

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**



**FEUP**

# **Computer-Based Assessment System for e-Learning applied to Programming Education**

**Pedro Xavier Pacheco**

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: António Fernando Coelho (Professor Auxiliar)

July 2010



# **Computer-Based Assessment System for e-Learning applied to Programming Education**

**Pedro Xavier Pacheco**

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: João Pascoal Faria (Professor Auxiliar)

External Examiner: Luís Borges Gouveia (Professor Associado com agregação)

Supervisor: António Fernando Coelho (Professor Auxiliar)

---

22<sup>nd</sup> July, 2010



# Abstract

E-learning systems have been widely adopted in Universities, helping to diminish the distance between teaching staff and students and allowing a better monitoring of the learning process. The impact of these systems can be quite significant in programming courses. It is essential for programming students to solve a considerable amount of problems and assignments in order to become familiar with the concepts involved in the complex craft of programming. The combination of this high amount of exercises with a high number of students is leading to staff work overload in many academic institutions. Nevertheless, the process of assessing programming assignments can be automated by using a specific kind of e-Learning systems, the Computer-Based Assessment (CBA) systems.

This dissertation proposes a new CBA system for supporting the programming components of the courses of DEI, FEUP. The system was specified to have a core component, behaving as an independent service, responsible for the automatic assessment of programming assignments and flexible enough to be integrated with other platforms such as Moodle, SIGEX and an eBook system.

In order to achieve the proposed goals, a number of steps were followed and the results are detailed in this document. A study of state-of-the-art CBA systems was performed in order to understand the types of available features. Then, it was analyzed how these systems are currently being used to support programming courses in academic institutions and what best-practices emerge from their use. Knowing what features are usually available in CBA systems, an online survey was built to collect the opinions and needs of DEI teaching staff. The system specification was then built, taking in consideration the survey results. It includes a prioritized list of features, as well as the system's architecture. During the specification phase it was also decided to use DOMjudge, a CBA system used in programming contest, as the basis for the new system. Then, a prototype of the system was implemented and the results of the project were evaluated.

During the results evaluation, it was concluded that the developed prototype, which was configured in a test server and can be freely used, implements a significant amount of the specified features. The system has two innovative mechanisms not found in other CBA systems: the possibility of defining test cases with attributes, such as personalized feedback messages, and a skeleton file mechanism. However, in order to become a mature system, further development and validation is still needed.



# Resumo

Os sistemas de e-Learning têm sido adoptados em Universidades, ajudando a vencer a barreira da distância entre docentes e estudantes e permitindo um melhor acompanhamento do processo de aprendizagem. O impacto destes sistemas pode ser bastante significativo em unidades curriculares de programação. É essencial que os estudantes de programação resolvam uma quantidade assinalável de exercícios de forma a se familiarizarem com os conceitos envolvidos. A combinação deste número elevado de exercícios com o elevado número de estudantes, tem levado a uma sobrecarga dos docentes em muitas instituições académicas. Contudo, o processo de correcção de exercícios de programação pode ser automatizado através da utilização de um tipo específico de sistemas de e-Learning: os sistemas de avaliação automática.

Esta dissertação propõe um novo sistema de avaliação automática para suporte às componentes de programação das unidades curriculares do DEI, FEUP. O sistema foi especificado de forma a conter uma componente central, funcionando como um serviço independente, responsável pela avaliação automática de exercícios de programação e suficientemente flexível para ser integrada com outras plataformas como o Moodle, SIGEX e um sistema de eBooks.

De forma a alcançar os objectivos propostos, uma série de passos foi seguida e os seus resultados encontram-se detalhados neste documento. Foi efectuado um estudo dos sistemas de avaliação automática correntemente em uso, de forma a perceber quais as funcionalidades disponibilizadas. De seguida, analisou-se a forma como estes sistemas têm sido utilizados em instituições académicas e quais as boas-práticas que emergem desse uso. Sabendo quais as funcionalidades habitualmente disponíveis neste tipo de sistemas, criou-se um inquérito online, para recolher as opiniões e necessidades dos docentes do DEI. Tendo em consideração os resultados do inquérito, desenvolveu-se a especificação do sistema. Esta inclui uma lista prioritizada das funcionalidades, assim como a arquitectura do sistema. Durante a fase de especificação, foi também decidido utilizar o DOMjudge, um sistema utilizado em concursos de programação, como a base para o novo sistema. Seguidamente, um protótipo foi implementado e os resultados do projecto foram avaliados.

Na avaliação de resultados, concluiu-se que o protótipo desenvolvido, que foi configurado num servidor de testes e pode ser livremente utilizado, implementa um conjunto significativo de funcionalidades. O sistema possui dois mecanismos inovadores, que não foram encontrados noutros sistemas de avaliação automática: a possibilidade de definir casos de teste com atributos, tais como mensagens personalizadas de feedback, e um mecanismo de código esqueleto. No entanto, ainda são necessários mais avanços em termos de implementação e validação.





# Acknowledgements

I would like to thank everyone who contributed in some way to this dissertation.

First of all, I would like to thank my supervisor, Professor António Fernando Coelho, for all enthusiasm, efforts and good ideas. Thank you for all the useful guidelines for keeping this project on track.

Special thanks go to Professor Fernando Nunes Ferreira, António Bandeira and Tito Vieira for all the suggestions concerning the final system specification.

I would like to add further thanks to António Bandeira for his suggestions about the system's architecture and communication protocol.

Thank you to all CICA's staff involved in creating the virtual machines used for setting up a test server for the new CBA system.

Special gratitude goes to my parents and sister for all their support, love and care. Thank you for creating an environment which made my life easier.

And last but not least, I would like to thank everyone who has made this semester such a fantastic one. Thank you for all the great chats, lunches, afternoon snacks, dinners, nights and amazing events full of cheer. I won't list any name, because I could forget someone and you folks know who you are ;)

Pedro Pacheco



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement . . . . .	2
1.2	Goals . . . . .	2
1.3	Methodology . . . . .	2
1.4	Document Structure . . . . .	3
<b>2</b>	<b>CBA Systems</b>	<b>5</b>
2.1	CourseMarker . . . . .	6
2.2	BOSS . . . . .	10
2.3	xLx . . . . .	13
2.4	Mooshak . . . . .	16
2.5	RoboProf . . . . .	18
2.6	Submit! . . . . .	19
2.7	GAME . . . . .	22
2.8	DOMjudge . . . . .	24
2.9	Summary of Features . . . . .	26
<b>3</b>	<b>Application of CBA Systems in e-Learning</b>	<b>35</b>
3.1	Case Studies Analysis . . . . .	35
<b>4</b>	<b>Analysis of Learning Needs</b>	<b>45</b>
4.1	Survey Analysis . . . . .	45
<b>5</b>	<b>Specification of a New CBA System</b>	<b>55</b>
5.1	New System Proposal . . . . .	55
5.2	Reusing an Existing System . . . . .	62
5.3	Integration With Moodle . . . . .	65
5.4	From Programming Contests to University Assignments . . . . .	68
5.5	Architecture . . . . .	69
<b>6</b>	<b>Implementation</b>	<b>73</b>
6.1	Entity Mapping . . . . .	74
6.2	Front End and Communication Protocol . . . . .	77
6.3	Extending DOMjudge's Features . . . . .	77
6.4	Polling of Automatic Assessment Results . . . . .	77
6.5	Test Cases Configuration and Personalized Feedback . . . . .	78
6.6	Skeleton File Mechanism . . . . .	79

## CONTENTS

6.7	Feedback System . . . . .	82
<b>7</b>	<b>Results</b>	<b>83</b>
7.1	Implemented Features . . . . .	83
7.2	Prototype Validation . . . . .	86
7.3	Summary . . . . .	87
<b>8</b>	<b>Conclusions and Future Work</b>	<b>89</b>
8.1	Work Summary . . . . .	89
8.2	Future Work . . . . .	91
	<b>References</b>	<b>93</b>
<b>A</b>	<b>Survey</b>	<b>99</b>
<b>B</b>	<b>Installation Guide</b>	<b>107</b>
B.1	Installing the Main Automatic Assessment Server . . . . .	107
B.2	Installing a Judgehost . . . . .	108
B.3	Installing the Moodle Server . . . . .	109
<b>C</b>	<b>Programming Language Configuration Guide</b>	<b>111</b>
C.1	DOMjudge Configuration . . . . .	111
C.2	Moodle Configuration . . . . .	112
<b>D</b>	<b>Assessment Creation Guide</b>	<b>115</b>
D.1	Accessing the Test Server . . . . .	115
D.2	Assessment with Skeleton Code . . . . .	116
D.3	Assessment without Skeleton Code . . . . .	121
<b>E</b>	<b>A Scheme Assessment</b>	<b>125</b>
E.1	Approach 1 . . . . .	126
E.2	Approach 2 . . . . .	127
<b>F</b>	<b>Solution Submission Guide</b>	<b>129</b>
<b>G</b>	<b>Communication Protocol</b>	<b>135</b>

# List of Figures

2.1	CourseMarker architecture . . . . .	8
2.2	CourseMarker main student interface . . . . .	9
2.3	CourseMarker student interface - representation of marks and feedback . . . . .	9
2.4	BOSS architecture . . . . .	12
2.5	BOSS student interface - Java client . . . . .	12
2.6	BOSS student interface - web client . . . . .	12
2.7	xLx interface - GUI for a tutor to configure new exercises . . . . .	15
2.8	xLx interface - Annotation window from the student's point of view . . . . .	15
2.9	Mooshak graphical interface - judges' view . . . . .	18
2.10	Submit! graphical interface - basic assessment configuration screen . . . . .	21
2.11	GAME user interface . . . . .	23
4.1	Curricular years of the positive answers' courses . . . . .	47
4.2	Study areas of the positive answers' courses . . . . .	47
5.1	Use cases overview . . . . .	56
5.2	Assessments Management use cases . . . . .	57
5.3	Assessments Solving use cases . . . . .	57
5.4	Submissions Management use cases . . . . .	58
5.5	DOMjudge's architecture . . . . .	70
5.6	Overall architecture . . . . .	71
6.1	DOMjudge main entities . . . . .	75
6.2	Moodle plugin main entities . . . . .	76
A.1	Survey page 1 - Initial page . . . . .	99
A.2	Survey page 2 - Creation of new assessments . . . . .	100
A.3	Survey page 3 - Test case definition and automatic assessment . . . . .	100
A.3	Survey page 3 - Test case definition and automatic assessment (cont) . . . . .	101
A.4	Survey page 4 - Creation and submission of solutions . . . . .	102
A.5	Survey page 5 - Feedback system . . . . .	102
A.5	Survey page 5 - Feedback system (cont) . . . . .	103
A.6	Survey page 6 - Submissions management . . . . .	103
A.6	Survey page 6 - Submissions management (cont) . . . . .	104
A.7	Survey page 7 - Extra features . . . . .	104
A.7	Survey page 7 - Extra features (cont) . . . . .	105
A.8	Survey page 8 - Final page . . . . .	105

## LIST OF FIGURES

B.1	Setting up the contest table . . . . .	108
B.2	Adding a new judgehost to DOMjudge . . . . .	109
C.1	Programming Language Configuration - Step 2 . . . . .	112
D.1	Programming assessment general settings . . . . .	116
D.2	Programming assessment grading settings . . . . .	117
D.3	Programming assessment uploading of files settings . . . . .	118
D.4	Programming assessment programming languages settings . . . . .	118
D.5	Programming assessment skeleton code . . . . .	118
D.6	Programming assessment feedback settings . . . . .	119
D.7	Programming assessment test cases . . . . .	119
D.8	Programming assessment description . . . . .	121
D.9	Programming assessment description - sample input and output . . . . .	122
D.10	Programming assessment grading settings . . . . .	122
D.11	Programming assessment uploading of files settings . . . . .	123
D.12	Programming assessment programming languages settings . . . . .	123
D.13	Programming assessment skeleton code . . . . .	123
D.14	Programming assessment feedback settings . . . . .	123
D.15	Programming assessment test cases . . . . .	124
F.1	Solution submission general view . . . . .	129
F.2	Compilation playground . . . . .	130
F.3	Compilation playground - successful compilation . . . . .	130
F.4	Solution submission . . . . .	131
F.5	Submission feedback . . . . .	132
F.6	Input and output data . . . . .	133

