automark. Any claimed benefit of using "deeper" NLP should be justified through empirical evidence.

The number of question types used in Automark's evaluation is much less than the number of question types used in IndusMarker's evaluation. Only four question types are used in Automark's evaluation compared with the sixteen question types used in IndusMarker's evaluation. The question types used in Automark's evaluation are: "single word generation", "single value generation", "generation of a short explanatory sentence" and "description of a pattern in data". Only one question was used for each question type. This is in contrast with IndusMarker's evaluation where multiple questions were used for each question type. IndusMarker has been shown to produce high human-system agreement rates with relatively complex question types such as "definition", "contrast", "compare", "composite questions", etc. Automark has not been tested on such complex question types and therefore it is not possible to comment on how Automark would perform on these question types. Unlike IndusMarker, the design of Automark contains no provision for marking longer, factual answers. The use of QAL/QAML "regions specification" enables IndusMarker to mark longer, multi-part answers such as answers to composite questions.

## **6.4 Conclusion**

The salient feature of IndusMarker (from the pedagogical perspective) is that it can provide practice tests and immediate feedback to students regardless of the size of the class. The lecturer has to initially spend some time developing and validating the required

structures when a practice test is conducted for the first time. But, once the required structures are finalized, the same practice test may be repeated wherever the same course material is taught. The lecturer conducting the practice test, after the first test, does not need to spend any time manually marking the test papers. No other similar automated short-answer marking system has been applied for such a practical and useful purpose.

Provision of immediate and on-time feedback about students' performance in practice tests, enables both teachers and students to adjust their teaching/study plans on-time, i.e. it helps them in decision-making. Making the right decision at the right time is very important for both students and teachers. For example, if a student performs poorly in a practice test conducted using IndusMarker, he/she can analyze his/her mistakes and shortcomings on-time. The student can then decide to concentrate on the topics in which he/she has performed poorly. If the student's overall performance is bad, then he/she may decide to increase his/her study time and effort. From the teacher's perspective, the system is equally useful. For example, a teacher may decide to speed up or slow down delivery of course material depending on students' collective performance in practice tests. If the students' collective performance is good, then the teacher has the option of speeding up (if needed) the delivery of course material. If the students' collective performance is bad, the teacher may decide to slow down the pace of the course material delivery. The basic point is that on-time availability of students' information is very important in decision-making. Taking mandatory practice tests at regular intervals during term-time also helps to keep students revising the course material throughout the term rather than just a few days before the final exam.

IndusMarker has been evaluated at two different educational institutions. At each institution, a subject of study was chosen to conduct practice tests. The allowed short-answer question types were determined and only questions of these types were included in the practice tests. The domain and scope of IndusMarker are much more clearly defined compared with other similar systems. Evaluation of IndusMarker is also much better planned and more systematic. High human-system agreement rates are achieved for all the allowed short-answer question types (but agreement rates are higher for simpler question types compared with the agreement rates for more complex question types). Moreover,

comparable human-system agreement rates are also achieved when IndusMarker does not consider linguistic features and only relies on non-linguistic, structural analysis of the student's answer text. This is an important finding as non-linguistic, structural analysis of student's answer text is much more efficient and much less computationally expensive.

The author has received positive feedback about the system from the lecturers involved and also from students of the two institutions. In chapter 1, the author referred to some experimental reports [18], [19], [20] that have demonstrated that taking a practice test on studied material promotes subsequent learning and retention of that material in a final test/exam. In addition, the experimental reports also demonstrate that practice tests produce learning/retention advantages beyond that enjoyed from repeated study. So, do these concepts materialize in the case of IndusMarker's usage? When IndusMarker was used to conduct practice tests at the three different locations, its usage resulted in superior students' performance in the final exams at all the three locations. This reconfirms the validity of the experimental reports and also demonstrates the positive impact of IndusMarker.

The following are the scientific contributions of the author's research:

1. A new automated short-answer marking system called IndusMarker. Unlike other similar systems, IndusMarker relies mainly on structure-editing and structure-matching rather than linguistic features analysis.
2. A new purpose-designed language called QAML (defined as a sub-language of XML) to specify the "required answer structures".
3. A structure-editor to help develop the QAML structures (i.e. the use of "structure editing" to support the development of descriptions of correct "answer structures" for subsequent "structure matching"). If the lecturers/teachers use the system properly they are forced to develop their descriptions in the form that the structure editor constrains them to, and the resulting answer-structures can be used for accurate automated marking.

In future, the author anticipates that researchers will be interested in extending his work. Possible directions for future research/work include:

1. Modification/improvement of IndusMarker's architecture and algorithm so that the range of the allowed short-answer question types can be increased.



2. Find out if some other short-answer question type(s) (apart from those included in the list of allowed short-answer question types) can be marked with a high degree of accuracy using the current IndusMarker's architecture and algorithm.
3. Evaluate IndusMarker's performance and usage at other levels of education and in other subject areas.
4. IndusMarker can be further enhanced by including features that can provide detailed statistical analysis of students' performance for both lecturers and students so that each may adjust or modify their teaching or learning approach for the course in a better and more comprehensive way.
5. IndusMarker may also be integrated with some other type of assessment system (such as essay marking system, program/software marking system, etc.) to form a single, comprehensive system. For example, in recent years a number of automated marking systems for program texts, i.e. texts written in a programming language, have been developed [53]. However, these systems cannot mark short-answers expressed in natural language<sup>13</sup>. Short-answer questions provide a very useful means of testing theoretical concepts associated with a programming course. IndusMarker may be integrated with other program marking systems to form a single system that can mark both programming exercises as well as short-answer questions.
6. Extension of QAML so that required answer structure for a greater range of short-answer question types may be expressed in QAML.

---

<sup>13</sup> It is common, though arguably not effective (a separate discussion to be held elsewhere), to include natural language comments in program texts. Such comments could, in principle, be analysed by IndusMarker and used in the assessment of program texts, albeit only the natural language commented parts of such texts.