# List of Tables

2.1	Types of assessments supported by CBA systems . . . . .	27
2.2	Methodologies for test case definition available in CBA systems . . . . .	28
2.3	Environment for Solutions Development . . . . .	29
2.4	Support for Skeleton Solution Mechanism . . . . .	30
2.5	Programming Languages Supported by the CBA systems . . . . .	32
2.6	Type of interface of CBA systems . . . . .	33
3.1	CBA systems used in different institutions . . . . .	35
4.1	Course information of positive answers . . . . .	46
4.2	Usefulness of the different assessment types . . . . .	48
4.3	Parameters for assessment configuration . . . . .	48
4.4	Votes on the different test-cases definition methodologies . . . . .	48
4.5	Importance of different test-case parameters . . . . .	49
4.6	Votes on the solution development alternatives . . . . .	49
4.7	Usefulness of a mechanism for definition of skeleton solutions . . . . .	50
4.8	Votes on the different levels of feedback . . . . .	50
4.9	Usefulness of a mechanism for personalized feedback messages . . . . .	51
4.10	Usefulness of features related to the management of solutions . . . . .	51
4.11	Usefulness of some suggested extra features . . . . .	52
5.1	Types of supported assessments . . . . .	59
5.2	Parameters for assessment configuration . . . . .	59
5.3	Methodologies for test-cases definition . . . . .	59
5.4	Parameters for test-case configuration . . . . .	60
5.5	Features for supporting the elaboration and submission of solutions . . . . .	60
5.6	Features related to the feedback system . . . . .	61
5.7	Features related to the management of submissions . . . . .	61
5.8	Extra features . . . . .	62
5.9	Features with high priority . . . . .	62
5.10	Features with medium priority . . . . .	63
5.11	Features with low priority . . . . .	63
5.12	Implementation freedom of the different plugin alternatives . . . . .	65
5.13	Implementation effort of the different plugin alternatives . . . . .	66
5.14	Number of pre-implemented configuration parameters for the different plugin alternatives . . . . .	66
5.15	Availability of a pre-implemented mechanism for upload of solutions in the different plugin alternatives . . . . .	66

## LIST OF TABLES

5.16	Vulnerability to future Moodle structural changes of the different plugin alternatives . . . . .	67
5.17	Means for reusing problems of the different plugin alternatives . . . . .	67
7.1	Implementation of the types of supported assessments . . . . .	84
7.2	Implementation of the parameters for assessment configuration . . . . .	84
7.3	Implementation of the methodologies for test-cases definition . . . . .	84
7.4	Implementation of the parameters for test-case configuration . . . . .	85
7.5	Implementation of the features for supporting the elaboration and submission of solutions . . . . .	85
7.6	Implementation of the features related to the feedback system . . . . .	85
7.7	Implementation of the features related to the management of submissions	86
7.8	Implementation of the extra features . . . . .	86



# Abbreviations

ACM	Association for Computing Machinery
API	Application Programming Interface
CBA	Computer-Based Assessment
CGI	Common Gateway Interface
CICA	Prof. Correia de Araujo Computer Centre
CPU	Central Processing Unit
DEI	Department of Informatics Engineering
FEUP	Faculty of Engineering of the University of Porto
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICPC	International Collegiate Programming Contest
ICT	Information and Communication Technology
IDE	Integrated Development Environment
MIEEC	Master in Electrical and Computers Engineering
MIEIC	Master in Informatics and Computing Engineering
RMI	Remote Method Invocation
SOAP	Simple Object Access Protocol
UML	Unified Modeling Language
SQL	Structured Query Language
SSL	Secure Sockets Layer
VPN	Virtual Private Network
WSDL	Web Service Definition Language
WWW	World Wide Web
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

## ABBREVIATIONS

# Chapter 1

## Introduction

E-Learning is defined as all forms of electronic supported learning and teaching, which are procedural in character and aim to effect the construction of knowledge with reference to individual experience, practice and knowledge of the learner. Information and communication systems, whether networked or not, serve as specific media to support the learning process [TLN<sup>+</sup>04]. An e-Learning system is a comprehensive software package that supports courses that depend on the WWW for some combination of delivery, testing, simulation, discussion, or other significant aspect [Rob99]. Examples of such systems are the Blackboard Learning System, Claroline, Dokeos, HotChalk, Moodle and Sakai. These systems have been widely adopted in universities, helping to diminish the distance between teaching staff and students and allowing a better monitoring of the learning process.

Programming problems and assignments are considered essential elements of software engineering and computer science education. They can help students become familiar with the attributes of modern programming languages, become acquainted with essential tools, and to understand how the principles of software development and design can be applied [DLO05]. In addition, a large number of students are enrolling in introductory programming courses annually. As a consequence, teaching staff in many universities and institutes are facing a great number of assignments that require marking and grading [MKKN05].

Automation of the assessment process can be achieved by using a specific kind of e-Learning systems, the Computer Based Assessment (CBA) systems. Moreover, these systems are a good way of creating new exercises for students and, at the same time, provide them with more and more accurate feedback, when compared with traditional teaching methods [CAMF<sup>+</sup>03]. This combination of factors is leading to a wide acceptance of CBA systems among teaching staff at many universities.

## 1.1 Problem Statement

For the last years, a simple CBA system has been used in a first-year programming course, where the functional language Scheme is taught. Despite its success, it has some limitations in terms of the supported functionalities and flexibility and hence the creation of a new system is desirable. This new system should be extensible in order to easily incorporate the introduction of new programming languages.

## 1.2 Goals

The goals of this dissertation are to specify and to develop a prototype of a CBA system for supporting the programming components of the courses of the Departamento de Engenharia Informatica (DEI - Department of Informatics Engineering) of the Faculdade de Engenharia da Universidade do Porto (FEUP - Faculty of Engineering of the University of Porto). The core component of the CBA system should behave as an independent service and it should be possible to integrate it with other platforms such as Moodle, the e-Learning platform used at the Faculty.

## 1.3 Methodology

A number of procedures were followed in order to produce a complete specification and to implement a prototype of the CBA system:

- **Study of the existing CBA systems** - a study of the existing CBA systems was performed with the objective of understanding what features are available in current state-of-the-art systems;
- **Evaluation of the needs DEI teaching staff** - this evaluation was performed with an online survey with the goal of understanding what courses may benefit with the CBA system and what functionalities are considered to be the most important;
- **Analysis of the system requirements** - using the answers to the online survey, the system requirements were analyzed;
- **Specification of the new CBA system** - having a description of the requirements of the system, its functional specification was documented;
- **Prototype development** - within the available time, a functional prototype of the system was implemented;
- **Evaluation of results** - finally, the results achieved by the system specification and by the prototype were assessed.

## 1.4 Document Structure

This document is structured in 8 chapters, being the first one composed by this introduction.

Chapter 2 gives an overview of the features available in eight of the CBA systems currently in use. These features are divided in topics for a better comprehension. The chapter ends with a summary that compares the features of the different systems.

The third chapter presents how CBA systems have been used to support the learning process in different Universities. Some case studies, from where some best practices were extracted, are discussed.

The fourth chapter consists of the analysis, supported by an online survey, of DEI's teaching staff learning needs. The survey structure was based on the features usually available on CBA systems.

Chapter 5 consists of the specification of the new CBA system, based on the requirements description. A proposal, including the features that should be supported, is introduced. Then, a study about reusing one of the studied CBA systems is presented and a decision is made about the type of Moodle plugin to use. The chapter ends with a discussion about the system's architecture.

The sixth chapter is about the prototype implementation. It contains the most relevant implementation details, divided in seven main topics.

The evaluation of the results of achieved by this project is presented in chapter 7. The implemented features are listed, the validation of the system is discussed and some general considerations about the system features are made.

The last chapter contains some conclusions about the project as well as future work perspectives.

Some appendixes, regarding the configuration and usage of the system, were included at the end of the document.

## Introduction

## Chapter 2

# CBA Systems

CBA systems seem to have existed for as long as educators have asked students to build their own software [DLO05]. Probably, the earliest example of such a system is the one developed by Hollingsworth for assessing programs written in assembly language on punched cards [Hol60]. Between the decades of 1980 and 1990, a number of CBA systems based on command-line interfaces and manual operation of scripts were developed [DLO05]. More recent systems such as CourseMarker, BOSS, TRAKLA and RoboProf make use of the recent developments in web technology, graphical interfaces and adopt increasingly sophisticated assessment approaches. A pretty recent project, the EduJudge project, is taking the concept of CBA systems one step further. It has the objective of integrating the University of Valladolid (UVa) Online Judge [uva10] into an effective educational environment, in order to satisfy the users demand of a greater pedagogic character and, in this way, facilitate the use of the Online Judge as one more activity for official courses in several institutions [edu10, RML08].

This chapter gives an overview of the features available in some of the CBA systems currently in use. If enough information available in existing literature, the following topics are presented for each system.

- **The Automatic Assessment** - description of how the solutions for exercises are assessed;
- **Setup of New Exercises** - explanation of the workflow that course tutors need to follow for setting up new exercises;
- **Solving Exercises** - explanation of the workflow that students need to follow for solving exercises;
- **Feedback System** - description of how (and how detailed) is the automatic feedback given to students;

- **Management of Submissions** - enumeration of the functionalities available to the course tutors for management and analysis of the students' submissions;
- **Architecture** - explanation of the architecture of the system: modules, relations between them and different responsibilities;
- **Graphical Interface** - description of the graphical user interface;
- **Security and Reliability** - presentation of the existing mechanisms for assuring system's security and reliability;
- **Extensibility** - description of how to customize/extend the system features;

After the description of the different systems, a set of tables is used to summarize and compare the available features.

## 2.1 CourseMarker

The CourseMarker CBA system, developed at the University of Nottingham and first used in 1998, was created as the replacement of the Ceilidh system from the same University [BBFH95, FHST00, FHST01]. Ceilidh was created in 1988 and was based on an automatic assessment mechanism that could test and mark programs using different perspectives: check a program's dynamic correctness, the programming style and layout, among others [BBFH95]. Copies of Ceilidh were taken by over 300 establishments worldwide and installed at many of them around the globe. However, Ceilidh had several limitations and hence CourseMarker was created to overcome them [HGST05]. As of 2005, about 50 universities worldwide have taken copies of CourseMarker for trial and approximately 15 purchased the system for actual use [HGST05].

### 2.1.1 The Automatic Assessment

CourseMarker has a set of marking tools to automatically assess the quality of students' submissions. Three tools are used for assessing programming exercises [HST02]: the `Typographic Tool` checks the program layout, indentation, choice and length of identifiers and usage of comments; the `Dynamic Tool` runs the students' programs with various sets of test data with the intent of verifying whether the students' programs satisfy the stated specification; the `Feature Tool` checks the students' source code for special features that are exercise dependant (for example, an exercise set on a unit that teaches the difference between `while` and `for` statements would typically include a feature test to ensure the appropriate use of each construct).



### 2.1.2 Setup of New Exercises

When setting up a new exercise, the exercise developer has to define a skeleton for the solution. This skeleton may be totally empty, contain a semi or fully functional program or even a flawed program to be debugged. The developer also has to customize the parameters of the different marking tools that are going to be used. In addition, the following files need to be specified:

- *mark.java* - contains Java code that can access CourseMarker's state or call external tools, like compilers, to help with the assessment;
- *properties.txt* - defines parameters for the exercise settings, such as: exercise's filename, maximum number of submissions, whether the exercise is open or closed to students, maximum size of a solution's output, maximum running time, etc;
- *mark.scale* - if required, this file can be used to specify information on how to scale marks.

### 2.1.3 Solving Exercises

The workflow followed by students for solving an exercise can be defined as follows [HST02]:

1. Login into the system and read the exercise description which explains the specifications that the solution has to comply with, and the formats of any input and output files;
2. Obtain a skeleton solution along with header files and/or testing tools;
3. Solve and test the exercise locally, in their workplace;
4. Submit their solution for assessment, by uploading it to the system, and receive marks and feedback. Solutions can be resubmitted as many times as allowed by the exercise developer.

### 2.1.4 Feedback System

CourseMarker has mechanisms for providing automatic and immediate feedback and results. This information is provided in a graphical tree structure which may also contain comments on how to improve the solutions and links for further reading material [HST02]. The amount of feedback can be regulated by the exercise developer to match the needs of the classroom.

### 2.1.5 Management of Submissions

When using CourseMarker, course tutors have access to a set of functionalities related to the management and analysis of students' solutions. Among them are [HHST03, HGST05]: view students' solutions and marks, view statistics of marks for particular exercises, view missing/submitted students, search for and view plagiarism results, expunge students' work.

### 2.1.6 Architecture

CourseMarker was architected with extensibility and maintainability in mind. As can be seen in figure 2.1 it is composed by several subsystems. Each one has different responsibilities [HGST05].

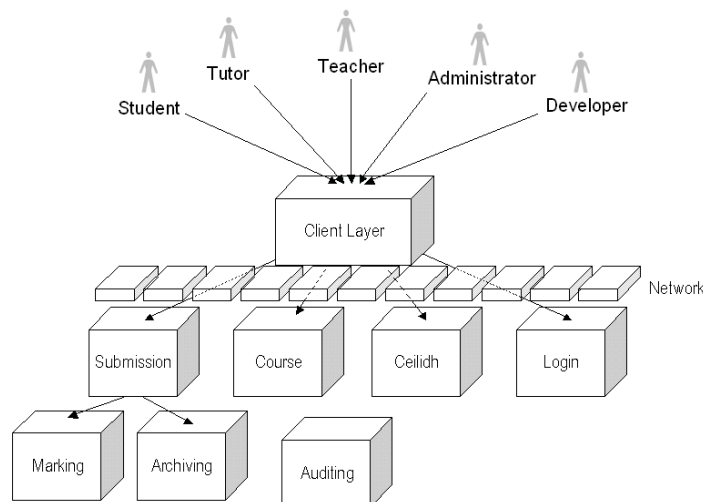


Figure 2.1: CourseMarker architecture

### 2.1.7 Graphical Interface

CourseMarker's graphical user interface was developed with Java Swing. The student's interface is split into three areas (figure 2.2). A row of function buttons that initiate a variety of actions appears along the top. The area on the left side is used as the course view frame. It displays the courses students are registered for and the units and exercises available for viewing and submitting. Information is displayed on the right side, which can consist of course notes, question specifications, or even test data [HGST05]. The students can also access a graphical representation of their marks as well of the feedback received for the different assessments (figure 2.3).

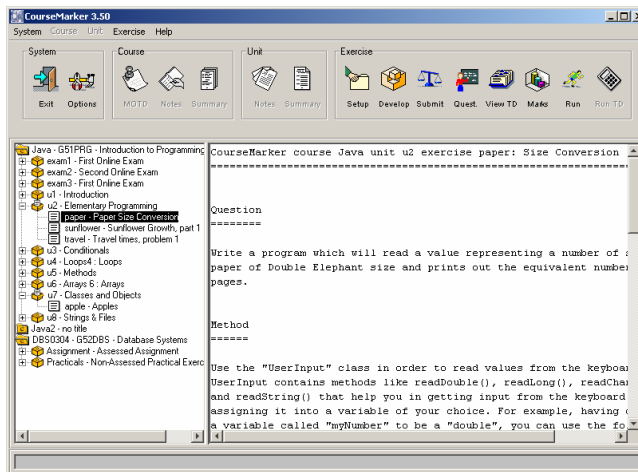


Figure 2.2: CourseMarker main student interface

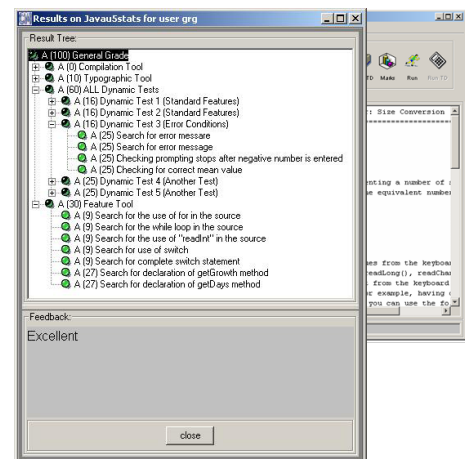


Figure 2.3: CourseMarker student interface - representation of marks and feedback

### 2.1.8 Security and Reliability

CourseMarker was designed and developed with security and reliability in mind. The main reliability problems which may arise are related to corrupted/invalid exercise data, faulty execution of student programs and problems with the marking subsystem. The system has mechanisms for coping with and for logging this kind of problems [HGST05].

In terms of general security, CourseMarker has a password encryption mechanism and a session key security feature. When it comes to the marking system, two security checks are performed [HGST05]: the first identifies potential security risks on students' code, such as "\*,\*", "unlink", "delete" and the use of network sockets/RMI; the second consists in a protected sand-box, with restricted privileges, for running the submissions.

### 2.1.9 Extensibility

CourseMarker was also designed with extensibility in mind, so that new features can be easily added, along with a variety of new types of courses and exercises. The marking process itself can be extended/customized in two different ways [HGST05]. The first one is related with the creation or modification of exercises for types of courses already supported by the system. In this case, exercise developers only need to customize the files referred in section in 2.1.2 and therefore do not need to alter the source code of the system. The second type of customization is related with the customization of the marking server. To introduce new functionalities, like a new kind of course still unsupported, it is necessary to create new marking tools and hence the source code has to be changed.

## 2.2 BOSS

BOSS is a CBA system developed by the University of Warwick. It started in the mid-1990s has a collection of programs without graphical interface [JL95], but soon it was decided that an improved interface was desirable and graphical interfaces were implemented [JL98, JGB05].

Since BOSS is an open source product [War09], the authors do not have accurate information as to which other institutions have deployed it and what local changes they have made to the software [JGB05].

### 2.2.1 The Automatic Assessment

The BOSS system automatically evaluates correctness of submitted solutions by the application of automatic tests. Two methodologies for defining and running the tests can be employed, although the software is structured to allow the incorporation of more in the future [JGB05]. The first methodology consists in the definition of the expected inputs and outputs in data files, while the second methodology consists in the definition of JUnit test. The latter approach applies only when Java is the language used in the submissions. To support the manual marking of subjective attributes related to code quality, BOSS provides a set of program metrics, such as number of comments and percentage of methods declared abstract.

### 2.2.2 Setup of New Exercises

In order to setup new exercises, exercise developers use a data model available in BOSS: the *component model*. This model is intended to support arbitrarily complex assessment structures, composed by one or more problems [JGB05]. Then, for each problem, the developers have to use one of the methodologies, presented in the previous section, for defining the test cases to assess students' solutions. Part of these tests can be made available to students, so they can use them to check their solutions before submitting.

### 2.2.3 Solving Exercises

The workflow followed by students for solving an exercise can be defined as follows:

1. Login into the system and read the exercise description which explains the specifications that the solution has to comply with, and the formats of any input and output files;
2. Solve and test the exercise locally, in their workplace;

3. Submit their solution for assessment, by uploading it to the system. Before submitting, students can run a set of automatic tests (which are a subset of the final tests) on their programs. There is no limit for the maximum number of submissions (within the prescribed deadline).

#### **2.2.4 Feedback System**

When running the set of automatic tests available before submitting their solution, students receive immediate feedback about whether or not their program succeeds in producing the right outputs. However, the feedback of their final mark is not immediate.

After the deadline for the submissions has passed, course staff can run the final automatic tests and mark the solutions. Since it is desirable for students to receive an explanatory result rather than just a number, the system allows the staff to attach written feedback to each submission. The final results and feedback are then e-mailed by the system to the students [JGB05].

#### **2.2.5 Management of Submissions**

The BOSS software permits staff to perform five principle tasks [JGB05]:

- Run automatic tests on the students' submissions;
- Use the plagiarism detection software to detect plagiarism in the solutions;
- Mark submissions, by viewing the results of the automatic tests, running the submitted programs and viewing the source code;
- Authorized staff can moderate the marks given to students' by other markers;
- Associate manual feedback to students' submissions.

Some other features are available from BOSS' interface [JGB05]: consult student details, penalize late submissions, save submission, submit on the behalf of a student, publish marks, etc.

#### **2.2.6 Architecture**

Figure 2.4 presents the architecture of the system. Technologies like SSL and RMI are used to support the communication between the different components [JGB05].

## CBA Systems

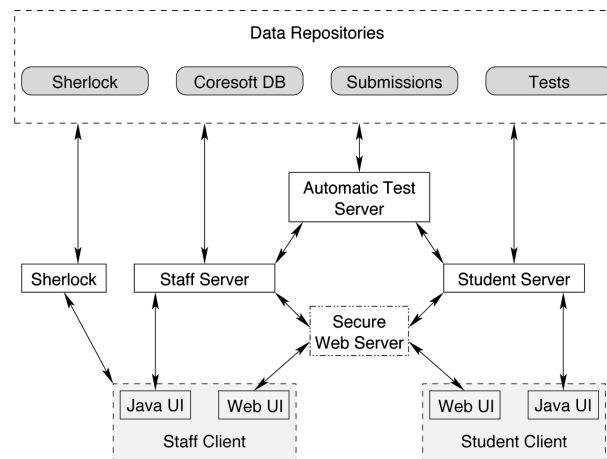


Figure 2.4: BOSS architecture

### 2.2.7 Graphical Interface

BOSS has two kinds of interfaces available: one developed with Java Swing, to be used mainly on machines connected to the campus network; the second is a web-based solution, to be used at home [JGB05]. Figures 2.5 and 2.6 present screenshots of the student interface of, respectively, the Java client and the web client.

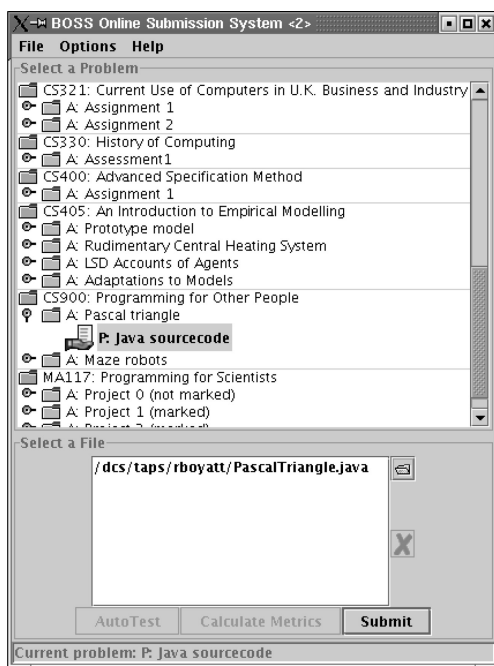


Figure 2.5: BOSS student interface - Java client

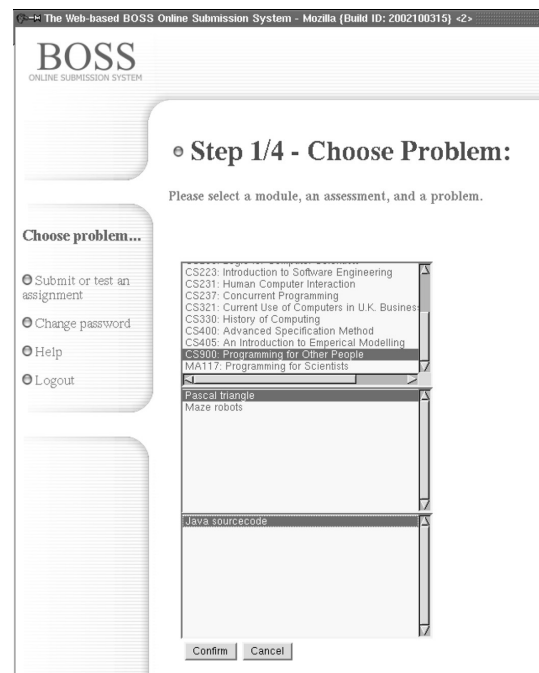


Figure 2.6: BOSS student interface - web client

### 2.2.8 Security and Reliability

BOSS has security mechanisms to identify the student using the software, to assure the integrity of submitted files and to protect data stored on the system from unauthorized access.

In order to protect the system from unsafe or malicious code, the automatic tests are executed inside a sandbox able to limit the amount of available CPU time and memory and with strictly controlled set of permissions [JGB05].

### 2.2.9 Extensibility

In terms of extensibility, it is known that BOSS is prepared to easily incorporate new methodologies for defining and running test cases [JGB05]. Although no more information was found about the ease of extending other parts of the system, it is important to note that, being an open source project, BOSS can be extended and customized by any institution.

## 2.3 xLx

The eXtreme e-Learning eXperience (xLx) system is a web based online learning platform developed at the University of Muenster and can either be used in university or commercial contexts. Its main objective is to support the exercise portion of technically oriented university courses [SVW06]. Initially, it supported the training of skills related to techniques like SQL queries, object-relational features of SQL and transformation of XML documents with XSLT or XQuery [HLVW02]. Later on, the possibility of automatic assessment of Java programming exercises was also incorporated [SVW06]. Quite recently, xLx was redesigned and rewritten over a 5 month period by a 12-student team and a xLx<sup>2</sup> system is now available [Mue08].

### 2.3.1 The Automatic Assessment

The xLx system automatically assesses the exercise solutions with two types of tests: static and dynamic. Only one static test is performed and simply consists in verifying whether or not the code compiles. The dynamic tests are specified with the JUnit test framework. To better handle the compilation and test process, Apache Ant is used to both compile the submitted solutions and execute the JUnit test cases [SVW06].

### 2.3.2 Setup of New Exercises

When setting up a new exercise, tutors have to define a set of properties [SVW06] (see also figure 2.7):

- **Section** - the exercise can be assigned to an existing section comprising associated exercises;
- **Level** - this property defines the difficulty of the exercise;
- **Exercise type** - the only exercise type available for programming assignments is Java;
- **Exercise text** - description of the exercise to be solved.

In addition, the files containing the dynamic JUnit test cases need to be indicated. When creating the tests (which as to be done with an external IDE or text editor), tutors can declare them either as *public* or *hidden*. The result of *public* tests is presented to students, but does not count for the grading of the exercise, while *hidden* tests results are not presented and count for the grading.