## References

- [1] K. E. Holbert and G. G. Karady, "Strategies, Challenges and Prospects for Active Learning in the Computer-Based Classroom". *IEEE Transactions on Education*, vol. 52, no. 1, pp. 31-38, Feb. 2009.
- [2] C. Leacock and M. Chodorow, "C-rater: Automated Scoring of Short-Answer Question". *Computers and the Humanities*, vol. 37, no. 4, pp. 389-405, Nov. 2003.
- [3] J. Z. Sukkarieh, S. G. Pulman and N. Raikes, "Auto-marking: using computational linguistics to score short, free text responses". *Paper presented at the 29<sup>th</sup> annual conference of the International Association for Educational Assessment (IAEA), Manchester, UK, 2003.*
- [4] J. Z. Sukkarieh, S. G. Pulman and N. Raikes, "Auto-marking 2: An update on the UCLES-Oxford University research into using computational linguistics to score short, free text responses". *Paper presented at the 30<sup>th</sup> annual conference of the International Association for Educational Assessment (IAEA), Philadelphia, USA, 2004.*
- [5] J. Z. Sukkarieh and S. G. Pulman, "Automatic Short Answer Marking". *Proceedings of the 2<sup>nd</sup> Workshop on Building Educational Applications Using NLP, Association for Computational Linguistics*, pp. 9-16, June 2005.
- [6] T. Mitchell, T. Russel, P. Broomhead and N. Aldridge, "Towards robust computerized marking of free-text responses". *Proceedings of the Sixth International Computer Assisted Assessment Conference, Loughborough, UK: Loughborough University, 2002.*
- [7] W. Rabinowitz and R. M. W. Travers, "Problems of defining and assessing teacher effectiveness". *Educational Theory*, vol. 3, no. 3, pp. 212-219, July 1953.
- [8] D. G. Ryans, "Prediction of teacher effectiveness". *Encyclopedia of educational research*, C. W. Harris, ed., New York: Macmillan, pp. 1486-1491, 1960.
- [9] R. J. Shavelson, "The basic teaching skill: decision making". Stanford, CA: Stanford University School of Education Centre for Research and Development in Teaching, 1973.

- [10] L. W. Anderson, *Classroom Assessment: Enhancing the Quality of Teacher Decision Making*. Mahwah, New Jersey: Lawrence Erlbaum Associates, 2003.
- [11] D. Rowntree, *Assessing Students: How shall we know them?* London: Kogan Page Ltd, pp. 15-31, 1987.
- [12] N. Falchikov, *Improving Assessment through Student Involvement: Practical Solutions for Aiding Learning in Higher and Further Education*. London: Taylor & Francis Routledge, pp. 1-5, 2005.
- [13] T. R. Guskey, "How Classroom Assessments Improve Learning". *Educational Leadership*, vol. 60, no. 5, pp. 6-11, Feb. 2003.
- [14] J. H. McMillan, *Classroom Assessment: Principles and Practice for Effective Instruction*. Boston, MA: Allyn and Bacon, 1997.
- [15] S. Chappuis and R. J. Stiggins, "Classroom Assessment for Learning," *Educational Leadership*, vol. 60, no. 5, pp. 40-43, Sep. 2002.
- [16] J. T. Dillon, *Questioning and Teaching: A Manual of Practice*. New York: Teachers Coll. Press, 1988.
- [17] L. R. Gay, "The Comparative Effects of Multiple-Choice versus Short-Answer Tests on Retention". *Journal of Educational Measurement*, vol. 17, no. 1, pp. 45-50, Spring 1980.
- [18] M. A. McDaniel, J. L. Anderson, M. H. Derbish and N. Morrisette, "Testing the testing effect in the classroom". *European journal of cognitive psychology*, vol. 19, no. 4/5, pp. 494-513, 2007.
- [19] M. A. McDaniel, M. D. Kowitz, and P. K. Dunay, "Altering memory through recall: The effects of cue-guided retrieval processing". *Memory and Cognition*, vol. 17, no. 4, pp. 423-434, 1989.

- [20] M. A. McDaniel and M. E. J. Masson, "Altering memory representations through retrieval". *Journal of Experimental Psychology: Learning, Memory and Cognition*, vol. 11, no. 2, pp. 371-385, 1985.
- [21] W. J. Hanson, "Creation of Hierarchic Text with a Computer Display", Ph.D. thesis. Stanford University, 1971.
- [22] T. K. Landauer, D. Laham and P. W. Foltz, "Automatic Essay Assessment". *Assessment in Education*, vol. 10, no. 3, Nov. 2003.
- [23] T. Kakkonen and E. Sutinen, "Automatic Assessment of the Content of Essays Based on Course Materials". *Proceedings of International Conference on Information Technology: Research and Education (ITRE)*, pp. 126-130, 2004.
- [24] J. Sukkarieh and E. Bolge, "Leveraging C-Rater's Automated Scoring Capability for Providing Instructional Feedback for Short Constructed Responses", *Intelligent Tutoring Systems, Lecture Notes in Computer Science*, vol. 5091, Heidelberg: Springer, pp. 779-783, 2008.
- [25] J. L. Fleiss, *Statistical Methods for Rates and Proportions*. New York: John Wiley & Sons, pp. 212-236, 1981.
- [26] D. Appelt and D. Israel, "Introduction to information extraction technology". *IJCAI Tutorial*, 1999.
- [27] S. Valenti, F. Neri and A. Cucchiarelli, "An overview of current research on automated essay grading". *Journal of Information Technology Education*, vol. 2, pp. 319-330, 2003.
- [28] J. P. Gibson, "A noughts and crosses Java applet to teach programming to primary school children," *Proceedings of the 2nd international conference on Principles and practice of programming in Java*, pp. 85 - 88, 2003.
- [29] S. Holzner, *Inside XML*. Indianapolis, Indiana: New Riders, pp. 9-10, 2001.
- [30] D. Gulbransen, *The Complete Idiot's Guide to XML*. Indianapolis, Indiana: Que, pp. 131 - 133, 2000.

- [31] S. S. Laurent and E. Cerami, *Building XML Applications*. New York: McGraw-Hill, p. 46, 1999.
- [32] The Stanford Natural Language Processing Group, “The Stanford Parser: A statistical parser”, <http://nlp.stanford.edu/software/lex-parser.shtml>. [Last accessed on 15-05-2010]
- [33] i-net software, “JOrtho – a Java spell-checking library”, <http://www.inetsoftware.de/other-products/jortho>. [Last accessed on 26-06-2010]
- [34] Wiktionary, “Wiktionary – a wiki-based Open Content dictionary”, <http://wiktionary.org/>. [Last accessed on 18-06-2010]
- [35] M. C. de Marneffe, B. MacCartney and C. D. Manning, “Generating Typed Dependency Parses from Phrase Structure Parses,” *Proc. The fifth international conference on Language Resources and Evaluation*, pp. 449-454, 2006.
- [36] B. McLaughlin, *Java and XML*. Sebastopol, California: O’Reilly & Associates, pp. 125-135, 2000.
- [37] K. Cagle, M. Corning, J. Diamond, T. Duynstee, O. G. Gudmundsson, M. Mason, J. Pinnock, P. Spencer, J. Tang and A. Watt, *Professional XSL*, Birmingham, UK: Wrox Press, 2001.
- [38] B. S. Bloom, *Taxonomy of Educational Objectives*. New York: Longman’s, Green & Company, 1954.
- [39] E. D. Gagne, C. W. Yekovich and F. R. Yekovich, *The cognitive psychology of school learning* (2<sup>nd</sup> ed.). New York: HarperCollins College Publishers, 1997.
- [40] Official website of Bahria University. Available online: <http://bimcs.edu.pk/about-bahria/why-bahria.htm>. [Last accessed on 2-07-2010]
- [41] Official website of Higher Education Commission Pakistan. Available online: <http://www.hec.gov.pk>. [Last accessed on 6-07-2010]

- [42] Official website of Pakistan Engineering Council. Available online: <http://www.pec.org.pk>. [Last accessed on 3-07-2010]
- [43] M. Farrow and P. J. B. King, "Experiences With Online Programming Examinations". *IEEE Transactions on Education*, vol. 51, no. 2, May 2008, pp. 251-255.
- [44] Course website of "Object Oriented Programming" course. Available online: <http://sites.google.com/site/sen142oop/Home>. [Last accessed on 6-05-2010]
- [45] F. Marton and R. Saljo, "On qualitative differences in learning- I: Outcome and process", *British journal of Educational Psychology*, vol. 46, no. 1, pp. 4-11, Feb 1976.
- [46] L. S. Ming, "An ICT-Based Technique for Assessment of Short-Answer Question". Paper presented at the 7<sup>th</sup> International Conference on Information Technology Based Higher Education and Training, Sydney, Australia, 2006.
- [47] Official website of the City School, Pakistan. Available online: <http://www.thecityschools.edu.pk>. [Last accessed on 18-05-2010]
- [48] A. J. Swart, "Evaluation of Final Examination Papers in Engineering: A Case Study Using Bloom's Taxonomy," *IEEE Transactions on Education*, preprint, doi: 10.1109/TE.2009.2014221.
- [49] R. Siddiqi, C. J. Harrison and R. Siddiqi, "Improving Teaching and Learning through Automated Short-Answer Marking," *IEEE Transactions on Learning Technologies*, preprint, 9 Feb. 2010.
- [50] J. Z. Sukkariéh and S. Stoyanchev, "Automating Model Building in C-rater," *Proceedings of the 2009 Workshop on Applied Textual Inference, ACL-IJCNLP 2009*, pp.61-69, 2009.
- [51] D. Gulbransen, K. Bartlett, E. Bingham, A. Kachur, K. Rawlings and A. H. Watt, *Using XML (Second Edition)*. Indianapolis, Indiana: Que, pp. 7-24, 2002.
- [52] W3C Communications Team, "XML in 10 Points," <http://www.w3.org/XML/1999/XML-in-10-points>. [Last accessed on 13-06-2010]

- [53] C. Daly and J. M. Horgan, "An Automated Learning System for Java Programming". *IEEE Transactions on Education*, vol. 47, no. 1, pp. 10-17, February 2004.
- [54] C. Stergiou and D. Siganos, "Neural Networks," [http://www.doc.ic.ac.uk/~nd/surprise\\_96/journal/vol4/cs11/report.html](http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html). [Last accessed on 28-08-2010]
- [55] R. Witte, "Introduction to Text Mining," <http://www.rene-witte.net/system/files/IntroductionToTextMining.pdf>. [Last accessed on 28-08-2010]
- [56] M. Hearst, "What is Text Mining?" <http://people.ischool.berkeley.edu/~hearst/text-mining.html>. 2003. [Last accessed on 28-08-2010]

## **Appendices**

The following are appendices for this thesis.

### ***Appendix 1: User Manual***

IndusMarker system is designed to be used by two types of users: (1) examiners, and (2) students. The system's user interface is divided in to two parts/sections i.e. one part/section is for examiners and the other part/section is for students. The system enables examiners to perform the following tasks:

1. *Test creation*: This task involves specifying questions (and their associated marks) for the test being created and stored in the system database.
2. *“Answer structure” preparation*: This task involves “answer structure” formulation, validation and correction. Once “answer structure” for all questions of a test have been prepared, the test becomes available for use as practice test.
3. *Viewing students’ results*: Examiners can view the overall performance and also the individual student’s performance in various practice tests.

The system enables students to perform the following tasks:

1. *Taking practice test*
2. *Viewing results*: A student can view his/her performance in the various practice tests that he/she has taken. Performance details are presented in the form of marks obtained in each practice test taken. In addition, marks obtained in each question (of the practice tests) can also be viewed. Student’s answer to each question can also be analyzed with respect to the required “answer structure” for that question.