The system also provides the tutors with the facility of uploading sample solutions that can be used by the correctors of the exercise.

### 2.3.3 Solving Exercises

The workflow followed by students for solving an exercise can be defined as follows:

1. Login into the system and read the exercise description which explains the specifications that the solution has to comply with;
2. Solve and test the exercise locally, in their workplace;
3. Submit their solution for assessment, by uploading it to the system. Once a correct submission is made, i.e. the code compiles, solutions cannot be changed anymore.

### 2.3.4 Feedback System

When students submit a solution, they receive immediate feedback about whether or not their solution compiles successfully. In the case of an unsuccessful compilation, students get the chance to submit the code again. After a successful compilation, students receive immediate feedback about the performance on the *public* test cases. The system distinguishes between two kinds of test case failure: the program output does not match the expected or an unhandled exception occurred during the execution.

The system provides the tutors with a feature to easily annotate students' source code. For these annotations, a special Java syntax-highlighting scheme (figure 2.8) is used to make the reading of the source code more comfortable [SVW06].



### 2.3.5 Architecture

In terms of architecture, xLx is a Web based application implemented in typical three-tier client-server architecture and to access it only a standard Web browser is needed. The xLx platform is implemented on top of an Apache Web server and a mySQL database running on a Linux platform [HLVW02, SVW06].

### 2.3.6 Graphical Interface

The Web pages that compose xLx's graphical interface are generated dynamically by PHP4 scripts and Java Servlets. Figure 2.7 presents the graphical interface used by a tutor to configure new exercises, while figure 2.8 presents the window where students can consult the annotations performed by their tutors.

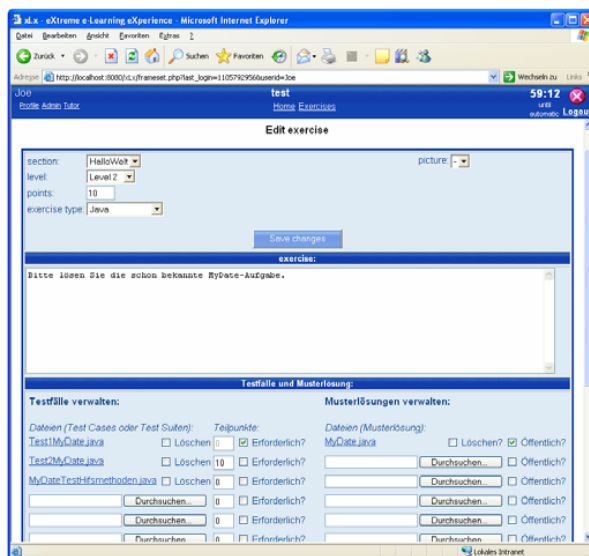


Figure 2.7: xLx interface - GUI for a tutor to configure new exercises

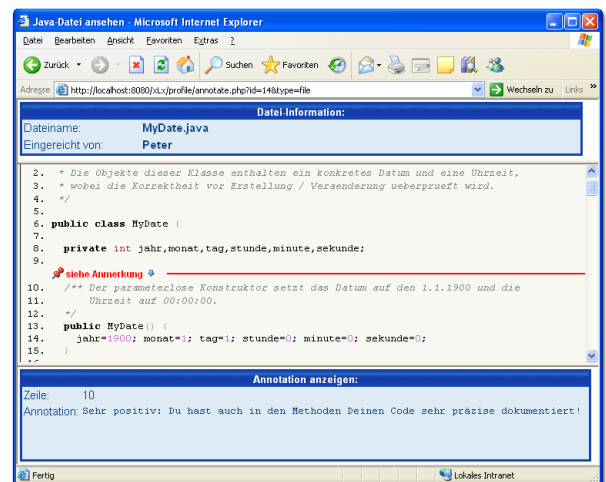


Figure 2.8: xLx interface - Annotation window from the student's point of view

### 2.3.7 Security and Reliability

Since unknown code provided by students is compiled and executed on the same server on which xLx itself is running on, special security measures are taken to guarantee that malicious code does not affect the system. Given that Java applications run on a virtual machine, Java policies are used to define a very restricted policy for the programming exercise module [SVW06].

## 2.4 Mooshak

Mooshak is a system developed at the DCC-FC and LIACC of the University of Porto [LS03] with the main purpose of conducting programming contests based on the ICPC rules [AI10]. Although the system was originally intended for contests, it is increasingly being used in programming courses: to give instant feedback on practical classes, to receive and validate assignments submissions, to pre-evaluate and mark assignments, etc [Lea09].

### 2.4.1 The Automatic Assessment

Mooshak's automatic assessment consists in two different types of analysis [LS03]:

- **Static** - checks for integrity of the submission. For instance, the size of the program is verified to prevent a denial of service attack by a submission of a too large program. Also, the system tries to compile the code. If the compiler produces errors or warnings, the assessment is aborted and an error message is presented to the student. Otherwise, an executable program is produced to be used in the dynamic analysis;
- **Dynamic** - in the dynamic analysis, the solution is tested against a set of test cases, defined by input and output files. Depending on the behavior of the submitted program, the system produces different classifications which have different levels of severity [LS03]. From the most to the least severe, the classifications may be: Requires re-evaluation, Time-limit exceeded, Output too long, Run-time error, Wrong answer, Presentation error and Accepted;

### 2.4.2 Setup of New Exercises

When setting up a new exercise, and after defining the text of its description, exercise developers have to specify a set of input and output files for testing the submissions. They also need to upload a sample solution to the problem which is used by the system to automatically determine the maximum allowed running time for the submissions [LS03].

### 2.4.3 Solving Exercises

The workflow for solving an exercise can be defined as follows:

1. Login into the system and read the exercise description which explains the problem and the formats of the expected input and output;
2. Solve and test the exercise in a local workplace;
3. Submit the solution for assessment, by uploading it to the system. The system shows the classification resultant from the static and dynamic analyses.

#### **2.4.4 Feedback System**

Immediately after a submission has been made, Mooshak produces feedback which consists in the classifications produced by the static and dynamic analyses [LS03]. In case of a compilation error or warning, an additional message containing the compiler output may be presented.

#### **2.4.5 Management of Submissions**

Mooshak provides some features related to the management of students' submissions. For instance, it is possible to re-evaluate submissions, by re-running the automatic assessment process. This is useful when mistakes in the input or output files are detected. It is also possible to consult information related to the submissions: source code, running time, memory used, produced output, etc.

#### **2.4.6 Architecture**

The architecture of Mooshak is that of a typical Web application: a client-server framework connecting the users with the machine where submissions are assessed [LS03]. The server is an Apache HTTP server extended with external programs using the CGI protocol and running on Linux. The external programs are responsible for generating HTML interfaces and processing form data [LS03].

#### **2.4.7 Graphical Interface**

Figure 2.9 presents a screenshot of Mooshak's HTML interface. The screenshot illustrates the judges' view of the list of submissions [LS03].

#### **2.4.8 Security and Reliability**

In order to assure the security of the server where Mooshak is running on, a set of security measures are used. For instance, all the submissions are executed in a sandbox with restricted privileges. Furthermore, the system defines upper bound limits for execution and compilation time, code size, output size, process data and stack size [LS03].

#### **2.4.9 Extensibility**

Although no information was found about the ease of extending Mooshak functionalities, it is important to note that, being an open source project, it can be extended and customized by any institution.

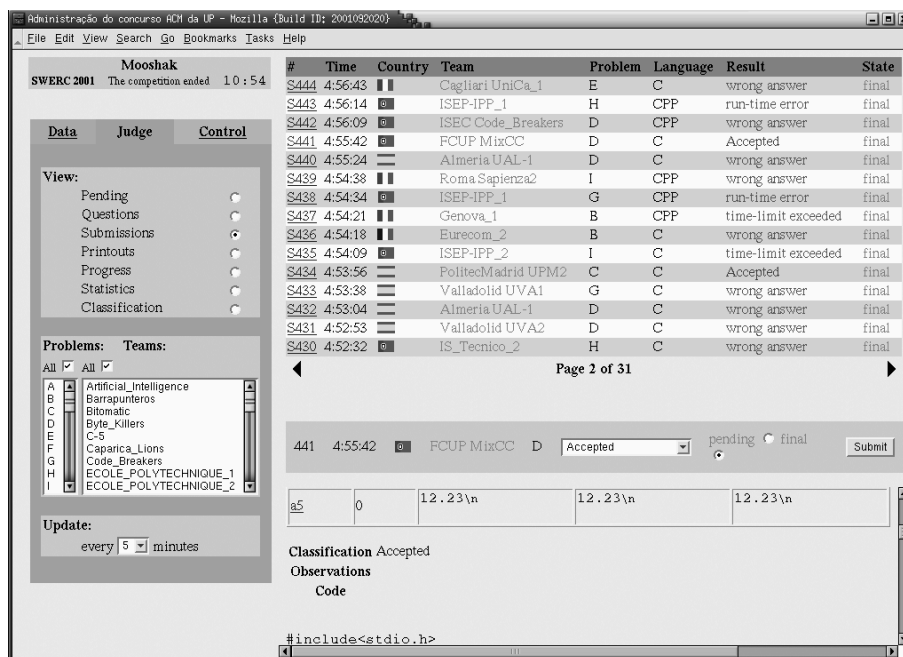


Figure 2.9: Mooshak graphical interface - judges' view

## 2.5 RoboProf

RoboProf is an online teaching system based on WWW technology and developed at the Department of Computer Applications of the Dublin City University to automatically mark Java programming exercises [Dal99, DH04]. The access to exercises is sequential, which means that students have to solve the easier problems in order to have access to the harder ones. Besides the automatic correction of code, RoboProf has a feature for plagiarism detection.

### 2.5.1 The Automatic Assessment

RoboProf has a marking scheme that corrects a program depending on its output. It compares the numbers in the output with the expected output, using a set of test data that may be fixed for each submission or generated randomly. Random generation of test data prevents students from tricking the system by writing code that just prints the expected output [DH04].

### 2.5.2 Setup of New Exercises

Setting up a new assignment requires the teaching staff to define the start time and deadline, as well as the information needed to set up the test data.

### **2.5.3 Solving Exercises**

The workflow followed by students for solving an exercise can be defined as follows:

1. Login into the system and read the exercise description which explains the problem and the formats of the expected input and output;
2. Solve and test the exercise locally, in their workplace;
3. Submit their solution for assessment, by uploading it to the system, and receive marks and feedback. There is neither limit for the maximum number of submissions nor penalization for multiple submissions (within the prescribed deadline).

### **2.5.4 Feedback System**

After submitting their solutions, students receive automatic and immediate feedback regarding the obtained mark, in percentage, and the expected and obtained outputs.

### **2.5.5 Security and Reliability**

The automatic testing is performed in students' machines in order to reduce the load on the server and to facilitate immediate feedback [DH04]. This also contributes for keeping the server safe from malicious code.

### **2.5.6 Extensibility**

As referred in [DH04], the default behavior of RoboProf is to present a constant specification, to mark the programming exercises based on output, and to read a file containing the required test data. However, since the system is written in Java, dynamic loading of program modules is possible without having to modify the engine. In addition, RoboProf is structured in a hierarchical fashion which makes it possible to add new programs and marking schemes. In addition, since marking is based on output, RoboProf may be used to teach any programming language using the same set of exercises.

## **2.6 Submit!**

Submit! is an open-source CBA system developed at the School of Communications and Informatics of the Victoria University of Technology with the goal of improving the quantity and quality of feedback given to students studying introductory Java programming [VH03, PRS<sup>+</sup>03].

### **2.6.1 The Automatic Assessment**

Submit! automatically evaluates the correctness of the submitted solutions by applying automatic tests defined by text files with input and output data. The system also analysis students' code for elements of good style such as comments and length of methods [VH03].

### **2.6.2 Setup of New Exercises**

Setting up an assignment requires some input from the lecturer, such as due date, number and weighting of marking criteria, the type of output and samples of input and desired output [VH03].

### **2.6.3 Solving Exercises**

The workflow followed by students for solving an exercise can be defined as follows:

1. Login into the system and read the exercise description which explains the problem and the formats of the expected input and output;
2. Solve and test the exercise locally, in their workplace;
3. Submit their solution for assessment, by uploading the source files to the system and by nominating a main class. Run the automatic tests by requesting for automatic feedback. There is no limit for the maximum number of submissions (within the prescribed deadline).

### **2.6.4 Feedback System**

After submitting their solutions and requesting for the automatic feedback, students immediately receive some comments regarding code quality. Afterwards, and if the code compiles successfully, the program's output is displayed for the students to compare with the model output supplied by the lecturer. Apart from this immediate and automatic feedback, students are also able to view tutor's comments and their final mark on-line as soon as the tutor has graded it [VH03].