To accomplish each of the above tasks, appropriate user interfaces have been developed. The following is explanation of how the above listed tasks are accomplished using IndusMarker’s user interface (since there are two sections of the system, i.e. examiner’s section and student’s section, the explanation of user interface has also been divided accordingly):



## Examiner's Section

When an examiner logs into the IndusMarker system, he/she is presented the examiner's main menu. The main menu contains examiner's options or the tasks that he/she can perform. Appendix 2 contains a screen shot of the "Examiner's Main Menu" screen. If the examiner wants to create a new test, he/she should press the "Test Creation" button. The test creation process starts once the examiner has pressed the "Test Creation" button. Appendices 3 to 7 depicts screen shots of various screens used in the test creation process. The test creation process is explained below (in the form of pseudo code):

First the examiner is asked about the number of questions in the test (Appendix 3)

FOR each question in the test BEGIN

    Ask whether or not the question has sub-parts (Appendix 4)

    IF examiner indicates that the question has sub-parts THEN

        A screen pops-up asking examiner to enter main text of the question and also requiring him/her to enter the number of sub-parts (Appendix 5)

        FOR each sub-part of the question BEGIN

            Enter sub-part's text and marks (Appendix 6)

        END FOR

    END IF

    ELSE IF examiner indicates that the question has NO sub-parts THEN

        Examiner is asked to enter question's text and marks (Appendix 7)

    END IF

END FOR

If examiner wants to prepare/develop "answer structure" for various questions of the test(s) created, then he/she should press the "Structure Preparation" button on the "Examiner's Main Menu" screen (Appendix 2). Once the examiner has pressed the "Structure Preparation" button, a "Test Selection" screen (Appendix 8) appears. The "Test Selection" screen enables examiner to select the test for "answer structure" development. On the left-hand side of the screen, there is a list of tests that have been created and stored. When a test is selected in the list, its details are displayed on the right-hand side. If the test is

ready for practice, i.e. all questions of the test have their “answer structure” finalized/developed, then this will be indicated in the corresponding text field on the right-hand side along with other test details. When examiner has selected the desired test, he/she should press the “OK” button to move to the “Question Selection” screen (Appendix 9).

The user interface of the “Question Selection” screen consists of three parts. On the left-hand side, there is a tree-based structure containing question and sub-question numbers. When a question number is selected, the question’s text appears in the text area at the middle of the “Question Selection” screen. Once the desired question has been selected, the examiner should press the “Manual Marking” button to start the manual marking of students’ answers for the question. If the manual marking has already been done, then the “Manual Marking” button will appear disabled and therefore “answer structure” formulation and validation can now be started by pressing the “Structure Creation” button.

When the “Structure Creation” button is pressed, the “Answer Structure” Formulation & Validation screen appears (Appendix 10). Training data set for the question is displayed in the middle. In order to formulate “answer structure” for the question, the “Formulate Structure” button should be pressed. When the “Formulate Structure” button is pressed, a new screen pops up on top of the “Answer Structure” Formulation and Validation screen. The new screen (Appendix 11) is for regions specification structure editing. Once the regions specification has been developed and saved, the “Finish” button on the screen is pressed and this brings up the “Possibility Structure Editor” screen (Figure 7 of the main thesis text). The GUI for the “regions” specification is similar to the GUI for “possibility structure” specification. The QAML specification is represented as a tree structure on GUI. The user of the editor constructs the QAML specification by selecting QAML elements from the list displayed on the left. When a user selects a node in the QAML specification tree, all the possible QAML child elements are displayed on the list. In order to insert a QAML element in to the specification tree, the user has to select a node in the tree and also select a QAML element from the list and then press the “Insert Element” button. The QAML element from the list is added as a child of the selected node in the tree. The leaf nodes of the QAML specification trees consist of parsed character data that is entered by the user through the text area just beneath the pane containing the QAML specification tree. Functionalities

available to the examiner may be executed by pressing buttons present on the screen. As the specification tree changes, the changes are reflected in the QAML document that is developed dynamically at the back end. The status of the QAML document can be easily viewed at any time by pressing the “View XML” button. The QAML document is validated by pressing the “Validate” button. The result of validation is displayed on the output text area of the screen. Once a QAML document is completed and validated, it is saved by pressing the “Save Possibility” or “Save Regions Specification” button.

Once “answer structure” formulation has finished, the regions specification structure editor (Appendix 11) and the possibility specification structure editor (Figure 7 of the main thesis text) screens close. The “Formulate Structure” button appears disabled on “Answer Structure” Formulation & Validation screen (Appendix 10) once “answer structure” formulation has finished. Now, the formulated “answer structure” is ready for validation. Validation is performed by pressing the “Validate Structure” button and this result in automated marks for students’ answers being computed and the discrepancies between automated and manual markings identified. If the examiner wishes to resolve the discrepancies then he/she should press “Correct Structure” button which will enable examiner to modify the stored “answer structure” using the same structure editor screens (Figure 7 and Appendix 11). Once “answer structure” has been finalized, the examiner should press the “Finish” button to return to the examiner’s main menu (Appendix 2).

An examiner can also view students’ results for the various practice tests conducted. To do this, the examiner has to press the “View Results” button on the “Examiner’s Main Menu” screen (Appendix 2). After selecting the desired test on the “Test Selection” screen, the details for all the practices (of that test) are displayed on “Practice Results” screen (Appendix 12). The average marks obtained and the maximum marks for the practice are given along with other details about the practice currently selected. Individual student’s performance can also be viewed by pressing “Students’ Details” button. This will bring “Individual Student’s Performance” screen (Appendix 13) on the computer display. The screen contains list of students who appeared in the practice along with the marks scored by each student in the practice.

## **Student's Section**

When a student logs into the IndusMarker system, he/she is presented the student's main menu (Appendix 14). A student can either take a practice test or view results of his/her past performances in various practice tests taken. If the student decides to take a practice test, he/she has to first select the test using "Test Selection" screen and then practice test questions start appearing one by one (Figure 6 of the main thesis text). Spelling mistakes in student's answer text are underlined. Suggestions for correct spelling can be viewed by pressing the "F7" button. Student can then replace the mis-spelled word in the answer text with correct spelling in the list of suggested words. Once the practice test has finished, the student's result for the practice is displayed immediately.