### **2.6.5 Management of Submissions**

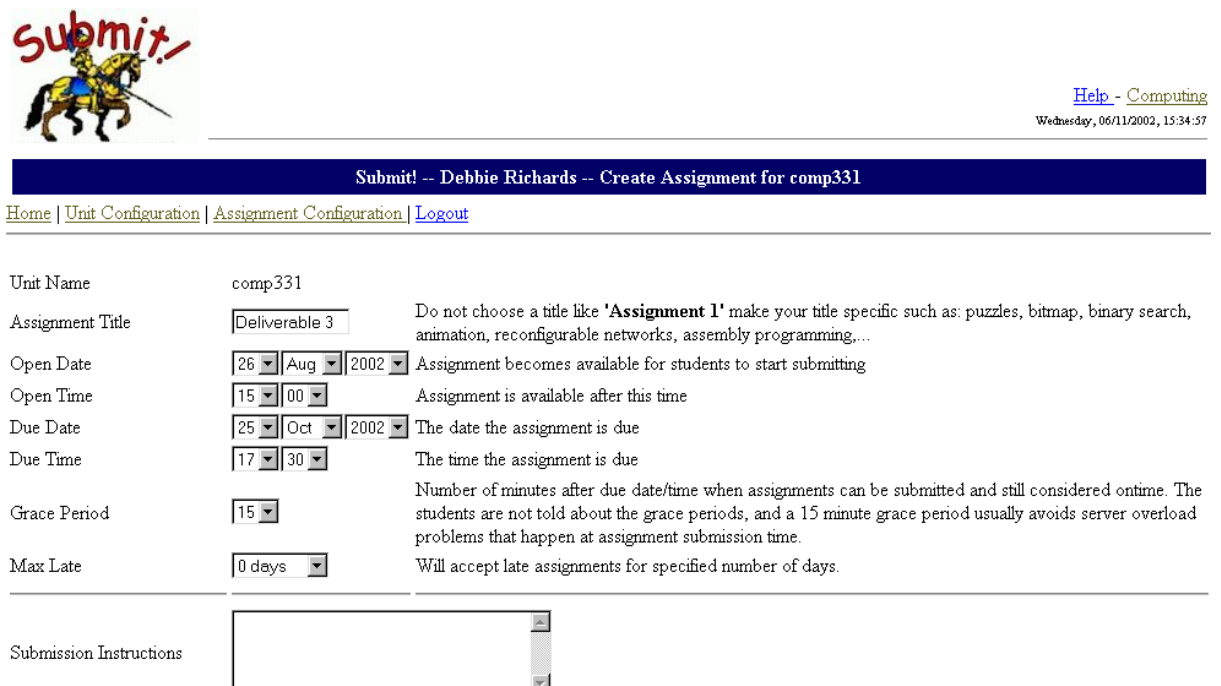
Submit! has interfaces available to the teaching staff for viewing submitted files and grade assignments, for consulting various statistics and review or re-assess students' submissions [VH03].

### 2.6.6 Architecture

Submit! is written in Java using servlets, residing on a dedicated UNIX machine running a Java servlet platform, Tomcat. The database backend is a hierarchy of flat files [VH03].

### 2.6.7 Graphical Interface

Figure 2.10 presents a screenshot of Submit!'s interface. The screenshot illustrates the screen for the basic configuration of an assessment [PRS<sup>+</sup>03].



**Submit!**

[Help - Computing](#)  
Wednesday, 06/11/2002, 15:34:37

**Submit! -- Debbie Richards -- Create Assignment for comp331**

[Home](#) | [Unit Configuration](#) | [Assignment Configuration](#) | [Logout](#)

Unit Name	comp331	
Assignment Title	<input type="text" value="Deliverable 3"/>	Do not choose a title like ' <b>Assignment 1</b> ' make your title specific such as: puzzles, bitmap, binary search, animation, reconfigurable networks, assembly programming,...
Open Date	<input type="text" value="26"/> <input type="text" value="Aug"/> <input type="text" value="2002"/>	Assignment becomes available for students to start submitting
Open Time	<input type="text" value="15"/> <input type="text" value="00"/>	Assignment is available after this time
Due Date	<input type="text" value="25"/> <input type="text" value="Oct"/> <input type="text" value="2002"/>	The date the assignment is due
Due Time	<input type="text" value="17"/> <input type="text" value="30"/>	The time the assignment is due
Grace Period	<input type="text" value="15"/>	Number of minutes after due date/time when assignments can be submitted and still considered ontime. The students are not told about the grace periods, and a 15 minute grace period usually avoids server overload problems that happen at assignment submission time.
Max Late	<input type="text" value="0"/> days	Will accept late assignments for specified number of days.

Submission Instructions

Figure 2.10: Submit! graphical interface - basic assessment configuration screen

### 2.6.8 Security and Reliability

Security is assured via login names and passwords already allocated to students by the School. Students must also be officially enrolled in a subject in order to login. A restricted execution environment, a Java sandbox, is used to execute the students programs [VH03].

### 2.6.9 Extensibility

Although no information was found about the ease of extending Submit! functionalities, it is important to note that, being an open source project, it can be extended and customized by any institution.

## 2.7 GAME

The Generic Automated Marking Environment (GAME) was developed at the School of Information Technology of the Griffith University, Australia. GAME is the successor of the C-Marker [GVN02] system which marks programming assignments written in C, and was designed to address its limitations [BGNM04a]. GAME has been developed to automatically assess programming assignments written in a variety of languages, to apply a generic strategy for looking at source code structure and to apply different types of marking strategies for examining the correctness of students' program output. As of the publication of [BGNM04a] it was still only able to mark programs written in Java and C.

### 2.7.1 The Automatic Assessment

Teaching staff can define different marking criteria for different types of assessment [BGNM04a]. This allows for the use of different marking strategies to test the correct results produced by the submitted programs. As of the publication of [BGNM04a] there were two marking strategies available: a "keyword" strategy that examines the student's program output for a keyword at any position and an "ordered keyword" strategy, that looks for an ordered set of keywords in the student's program output.

Students' solutions also receive marks regarding the code structure, which is calculated in three parts [BGNM04a]: number of comments, valid declaration of variables and number of "magic numbers" used in the code.

### 2.7.2 Setup of New Exercises

To be able to dynamically mark different types of programming assignments, the GAME system requires the assessors to fill out a marking schema when creating a new assignment [BGNM04b]. The marking schema contains information about assignment files, assignment marks and marking criteria [BGNM04a].

### 2.7.3 Solving Exercises

The workflow followed by students for solving an exercise can be defined as follows:

1. Login into the system and read the exercise description which explains the problem and the marking criteria;
2. Solve and test the exercise locally, in their workplace;
3. Submit their solution for assessment, by uploading the source files to the system;



### 2.7.4 Feedback System

GAME does not generate feedback to students. The only feedback is available to tutors and contains, for each submission, a mark for source code structure, a mark for correctness of the produced output and a list of warnings or compile-time errors (if any) [BGNM04a].

### 2.7.5 Management of Submissions

Teaching staff can browse the overall results for each student. This is displayed in a summary format that shows the feedback described in the previous subsection.

### 2.7.6 Graphical Interface

GAME's interface was developed with Java Swing and can be accessed with a web browser. Figure 2.11 presents a sample screenshot [BGNM04a].

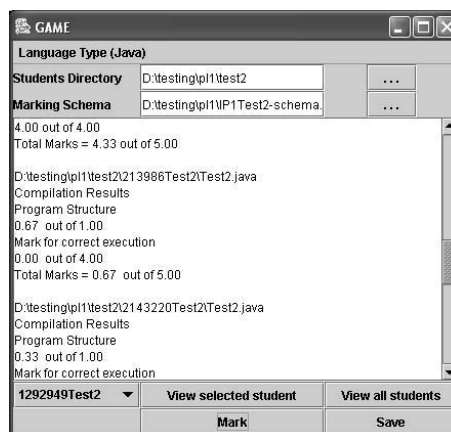


Figure 2.11: GAME user interface

### 2.7.7 Extensibility

GAME was designed with many extensibility issues in mind. As referred in [BGNM04a], a framework has been set in place to enable easy addition of new marker modules to extend the system's functionalities. Although it still only supports marking of Java and C assignments, it is relatively easy to include further types of languages. Moreover, the marking system schema will play an important role in the future development of GAME, since it will eventually enable teaching staff to build marking schemas without understanding the schema's structure [BGNM04b].

## 2.8 DOMjudge

DOMJudge is a system created at Study Association A-Eskwadraat of the Utrecht University, the Netherlands [EKW09a], for running programming contests like the ACM ICPC regional and world championship programming contests [EKW10b, EKW10a]. In this type of programming contests, teams are on-site and have a fixed time period (mostly 5 hours) and one computer to solve a number of problems. Problems are solved by writing a program in one of the allowed languages and the judging is done by submitting the solution source code to the jury, where the code is compiled and run against a set of test cases [EKW10b, EKW10a].

### 2.8.1 The Automatic Assessment

The flow of the automatic assessment can be described as follows [EKW10b]:

1. Team submits solution. It will either be rejected after basic checks, or accepted and stored as a submission;
2. The first available judgehost compiles, runs and checks the submission. The outcome and outputs are stored as a judging of this submission;
3. The result is automatically recorded and the team can view the result and the scoreboard is updated. A judge can optionally inspect the submission and the automatic judging and mark it verified.

The submitted solutions are tested against a set of test cases, defined by input and output files. To allow for problems that do not fit within the standard scheme of fixed input and/or output, DOMjudge has the possibility of changing the way submissions are run and checked for correctness. This can be done by implementing a special compare script, also called a *validator* [EKW10a].

The status/classification of a given submission can be one of the following [EKW10b, EKW09b]: Queued/Pending, Judging, Too-Late, Correct, Compiler-Error, Timelimit, Run-Error, No-Output, Wrong-Answer and Presentation-Error.

### 2.8.2 Setup of New Exercises

Configuring DOMjudge to run a contest involves configuring the contest data, setting up authentication for teams, supplying input and output test data and checking that everything works [EKW10a]. For each specific exercise, the following data has to be specified: problem description and id, whether or not submissions are accepted, runtime limit for the solutions, input and output files with test data and, optionally, a *validator* for handling problems that do not fit within the standard scheme of fixed input and/or output.

### 2.8.3 Solving Exercises

The workflow for solving an exercise can be defined as follows:

1. Login into the system and read the exercise description which explains the problem and the formats of the expected input and output;
2. Solve and test the exercise in a local workplace;
3. Submit the solution for assessment, either via a command line submit program or via a web interface. The system shows the classification resultant from the automatic assessment and updates the scoreboard.

### 2.8.4 Feedback System

Immediately after a submission has been made and the code has been compiled and tested by an automatic judge, DOMjudge produces feedback containing the status/classifications listed in [2.8.1](#).

### 2.8.5 Management of Submissions

DOMjudge provides some features related to the management of submissions. For instance, it is possible to consult details of submissions such as its id, source code, author, submit time, runtime, produced output and diff with test data. It is also possible to rejudge submissions and to restart pending judgments.

### 2.8.6 Architecture

DOMjudge has a distributed architecture, based on a client-server framework. It discerns three different kinds of hosts [[EKW10a](#)]:

- **Team computer** - team workstations where solutions are developed and submitted. The only part of DOMjudge that may run here is the command line submit client;
- **DOMjudge server** - host, running on Apache, receives the submissions, runs a MySQL database for keeping the submissions and serves web pages to teams, jury and administrators;
- **Judgehosts** - a number of hosts, at least one, that retrieves submitted solutions from the DOMjudge server, compile and run them and send the result back to the server.

### 2.8.7 Graphical Interface

DOMjudge provides web interfaces for teams, juries and administrators. The web interface allows the teams to submit their solutions and to read problems' descriptions. Juries and administrators have access to a set of features for monitoring contest data [EKW10a]. Moreover, the interface presents the scoreboard with contests' results.

### 2.8.8 Security and Reliability

DOMjudge judging system was developed with security as one of the main concerns [EKW10a]. There are mechanisms for restricting team access to others and the internet, for restricting access to the submitted programs on the jury computers, for restricting the environment where solutions are run, etc [EKW10a].

Since compiling and testing submissions is computationally intensive, it is recommended to use at least a couple of judgehosts. Moreover, for security and performance reasons it is highly recommended not to use the DOMjudge server as a judgehost [EKW10a].

### 2.8.9 Extensibility

Although no information was found about the ease of extending DOMjudge functionalities, it is important to note that, being an open source project, it can be extended and customized by any institution.

## 2.9 Summary of Features

This section contains a summary and comparison of the features available in the different CBA systems presented throughout the current chapter. The different features were grouped in nine categories, each one represented by one subsection.

### 2.9.1 Support for Assessment Creation

The analysis of the features available in the different systems, and related to the support for assessment creation, can be divided in two parts: analysis of the different types of assessment that can be created and analysis of the parameters that can be used to configure those assessments.

#### 2.9.1.1 Types of Assessment Supported

Table 2.1 shows the type of assessments supported by the CBA systems. The meaning of each type of assessment is as follows:

- **Exams** - traditional (programming) exams that have to be solved by all the students at the same time, within a given time limit;
- **Tests** - programming tests where students are divided in groups. All the students in each group have to solve the test at the same time, within a given time limit;
- **Exercises** - self-learning programming exercises that can be solved by all the students, without time limit;

	<b>Exams</b>	<b>Tests</b>	<b>Exercises</b>
CourseMarker	Yes	Yes	Yes
BOSS	Yes	Yes	Yes
xLx	Yes	Yes	Yes
Mooshak	Yes	Yes	Yes
RoboProf	Yes	Yes	Yes
Submit!	Yes	Yes	Yes
GAME	Yes	Yes	Yes
DOMjudge	Yes	Yes	Yes

Table 2.1: Types of assessments supported by CBA systems

As can be seen, none of the analyzed systems presented restrictions in terms of the types of assessment supported.

### 2.9.1.2 Parameters for Assessment Configuration

The concept of assessment varies between systems. For instance, in BOSS, an assessment may be composed by several exercises, while in CourseMarker, Mooshak and xLx it is composed by one single exercise. The parameters available in the different systems for configuring an assessment are as follows:

- **CourseMarker** - start time, deadline, exercise filename, maximum number of submissions and whether the exercise is open or closed to students;
- **BOSS** - start time, deadline and contribution of each problem/exercise for the global mark;
- **xLx** - start time, deadline, level and type;
- **Mooshak** - start time, deadline and accepted programming languages;
- **RoboProf** - start time and deadline;
- **Submit!** - start time, deadline and penalization for late submissions;
- **GAME** - start time and deadline;
- **DOMjudge** - start time, deadline and accepted programming languages.

## 2.9.2 Support for Test-Cases Definition

Similarly to what was done for the assessment creation analysis, the analysis of the features related to the definition of test-cases was divided in two parts: analysis of the available methodologies and analysis of the configuration parameters.

### 2.9.2.1 Available Methodologies

Table 2.2 presents the methodologies for test case definition supported by the different CBA systems. Two main methodologies were identified: usage of text files with the expected input and output and definition of JUnit test cases.

	Input and output files	JUnit test cases
CourseMarker	Yes	No
BOSS	Yes	Yes
xLx	No	Yes
Mooshak	Yes	No
RoboProf	Yes	No
Submit!	Yes	No
GAME	Yes	No
DOMjudge	Yes	No

Table 2.2: Methodologies for test case definition available in CBA systems

Almost all of the systems support the usage of input and output text files, the only exception being the xLx system. Only BOSS supports both input and output files and the definition of JUnit test cases.

### 2.9.2.2 Test-Cases Configuration

The available parameters for test-cases assessing and configuration are as follows:

- **CourseMarker** - maximum size of solution output, maximum running time, static analysis of a set of code quality parameters;
- **BOSS** - contribution of each test case for the global mark, static analysis of a set of code quality parameters;
- **xLx** - contribution of each test case for the global mark;
- **Mooshak** - contribution of each test case for the global mark, maximum running time, maximum heap and stack memory, maximum code size, maximum size of solution output;
- **RoboProf** - possibility of random generation of test data;

- **Submit!** - contribution of each test case for the global mark, static analysis of a set of code quality parameters;
- **GAME** - static analysis of a set of code quality parameters;
- **DOMjudge** - maximum running time and possibility of defining *validators* for special output correction.

### 2.9.3 Support for Elaboration and Submission of Solutions

The support for elaboration and submission of solutions can be summarized by analyzing: if the students can directly use the CBA systems to develop their solutions or if they have to develop them in their local environment; if the systems allow the teaching staff to define a skeleton solution for the exercises.

#### 2.9.3.1 Environment for Solutions Development

Table 2.3 shows the environment where the students have to develop their exercise solutions: directly in the CBA system or in their local desktop using the tools of their choice. As can be easily observed, none of the systems supports the direct elaboration of solutions.

System	Environment
CourseMarker	Locally
BOSS	Locally
xLx	Locally
Mooshak	Locally
RoboProf	Locally
Submit!	Locally
GAME	Locally
DOMjudge	Locally

Table 2.3: Environment for Solutions Development

#### 2.9.3.2 Support for Skeleton Solution Mechanism

Table 2.4 shows whether or not the different CBA systems allow the definition of skeleton solutions. This feature proved to be a bit unpopular, being only supported by the CourseMarker system.

### 2.9.4 Feedback System

Different CBA systems usually have different mechanisms for handling feedback. The feedback may be immediate or non-immediate, manual or automatic (or a combination

## CBA Systems

System	Supports?
CourseMarker	Yes
BOSS	No
xLx	No
Mooshak	No
RoboProf	No
Submit!	No
GAME	No
DOMjudge	No

Table 2.4: Support for Skeleton Solution Mechanism

of both) and its level of detail may vary. The different feedback mechanisms may be summarized as follows:

- **CourseMarker** - feedback is fully automatic and immediate. It may include, for instance, comments about the quality of the code and on how to improve the solutions. The feedback detail can be regulated;
- **BOSS** - the only immediate feedback comes from a set of test cases made available before the submission. The final feedback is not immediate and is a combination of automatic (only about the final grade) and manual feedback which can be added by human markers;
- **xLx** - the only immediate feedback comes from a set of public test cases that do not count for the final grade. The final feedback is not immediate and is a combination of automatic (only about the final grade) and manual feedback which can be added by human markers;
- **Mooshak** - feedback is fully automatic and immediate. Each submission receives a classification (e.g.: Requires re-evaluation, Time-limit exceeded, Output too long, Run-time error, Wrong answer, Accepted);
- **RoboProf** - the feedback is fully automatic and immediate. For each submission, the system indicates the test data and the expected and obtained outputs;
- **Submit!** - the feedback is composed by two components: an immediate and automatic one and a manual and non immediate one. The students automatically receive feedback about the static analysis of the code quality and about the expected and obtained output. The manual feedback comprises tutor's comments and the final grade;
- **GAME** - the version of GAME described in literature did not have any mechanism for feedback generation. However, newer versions should include this feature [BGNM04a];



- **DOMjudge** - whenever human verification of the submissions is not required, feedback is fully automatic and immediate. Each submission receives a classification (e.g.: Queued/Pending, Judging, Too-Late, Correct, Compiler-Error, Timelimit, Run-Error, No-Output, Wrong-Answer).

### 2.9.5 Features for Submission Management

The main functionalities available to the course tutors for management and analysis of the students' submissions are:

- **CourseMarker** - consult history of submissions and grades of each student, view grades statistics, view missing/submitted students and expunge students' work;
- **BOSS** - run the automatic tests on the submissions, mark the submissions manually, provide manual feedback, consult students' details, penalize late submissions, submit in behalf of a student and publish marks;
- **xLx** - consult history of submissions, provide manual feedback;
- **Mooshak** - consult history of submissions, re-evaluate submissions and consult information related to the submissions such as: source code, runtime, memory used and produced output;
- **RoboProf** - none;
- **Submit!** - consult history of submissions, provide manual feedback, consult various statistics and review or re-assess students' submissions;
- **GAME** - consult history of submissions and grades of each student;
- **DOMjudge** - consult history of submissions, rejudge submissions, restart pending judgments and consult information related to the submissions such as: id, source code, author, submit time, runtime, produced output and diff with test data.

### 2.9.6 Extra Features

This section groups that features that, although not essential for the main purposes of a CBA system, add useful value.

- **CourseMarker** - plagiarism detection and built-in Q&A mechanism for doubts clarification;
- **BOSS** - plagiarism detection;
- **xLx** - none;

- **Mooshak** - built-in Q&A mechanism for doubts clarification and submissions ranking;
- **RoboProf** - plagiarism detection;
- **Submit!** - none;
- **GAME** - plagiarism detection;
- **DOMjudge** - built-in Q&A mechanism for doubts clarification, submissions ranking and tools for checking mistakes in test data, such as leading trailing whitespace and non-printable characters.

### 2.9.7 Programming Languages Supported

Table 2.5 presents the programming languages supported by the CBA systems (some non-programming languages such as SQL, XSLT and XQuery were also included).

System	Languages
CourseMarker	C/C++ and Java
BOSS	C/C++ and Java (but in theory can be configured for any language)
xLx	Java, SQL, XSLT and XQuery
Mooshak	Any (e.g: C/C++, Java, Python, Lisp, Prolog, Haskell)
RoboProf	Java
Submit!	Java
GAME	C and Java (but in theory can be configured for any language)
DOMjudge	In theory can be configured for any language

Table 2.5: Programming Languages Supported by the CBA systems

### 2.9.8 Security

The implemented mechanisms for assuring systems' security are the following:

- **CourseMarker** - has a password encryption mechanism, a session key security feature and a sandbox with restricted privileges for running the submissions;
- **BOSS** - has a sandbox able to limit the amount of available CPU time and memory and with strictly controlled set of permissions;
- **xLx** - Java policies are used to define a very restricted policy for running students' code;
- **Mooshak** - has a sandbox with restricted privileges and able to limit execution and compilation time, code size, output size, process data and stack size;

- **RoboProf** - the automatic assessment is performed on students' machines, which keeps the server safe from malicious code;
- **Submit!** - a Java sandbox with restricted privileges is used to execute students' programs;
- **GAME** - no security mechanism was found in literature;
- **DOMjudge** - has mechanisms for restricting team access to others and the internet, for restricting access to the submitted programs on the jury computers and has a sandbox with restricted privileges for running the submissions.

### 2.9.9 Type of Interface

Table 2.6 presents the type of interfaces implemented in the different CBA systems. Almost all of the systems can be accessed via a web browser. The only exception is the CourseMarker system which was developed has a desktop application which connects to the web.

System	Interface
CourseMarker	Java Swing
BOSS	Web and Java Swing
xLx	Web
Mooshak	Web
RoboProf	Web
Submit!	Web
GAME	Web (Java Applet)
DOMjudge	Web

Table 2.6: Type of interface of CBA systems



## Chapter 3

# Application of CBA Systems in e-Learning

This chapter approaches the topic of how the CBA systems are being used to support the e-Learning process in some academic institutions. Some case studies, related to the application of five of the systems previously presented, are analyzed. From this analysis emerged some best-practices of the application of CBA systems in e-Learning.

### 3.1 Case Studies Analysis

This section presents how CBA systems support the learning process in different Universities. The list of Universities used as case studies, as well as the system used by each one, is presented in table 3.1.

Institution	CBA System used
University of Nottingham	CourseMarker
University of Warwick	BOSS
University of Muenster	xLx
University of Murcia	Mooshak
Dublin City University	RoboProf

Table 3.1: CBA systems used in different institutions

In order to contextualize the use of the CBA systems, there is a description of the programming courses that used them as a support. The comparison of the different approaches is based on learning components such as the automatic assessment, test case definition methodologies, feedback, resubmission policy, plagiarism control and security. Some quantitative and qualitative results about the CBA usage are also presented, as well as some best-practices that emerged from the different approaches.

### 3.1.1 Course Content

CourseMarker has been used by the University of Nottingham to support the teaching of Java to first year students. Since many of the students had never programmed before, the first two-thirds of the first semester are spent teaching as if Java were a procedural language. The course is split into several units based on simple programming concepts, from the very basic use of variables, to methods, strings and file processing. Students are assessed with a mixture of different types of assignments [HGST05]: weekly exercises, exercise plus report, a programming exam and multiple-choice questions.

The BOSS system was used in fifteen courses at the University of Warwick. However, only four of them needed the automatic assessment facilities. Three of the courses comprise the teaching of Java and the other simple UNIX Shell and Perl [JGB05].

As of the publication of [SVW06], xLx had been only tested in small Java courses at the University of Muenster. These courses are usually organized in sections comprising several exercises related to different programming concepts.

The RoboProf system has been also used to teach a Java course. Thought at the Dublin City University, the course contains nearly 300 students [DH04]. The assessment is comprised by exercises and a final examination, corrected by RoboProf, and by three assignments that are manually marked. At total, there are 51 RoboProf exercises of increasing difficulty and students cannot proceed from one question to the next until a passing grade has been awarded [DH04].

At the University of Murcia, Mooshak was used to support the replacement, in a computer science course, of a traditional final exam evaluation by a series of programming activities. These activities involve [GMFA09]: independent problems with varying difficulty; dependent problems that have to be solved in a given order; contest-style activities where the students have to solve up to 9 problems within 4 to 6 hours; designing activities where the students have to create new problems in the Mooshak system. These activities are closely related to the six cognitive levels of Bloom's taxonomy [GMFA09, Blo56].

### 3.1.2 Assessment

Student's solutions are usually assessed with two different types of tests based on well known methods from the field of software testing: dynamic and static.