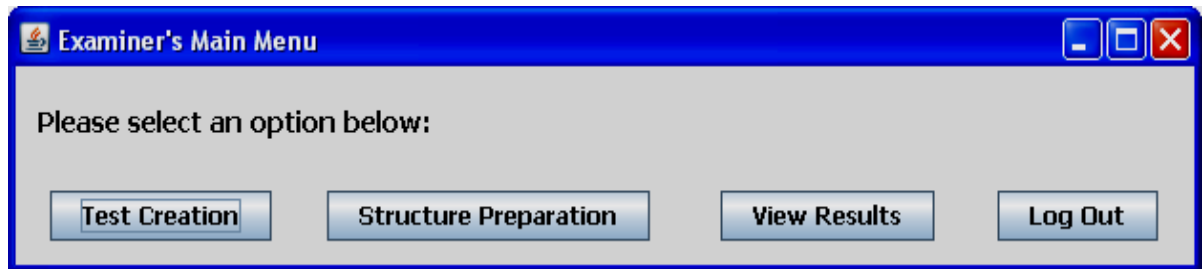
If a student wants to view his/her practice performance history (Appendix 15), he/she should press the "View Results" button on "Student's Main Menu" screen (Appendix 14). The "Practice Performance History" screen (Appendix 15) shows details of all the practice tests taken by the student. The following details are displayed for each practice:

1. Practice number
2. Subject
3. Topics covered
4. Marks obtained by the student
5. Total / maximum marks for the test

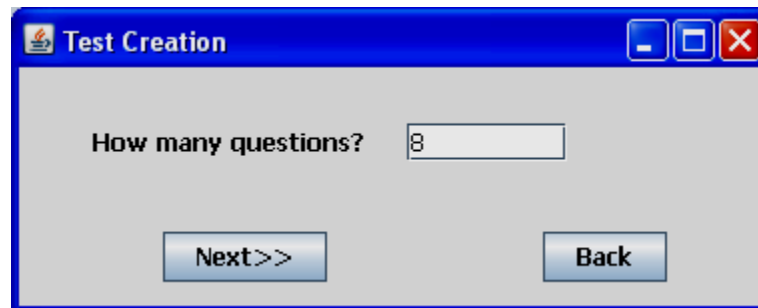
The student can also analyze his/her answers to various questions of the practice test(s) taken. To do this, he/she should press the "Analyze Answers" button on "Practice Performance History" screen (Appendix 15). The student will then be required to enter the practice (test) number (Appendix 16). Once practice number has been specified, a "Practice's Question Selection" screen (Appendix 17) appears on the computer display. A tree-based structure on the left-hand side of the "Practice's Question Selection" screen consists of all the question numbers (and sub-question numbers) of the practice (test). Selecting a question number on the tree-based structure, results in details (such as question text, marks obtained in this question and maximum marks for this question) to be displayed on the right-hand side. Once the desired question has been selected, the "Analyze Answer" button should be pressed to start student's answer text analysis for the selected question.

Figure 9 (of the main thesis text) depicts the IndusMarker's GUI screen for carrying out student's answer text analysis. The student's answer text is displayed in a text area in the upper right side of the screen. The list of required "regions" for the answer is given in the lower part of the GUI screen. The screen enables its user (i.e. the student) to check whether a particular required "region" has been matched in the student's answer text or not. A "region" may be "fully matched", "partially matched" or "not matched" at all in the student's answer text. In order to check whether a particular "region" is matching or not, the student selects a "region" and presses the "Match Region" button. The sentence that matches the most, with the selected "region", is highlighted. Other important information such as "Matching Result", "Marks Obtained" and "Total Marks" can also be viewed on the screen. Once the student has finished analyzing his/her answer, he/she should press the "Finish" button.

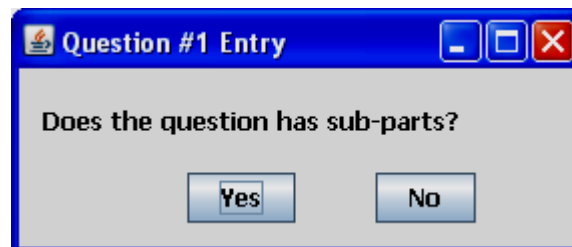
***Appendix 2: Screen Shot of “Examiner’s Main Menu” Screen***



***Appendix 3: Screen Shot of Screen for Asking Examiner to Specify the Number of Questions in the Test***



***Appendix 4: Screen Shot of Screen for Asking Examiner to specify whether or not the Question to be entered has Sub-parts***



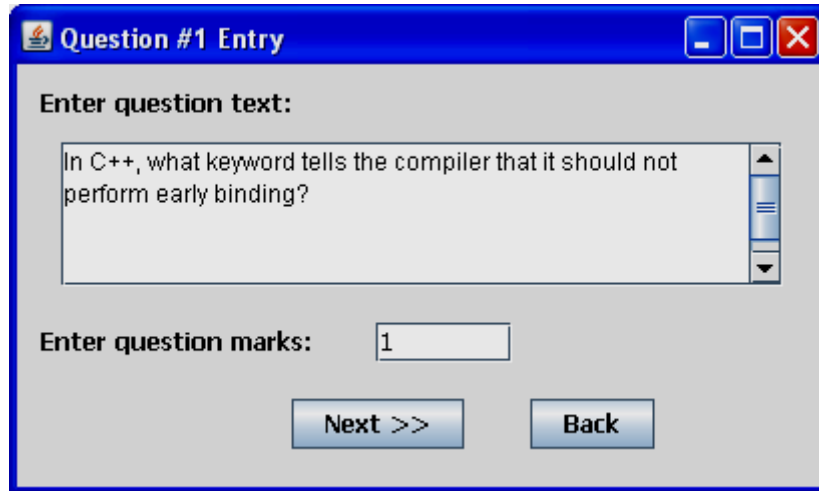
***Appendix 5: Screen Shot of Screen for Entering Main Question Text and Number of Sub-parts***

The screenshot shows a window titled "Question #1 Entry" with a blue title bar. The main content area is light gray. At the top, it says "Enter the main question text:". Below this is a large text input field with the placeholder text "Define (in a single sentence):". To the right of the text field is a vertical scrollbar. Below the text field, it asks "How many sub-parts?" followed by a small text input field containing the number "3". At the bottom of the window, there are two buttons: "Next>>" and "Back".