With dynamic tests, solutions are run against a set of predefined test cases. In CBA systems where the feedback is totally automatic, the results of these tests are automatically incorporated in the final mark. This kind of approach is used at the Universities of Nottingham, Murcia and Dublin. A down point is that it may penalize students that fail the test cases due to small details (like white spaces) concerning the input and output specifications. This can be avoided if a multiple submission policy is adopted or if the system allows the re-evaluation of submissions. At Warwick and Muenster, instead of being

automatically incorporated in the final mark, the results of the dynamic tests are used to support the grading by human markers. With this methodology, systems like BOSS allow the teaching staff to submit in the behalf of the students in order to fix small submission mistakes [JGB05].

There is a wide range of static tests that can be performed. The most basic one, and the only used at Muenster, is to check for the syntactic and semantic correctness of the code, i.e. to compile the code. The Mooshak system performs an additional check to prevent a denial of service attack by a submission of a too large program [LS03].

Static tests can also be used to evaluate aspects concerning the quality of a program. BOSS authors identify a set of these aspects [JGB05]: comments in code, code style, code structure, use of external libraries, choice and efficiency of algorithm. They consider that many of these aspects are subjective and cannot easily be assessed automatically and hence should be assessed manually by a human marker. To support the manual assessment, BOSS provides a set of program metrics, such as number of comments and percentage of methods declared abstract. However, since the incorporation of these metrics in the system is recent, as of the publication of [JGB05], they were still not being used in any course at Warwick.

At Nottingham, metrics on the quality of the code are directly and automatically incorporated in grading. Some of the tests performed by CourseMarker involve verification of layout, indentation, choice and length of identifiers and use of comments [HGST05]. CourseMarker contains a type of static test that was not found in any other system: feature test. At Nottingham, the use of exercises to teach the difference between different programming constructs (e.g. the difference between "if-then-else" and a "switch-case" statement) is common. This kind of exercises typically includes feature tests to check if the different constructs are properly used [HGST05].

### 3.1.3 Test Case Definition

Two main test case definition methodologies were found throughout literature. The first one consists in the utilization of input text files with test data and output files with the expected answers to the input data. The second methodology consists in the definition of JUnit test cases.

With the text files methodology, the input files are passed to the standard input of the programs and the results produced by the standard output are compared to the output files. Although this is a simple mechanism, it is also very powerful and allows the modeling of the strict input and output requirements that are often present in real-world software engineering tasks [JGB05]. It is the most popular methodology among the case studies, being used at the Universities of Nottingham, Warwick, Murcia and Dublin. At the latter, since an unlimited submission policy was adopted, random generation of test data is

used to prevent students from tricking the system by writing code to print the appropriate output [DH04]. It is important to note that it should be possible to easily incorporate new programming languages in a CBA system supporting this methodology (RoboProf is such an example [DH04]).

The use of JUnit test cases applies only when Java is the language used in the submissions. With this methodology, input and output are specified as Java objects and a test is constructed by specifying a method to be run, taking the input object as argument and returning the expected output object. Among the study cases, the only institution using exclusively JUnit tests was the University of Muenster. At the University of Warwick, although the BOSS system also supports the usage of JUnit tests, as of the publication of [JGB05] this functionality has not still been used in any course.

### 3.1.4 Feedback

As stated in [Cum08], the provision of feedback is arguably the most important aspect of the educational process. It "... is the life blood of learning" [Row87]. It allows students to get a commentary about their work and enables them to adjust their mental model in the light of the communication received. Any learning activity without associated "feedback is completely unproductive for the learner" [Lau93]. We as human beings learn through interacting with the external world and getting some sort of feedback from it [Lau93]. Academic knowledge is no exception to this, and so it follows that high quality, timely feedback is required in order to the learner to be able to learn. The sooner students can get feedback on what they have done the more effective it will be in causing modifications to their mental model and thus develop deeper understandings on their work [Cum08]. Different approaches have been used in respect to the nature, level of detail and immediacy of the feedback given in the automatic assessment of programming assignments.

At the Universities of Nottingham, Dublin and Murcia, feedback is automatically generated by the marking tools and is given immediately after students submit their solutions and the automatic assessment process concludes its work. Still, there are significant differences in their feedback methodologies.

At Nottingham, experience has shown that ultra-detailed feedback can be detrimental to the learning experience [HGST05]. Therefore, the CourseMarker system includes the possibility of regulating the amount of feedback given to students. This is used by exercise developers to adapt the level of feedback to the needs of specific classrooms. It is possible that futures versions of CourseMarker may include an intelligent mechanism for deciding how much feedback should be provided to students, based on their past grades and performances [HGST05, YH03].

Dublin's students get to know the input data that was used to assess their solutions, as well as both the expected and generated output [DH04]. Although this information may



be too detailed in some situations, it provides the students with the data needed to debug their incorrect submissions.

At Murcia, students receive the classifications produced by Mooshak's static and dynamic analyses [LS03]. Even though this information is not as detailed as the one from RoboProf, it provides useful information about the submission status.

At the Universities of Warwick [JGB05] and Muenster [SVW06] a different approach was chosen. The final feedback concerning the correctness of the solutions is not immediately delivered nor fully automatically generated. After the deadline for submissions has passed, course staff runs the automatic tests on students' solutions, analyses the source code and, in the case of Warwick, some automatically generated metrics concerning code quality. Based on all this information, the staff manually marks and attaches personalized feedback to the submissions. At Warwick this information is then e-mailed to students. At Muenster, the feedback is given directly in students' code by using xLx's special annotation system and hence students have to directly consult the assessment system. It is also important to add that, before submitting their solutions, students can still obtain some automatic feedback from the system by testing their code against sample tests provided by the exercise developer.

### 3.1.5 Resubmission Policy

Different resubmission policies may be used in a programming course with automatic assessment. Although a bit restrictive, a single submission policy is used at the University of Muenster [SVW06]. With this approach, after being submitted, a solution cannot be changed anymore.

A different approach was adopted in Nottingham, where students are usually given the chance to perform three submissions per exercise [HGST05]. Since CourseMarker generates automatic and immediate feedback, students get to know their mark right after submitting.

At Warwick, Dublin and Murcia there is no limit to the amount of submissions, although they all have to be made within a given deadline. In Dublin and Murcia all the submissions are automatically marked after being submitted. However, at Warwick, only the last submission is marked [JGB05].

BOSS authors argue that the fundamental difference between CourseMarker and their system is the paradigm for interacting with the students [JGB05]. They wanted BOSS to focus on the process of online submission and measuring the correctness of students' code, rather than become a tool with a broader support for formative assessment.

### 3.1.6 Plagiarism Control

Combating plagiarism is a major challenge for many educational establishments [HTS02, CL01] and this is reflected by the fact that plagiarism control facilities were used in all the case studies. Even at the University of Murcia an external tool was used to detect plagiarism, since Mooshak does not provide such feature [GMFA09]. It is also curious to note that, at Warwick, the BOSS system was used to detect plagiarism in non-programming courses [JGB05].

Most of the implemented techniques for detecting plagiarism consist in pair-wise analysis of text-free data files and reporting of pairs of documents that contain significant similarities. Such an example is the "Sherlock" [JL99] software, used in the BOSS system. A different technique based on an idea by Plauger [Pla94] is used in RoboProf. As described in [DH04], when a student submits a program to the system, a binary code invisible to most text editors and comprising the student ID, the assignment ID, the academic year and a checksum is added to the end of the source code. This code is then used to determine the extent of plagiarism for each student. The main advantage of this method over the pair-wise comparison is that plagiarism is detected in the moment of the submission and the system does not need to run an analysis program to perform explicit pair-wise comparison.

### 3.1.7 Security

Security is a very important factor in CBA systems [FHST98]. As stated in [HGST05], a poorly guarded and insecure system can potentially put at risk the marks, the submissions, and the course content and be open to malicious practice by dishonest students.

Usually, a CBA system contains two security levels. The first comprises, for instance, mechanisms to identify the student using the software, to assure the integrity of communication between server and client and to protect data stored on the system (including the submissions) from unauthorized access.

Since the systems run code written by students, there is a danger that a student's program may perform an unsafe operation, potentially damaging the machine running the system [JGB05]. Therefore, the second security level usually comes in the form of a sandbox with restricted privileges. Some of the typical restrictions are: access only to a temporary directory in the file system, limited CPU time, limited memory, limited compilation time, limited program size, etc. In systems implemented in Java, like xLx, the privileges are usually restricted by using the so-called Java policies [SVW06].

### 3.1.8 Results

The literature of four of the case studies also contained some qualitative and quantitative results about the use of the CBA systems. These results are now presented.

#### 3.1.8.1 CourseMarker

At Nottingham, CourseMarker's authors state that data, gathered during six years of usage of the system for the Java course, show that students became good programmers and achieved better marks as a result of the detailed on-demand feedback, coupled with the possibility of multiple submissions [HGST05]. This success is also attributed to the carefully designed exercises and to the improved support available for students. Furthermore, surveys among students show that this methodology motivates them into putting more effort to achieve better grades [HGST05]. A test to the efficacy of CourseMarker was also performed. In 2003, the students of the first semester were divided into two groups. One of the groups used CourseMarker and the other did the coursework manually and e-mailed the solutions to the lab assistants, who would then mark them by hand. The authors argue that comparing the results of both groups shows that CourseMarker marks at least as well as humans do, provides on-demand impartial feedback and, as a bonus, saves hundreds of marking hours for the academic staff.

#### 3.1.8.2 BOSS

BOSS authors employed a combination of techniques, such as interviews with staff from Warwick's University, in order to evaluate the system's impact [JGB05]. They argue that the software is now stable and that remaining issues relate principally to the lesser-used dialogs within the staff clients. Some interesting conclusions were drawn from the evaluation [JGB05]:

- Teaching staff suggested that the time necessary to devise and deploy a set of automatic tests with input and output files is typically 1 or 2 hours and that the time taken to mark single student's submission may be as low as a couple of minutes;
- Some students considered the automatic tests unfair against those who have tried and just failed to reach the required outcome. Others referred that the tests were too picky with white spaces;
- The fact that the final automatic tests were not provided to students was a source of complaints. Therefore, BOSS authors consider that it may be desirable to allow students access to some automatic tests to assist them in their program development.

### 3.1.8.3 Mooshak

At the University of Murcia, the results of the application of the automatic assessment with Mooshak were considered very promising [GMFA09]. For instance, the passing rate increased from 11% to 22% and the dropout rate decreased from 72% to 45%. Also, it was noted that the programming contest related features available in Mooshak played a fundamental role in the motivation of students. In one hand, the ranking system encourages the students to solve more problems, faster and more efficiently. On the other hand, since submissions are public, students do not have to feel frustrated when they receive a "wrong answer". They can see that their classmates are going through the same troubles, and that it is a part of learning [GMFA09].

### 3.1.8.4 RoboProf

At Dublin, RoboProf authors used the results of the final examination, done by 282 students and assessed by RoboProf, to analyze the system's effectiveness for teaching computing and of determining factors influencing programming performance [DH04]. The analysis was divided in 4 areas:

- **Previous qualifications** - since the course that used RoboProf is taken by first year students, the authors were interested in how differing entry qualifications affect subsequent performance. It was observed that many students that entered the university with low points achieved high grades in the programming examination. However, the correlation coefficient [DS97] between the entry points and the examination results pointed out that there is a positive linear trend between the two variables;
- **Gender** - the authors also analyzed whether there was a difference between females and males with respect to achieved programming skills. Out of the 282 students that took the exam, 79 (20%) were female, which is consistent with participation rates of women in computer science courses at universities worldwide [Cam97]. The analysis results show that males outperformed females. The authors had hoped that the use of RoboProf might have helped to improve the performance of women relative to their male counterparts [DH04];
- **Usage patterns** - authors also determined how the students' use of the RoboProf learning tool during the course affected their performance in the programming examination. Patterns and frequency of usage of the system were monitored and recorded. These records allowed for the definition of four variables that were used to investigate which factors impacted on programming ability [DH04];
- **Comparison with traditional computing course** - a comparison between the performance on the RoboProf course and a subsequent computing course in Java was

also performed. Among other conclusions, it was noted that who got high (low) marks in the first course are likely to get high (low) marks in the subsequent one.

### 3.1.9 Best Practices

Based on the presented analysis and results, some best-practices (BP) of the application of CBA systems in e-Learning may be pointed-out:

- **BP1** - Dynamic tests are essential for the automatic assessment process. Static tests, although not essential, may be useful to help students improving their coding style;
- **BP2** - None of the test case definition techniques may be considered to be the silver bullet. Although the usage of unit tests avoids difficulties related to the usage of text files like misspellings, control characters and whitespaces, it requires technical skills to code and are language-specific [JGB05];
- **BP3** - When using text files for defining test cases, problems with control characters and whitespaces may be minimized if a resubmission policy allowing multiple submissions is adopted and if sample input and output data is provided to students;
- **BP4** - Special care should be taken when using text files for defining test cases for introductory programming courses, where students still do not master input and output handling;
- **BP5** - Giving relevant feedback to students about their submissions is essential. The following combinations of feedback/resubmission policies have had good results:
  - Detailed feedback and limited (but more than one) number of submissions;
  - Less detailed feedback and unlimited number of submission within a given deadline;
- **BP6** - Plagiarism control plays a very important role in the University context. Knowing that plagiarism is controlled is a motivation for students to learn by themselves;
- **BP7** - Security in a CBA system is essential and should come in two levels: the first one to assure the authenticity and the integrity of information and the second one should be in the form of a sandbox for protecting the system from malicious code;
- **BP8** - Competition-like assessments may contribute to motivate students.