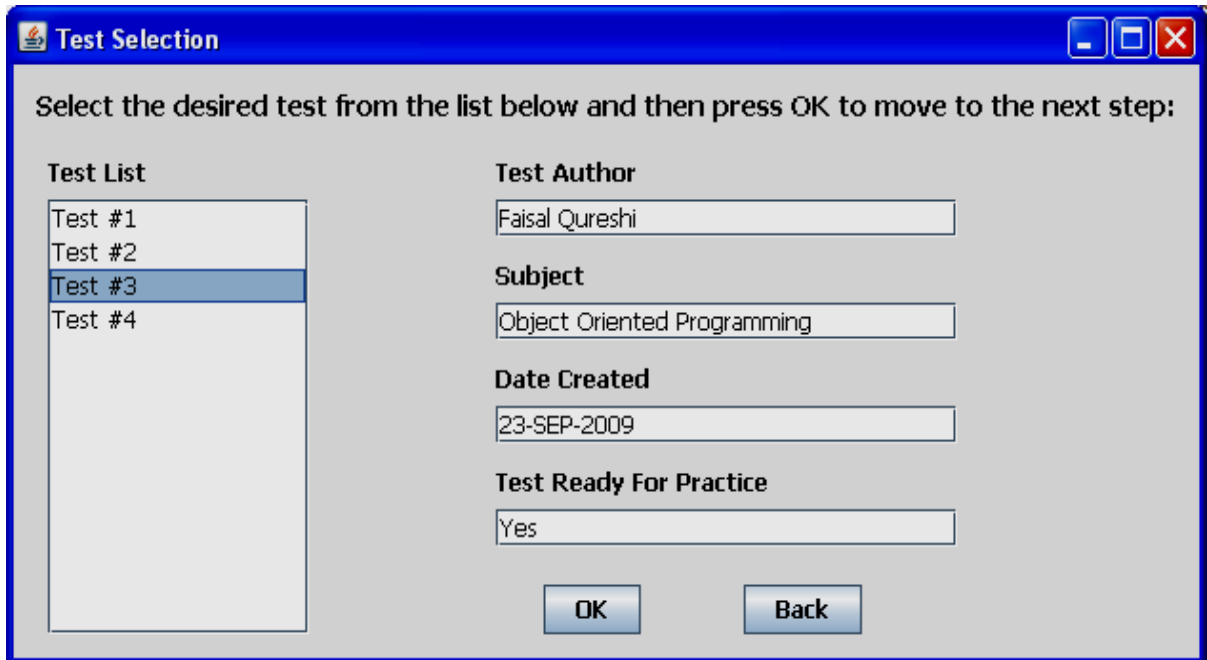
***Appendix 6: Screen Shot of Screen for Sub-part Entry***

The screenshot shows a window titled "Question #1 Entry" with a blue title bar. The main content area is light gray. At the top, it says "Enter sub-part #1:". Below this is a large text input field containing the word "Binding". To the right of the text field is a vertical scrollbar. Below the text field, it asks "Enter sub-part marks:" followed by a small text input field containing the number "1". At the bottom of the window, there are two buttons: "Next>>" and "Back".

**Appendix 7: Screen Shot of Screen for Entering Question Text and Question Marks**

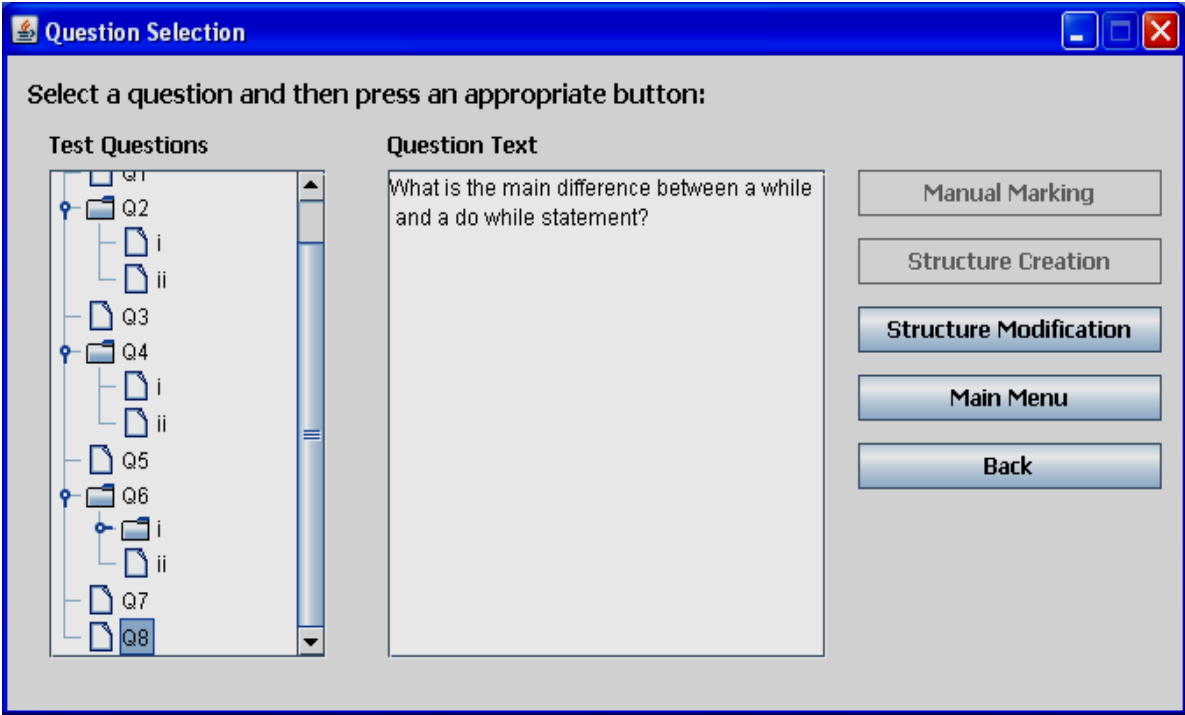


**Appendix 8: Screen Shot of "Test Selection" Screen**





**Appendix 9: Screen Shot of "Question Selection" Screen**



## Appendix 10: Screen Shot of “Answer Structure” Formulation & Validation Screen

**"Answer Structure" Formulation & Validation**

Question: How many constructors can be created for a class?

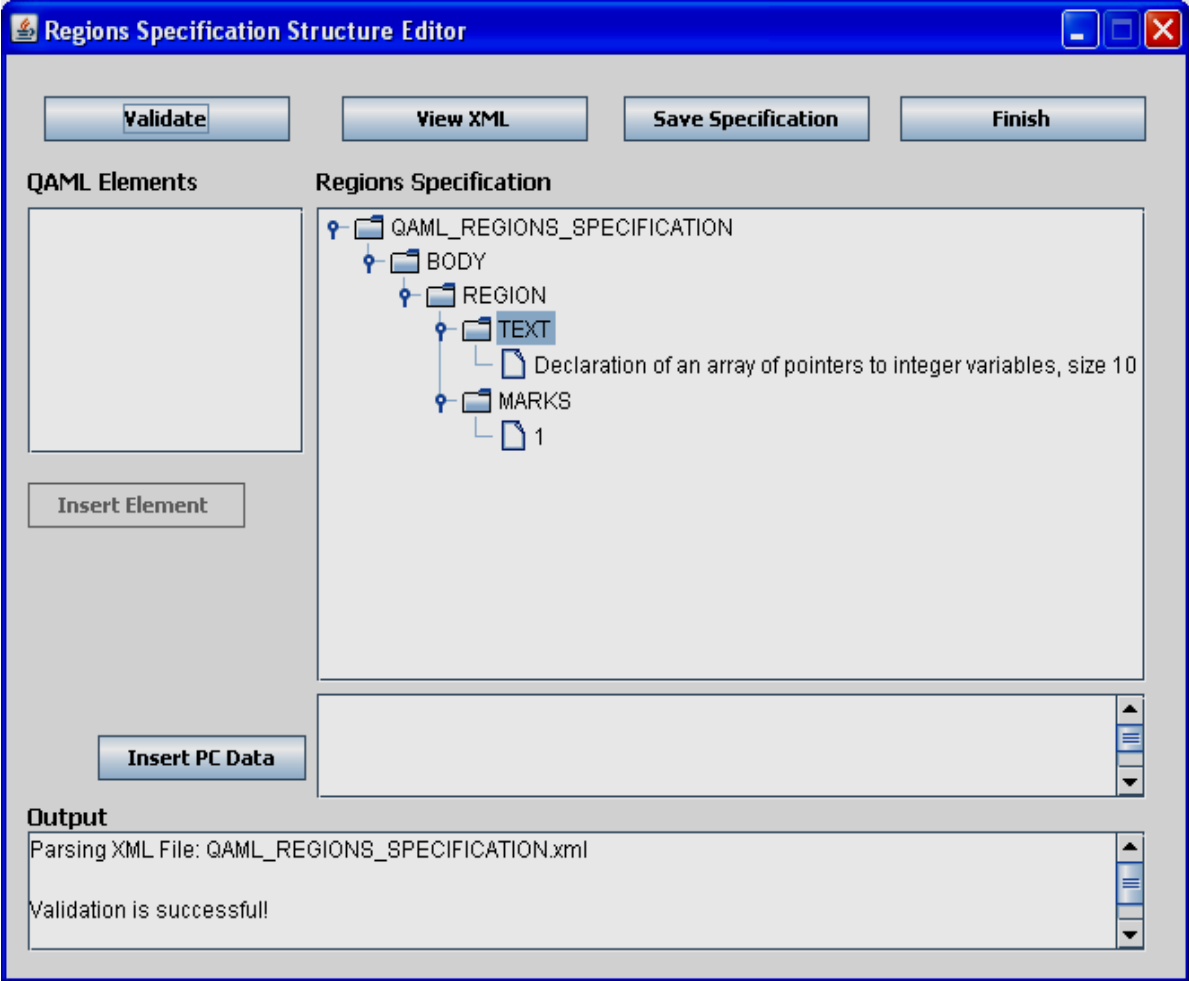
Formulate Structure      Validate Structure      Correct Structure

| No. | Students' Answers                                                                  | Manual Marking | Automated Marking | Discrepancy                         |
|-----|------------------------------------------------------------------------------------|----------------|-------------------|-------------------------------------|
| 1   | For a single class unlimited number fo constructors can be created                 | 1              | 1                 | <input type="checkbox"/>            |
| 2   | one class can be created for each class                                            | 0              | 0                 | <input type="checkbox"/>            |
| 3   | u can have as many constructors as you want but you must have different paramet... | 1              | 1                 | <input type="checkbox"/>            |
| 4   | number of constructors depends on the programmer                                   | 1              | 0                 | <input checked="" type="checkbox"/> |
| 5   | unlimited constructors                                                             | 1              | 1                 | <input type="checkbox"/>            |
| 6   |                                                                                    | 0              | 0                 | <input type="checkbox"/>            |
| 7   | as many as required                                                                | 1              | 1                 | <input type="checkbox"/>            |
| 8   | as many as needed                                                                  | 1              | 1                 | <input type="checkbox"/>            |
| 9   | unlimited constructors                                                             | 1              | 1                 | <input type="checkbox"/>            |
| 10  |                                                                                    | 0              | 0                 | <input type="checkbox"/>            |
| 11  | one                                                                                | 0              | 0                 | <input type="checkbox"/>            |
| 12  | As many as we want                                                                 | 1              | 1                 | <input type="checkbox"/>            |
| 13  | as many as required                                                                | 1              | 1                 | <input type="checkbox"/>            |
| 14  | as many as required                                                                | 1              | 1                 | <input type="checkbox"/>            |
| 15  | multiple constructors can be created for a class                                   | 0              | 0                 | <input type="checkbox"/>            |
| 16  | As many as we want                                                                 | 1              | 1                 | <input type="checkbox"/>            |
| 17  | as many as required                                                                | 1              | 1                 | <input type="checkbox"/>            |
| 18  | as many as required                                                                | 1              | 1                 | <input type="checkbox"/>            |
| 19  |                                                                                    | 0              | 0                 | <input type="checkbox"/>            |
| 20  | as many as required                                                                | 1              | 1                 | <input type="checkbox"/>            |

Full Text: u can have as many constructors as you want but you must have different parameter list for each

Finish      Main Menu      Back

**Appendix 11: Screen Shot of “Regions Specification Structure Editor”  
Screen**



**Appendix 12: Screen Shot of "Practice Results" Screen**

The screenshot shows a window titled "Practice Results". It contains a "Practice list" on the left with "Practice #1" selected. On the right, there are several input fields with the following values:

- Teacher: Javeria Shamim
- Subject: Object Oriented Programming
- Topic(s) Covered: \_oops, pointers, arrays and structures
- Date: 18-SEP-2009
- Campus: Islamabad Campus
- Institution: Bahria University
- Average Marks: 7.665
- Maximum Marks: 12

At the bottom, there are three buttons: "Students' Details", "Main Menu", and "Back".

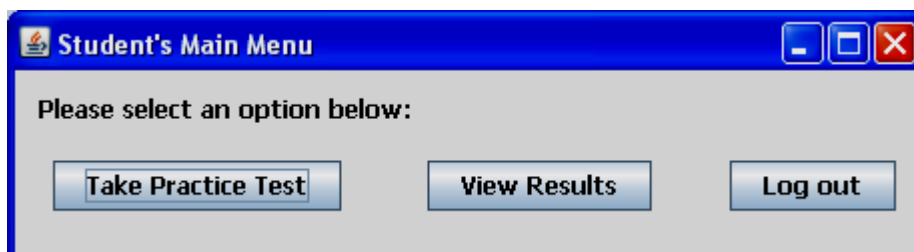
**Appendix 13: Screen Shot of "Individual Student's Performance" Screen**

The screenshot shows a window titled "Individual Student's Performance". It displays a table titled "Students' performance in practice #1 (Maximum Marks: 12):". The table has three columns: "Reg. #", "Student Name", and "Marks Obtained".

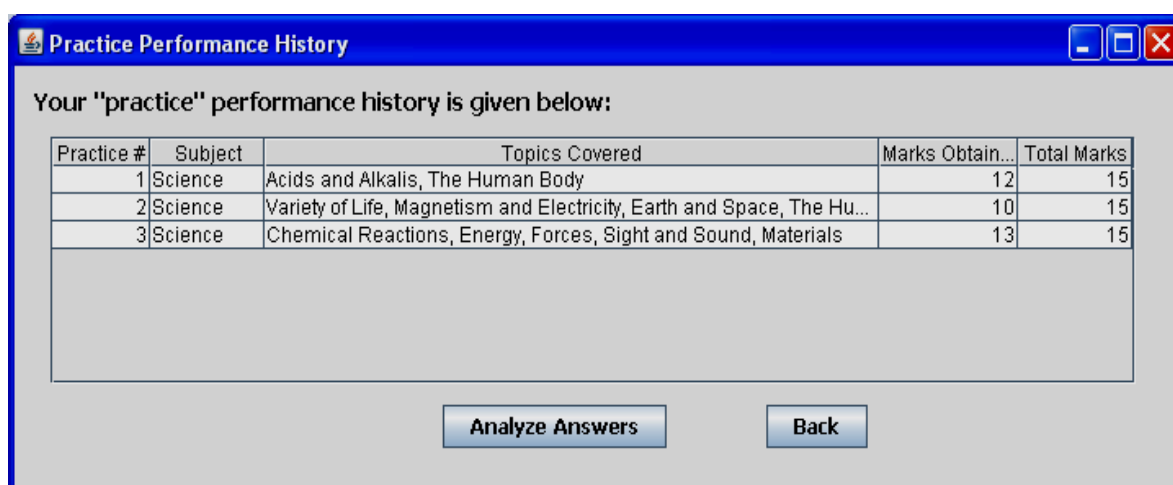
| Reg. #  | Student Name      | Marks Obtained |
|---------|-------------------|----------------|
| CE-3543 | Romaila Abbasi    | 9              |
| CE-3544 | Sadia Naz         | 5              |
| CE-3545 | Javeria Aqeel     | 6              |
| CE-3546 | Anees Ahmed       | 11             |
| CE-3547 | Gul Sher          | 9              |
| CE-3548 | Saba Karim        | 4              |
| CE-3549 | Haider Abbas      | 8              |
| CE-3550 | Rizwan Iqbal      | 6.5            |
| CE-3551 | Zafar Khan        | 10             |
| CE-3552 | Muhammad Farhan   | 8              |
| CE-3553 | Ameer Illahi      | 7.5            |
| CE-3554 | Shoib Kamran      | 10             |
| CE-3555 | Muhammad Baber    | 4              |
| CE-3556 | Kashan Masud      | 3              |
| CE-3557 | Zulfikar Ali Khan | 6              |

At the bottom, there are two buttons: "Main Menu" and "Back".

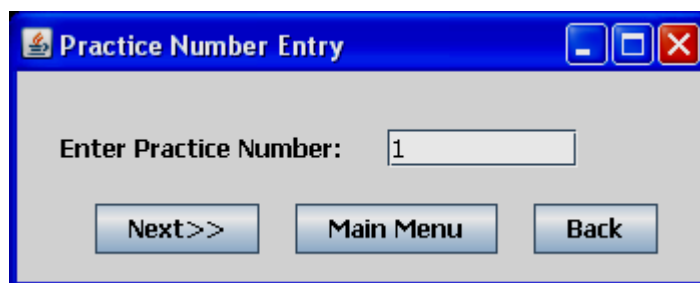
**Appendix 14: Screen Shot of “Student’s Main Menu” Screen**



**Appendix 15: Screen Shot of “Practice Performance History” Screen**



**Appendix 16: Screen Shot of “Practice Number Entry” Screen**



*Appendix 17: Screen Shot of "Practice's Question Selection" Screen*