## Chapter 4

# Analysis of Learning Needs

This chapter presents the study performed to determine the learning needs of the teaching staff of DEI (Department of Informatics Engineering). In order to collect the opinions and CBA system feature needs, an online survey was conducted. The information gathered in the survey was then used to build the system specification.

### 4.1 Survey Analysis

In order to collect opinions from the teaching staff of DEI, a survey was created. This survey is written in Portuguese and is available online at <http://tinyurl.com/376yb9f> as well as in appendix A.

DEI teaching staff is involved in the courses of the Integrated Masters in Informatics Engineering and of the Integrated Masters in Electrical and Computers Engineering. The main goals were to understand staff needs, which features are considered to have more priority and which courses may use the system in the future. The survey was composed by 7 sections and the total number of answers was 17. The division in sections follows a similar pattern to the one used in the summary of the features of the CBA systems (section 2.9). The first 6 sections of the survey included questions related to the features presented in the first 6 sub sections of the summary of features:

1. Support for assessment creation;
2. Support for test-cases definition;
3. Support for elaboration and submission of solutions;
4. Feedback system;
5. Features for submission management;
6. Extra features.

#### 4.1.1 Responses Overview

The first part of the survey asked for the course taught by the teaching staff and whether or not a CBA system might be useful for the course. The majority (14 out of 17) of the answers showed interest in the system. The 3 answers that did not reveal interest in the CBA system were related to the courses Agents and Distributed Artificial Intelligence (2 of them) and Formal Methods in Software Engineering. Table 4.1 presents some information about the courses indicated in the positive answers: *Count* represents the number of answers received for a given course, *Year* represents the year in which the course is positioned at the Integrated Masters syllabus and *Area* identifies the study field.

Course	Count	Year	Area
Algorithms and Data Structures	2	2	Programming
Compilers	1	3	Programming
Computer Graphics	1	2	Interaction and Multimedia
Computing Theory	1	2	Programming Fundamentals
Databases	1	3	Information Systems
Graphical Applications Laboratory	1	3	Interaction and Multimedia
Information Systems and Databases (MIEEC)	1	4	Information Systems
Logic Programming	1	3	Programming
Operating Systems	1	2	Operating Systems and Networks
Programming	1	1	Programming
Programming 1 (MIEEC)	1	1	Programming
Programming Fundamentals	2	1	Programming Fundamentals

Table 4.1: Course information of positive answers

As can be seen in figure 4.1, almost all of the courses are from the first Integrated Masters cycle: 3 are from the first year, 4 from the second and 4 from the third. Only the course Information Systems and Databases is placed in the second cycle, being taught in the fourth year of MIEEC.

Figure 4.2 summarizes the information related to the study areas of the courses. Not surprisingly, most of them involve the teaching of programming (5) and its fundamentals (2). However, the CBA system was also considered to be useful for 1 course on operating systems, 2 courses on information systems involving the teaching of databases and 2 courses on interaction and multimedia involving the teaching of computer graphics.

#### 4.1.2 Assessment Creation

The CBA system can be used to support different types of assessment. Moreover, each assessment should be configured with a set of parameters. Therefore, the objectives of the second section of the survey were to understand the types of assessment which the



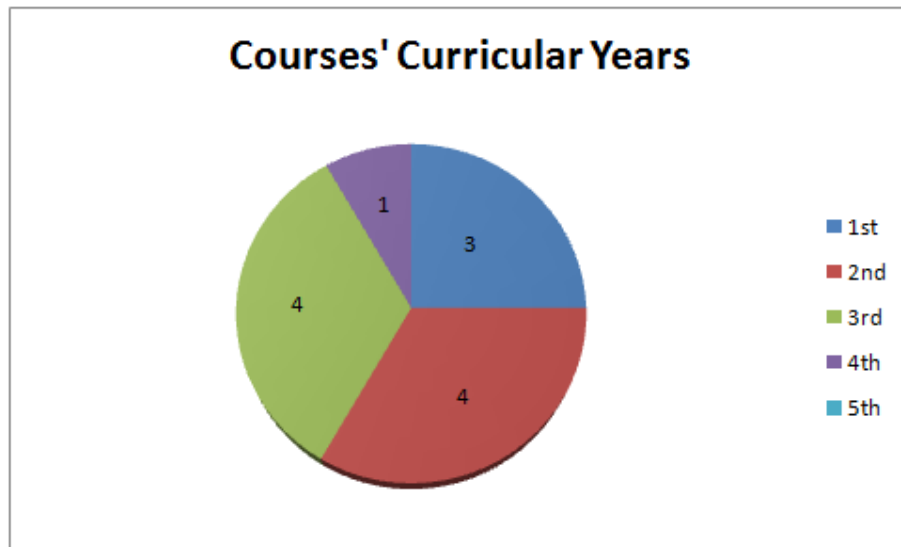


Figure 4.1: Curricular years of the positive answers' courses

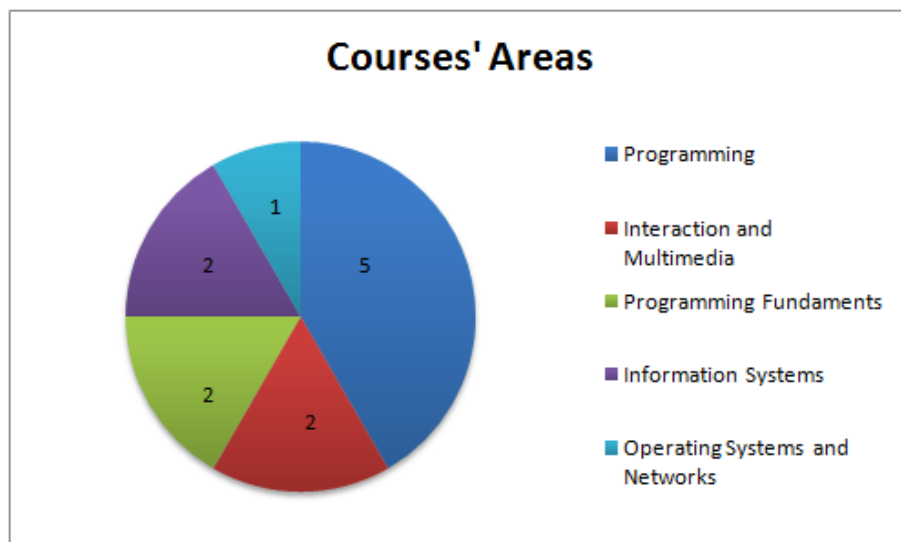


Figure 4.2: Study areas of the positive answers' courses

teaching staff considers to be more important to be supported by the system and also to identify the main configuration parameters.

The teaching staff was asked to rate the usefulness, from 1 to 5, of three different types of assessment: exams, tests and self-learning exercises. As can be seen on table 4.2, exams received, on average, higher ratings. However, the difference between the three types of assessment is not significant and the mode is even the same.

The survey also contained an open question for indicating the set of parameters considered to be more important for configuring a new assessment. The collected parameters are presented, ordered by popularity, in table 4.3. In this open question it was also suggested, by a Databases teacher, that the system should support the automatic assessment

## Analysis of Learning Needs

	Average	$\sigma$	Mode
Exams	4.3	1.1	5
Tests	4.1	1.2	5
Self-learning exercises	3.9	1.1	5

Table 4.2: Usefulness of the different assessment types

of SQL exercises in addition to the support of traditional programming languages.

Parameter	Count
Start time	10
Duration	10
Accepted programming languages	8
Tolerance time (and associated penalization)	8
Maximum number of submissions	3
Duration of each individual question	1
Toggle immediate/non-immediate feedback	1
Template for solutions	1

Table 4.3: Parameters for assessment configuration

### 4.1.3 Test-Cases Definition and Automatic Assessment

The third part of the survey had the goal of evaluating the test-case definition methodologies and test-case assessing parameters that the teaching staff would like to use. The staff could vote in three suggested methodologies and, optionally, indicate a new one. The suggestions were: definition of text files with the input and output test data, usage of regular expressions for automatic generation of inputs and outputs and definition of unit tests. Table 4.4 presents the number of votes received by each one. As can be seen, the use of text files and unit tests were the most popular choices, leaving the use of regular expressions at a considerable distance.

Methodology	Votes
Text files	12
Regular expressions	6
Unit tests	13

Table 4.4: Votes on the different test-cases definition methodologies

The survey also requested the staff to rate the importance, from 1 to 5, of 4 test-case assessing parameters: weight for the final grade, maximum runtime, maximum memory and automatic static analysis of the quality of the code. The average, standard deviation and mode of the ratings are presented in table 4.5. Not surprisingly, the possibility of

defining the weight of the test-cases received high ratings. It is important to note that the maximum runtime and memory parameters were only considered relatively significant.

	<b>Average</b>	$\sigma$	<b>Mode</b>
<b>Weight</b>	4.6	0.6	5
<b>Max runtime</b>	3.1	1.0	4
<b>Max memory</b>	2.7	1.2	4
<b>Static analysis</b>	3.6	1.2	4

Table 4.5: Importance of different test-case parameters

Besides the poll for the 4 suggested parameters, the respondents could suggest new assessing parameters. The suggestions included the possibility of:

- Distinguishing student grades based on the time they take to solve each problem. As an example let's take an assessment composed by 5 problems. The system can be used to set that the final grade of students who answer correctly the 5 problems will be in the range [18, 20]. The definitive placement in the range can be based on the time spent on solving the problems;
- Ad-hoc definition of new assessment parameters;
- Associating a set of different solutions to a test case, each one contemplating one common mistake and with a predefined penalization.

It was also suggested, by computer graphics teaching staff, that the system should be able to automatically generate views to help with the assessment of the graphical look of solutions for graphical exercises.

#### 4.1.4 Elaboration and Submission of Solutions

In this section, the participants were requested to vote in one of two alternatives regarding the environment where the students should develop their assessment solutions: directly in the CBA system by using a built-in editor and compiler or locally, in an editor and compiler of students' choice. As can be seen in table 4.6, the latter option is by far the most popular.

<b>Alternative</b>	<b>Votes</b>
Directly in the CBA system	3
Locally	11

Table 4.6: Votes on the solution development alternatives

The survey also requested for the rating on the usefulness, from 1 to 5, of a mechanism for allowing the teaching staff to define and make available skeleton solutions for

the assessments. The results are presented in table 4.7. Although this feature was not considered a top priority, the results point out that it can be relatively useful.

	Average	$\sigma$	Mode
<b>Skeleton mechanism</b>	3.6	1.2	4

Table 4.7: Usefulness of a mechanism for definition of skeleton solutions

#### 4.1.5 Feedback System

The provision of useful feedback should be one of the main concerns of the CBA system. The fifth section of the survey involved two questions regarding the feedback system. In the first one, the teaching staff was requested to vote in the type of feedback they would like to be made available by the CBA system. They could vote in at least one of three levels of feedback:

- **Minimalist** - the students only receive the grade they obtained in the assessment;
- **Moderated** - the feedback includes the grade and, for each test case, if whether or not it was accepted. In the case of a non-accepted test case, error messages such as `Wrong Answer`, `Time Limit Exceeded` or `Runtime Error` may be presented;
- **Detailed** - for each submission the final grade is presented and for each test case the input and the expected and obtained output are presented. The other assessment components, like the results of the static analysis of the code, may also be presented.

As shown in table 4.8, the detailed feedback alternative is the most popular one with 12 votes, followed by the moderated with 8 votes. The minimalist option only gathered 3 votes.

Feedback level	Votes
Minimalist	3
Moderated	8
Detailed	12

Table 4.8: Votes on the different levels of feedback

The participants were also requested to rate the usefulness, from 1 to 5, of a mechanism for allowing the association of personalized feedback messages to test cases. This mechanism can be used to associate messages to incorrect answers representing common mistakes. The results are shown in table 4.9.

The answers were far from being concordant, as can be seen by the relatively high standard deviation. Moreover, although the average of the results was only 3.6, the mode

	<b>Average</b>	$\sigma$	<b>Mode</b>
<b>Personalized feedback</b>	3.6	1.5	5

Table 4.9: Usefulness of a mechanism for personalized feedback messages

was 5. This allows to conclude that the personalized feedback mechanism may be useful on some specific situations.

#### 4.1.6 Submission Management

In the sixth part of the survey, the respondents were asked to rate, from 1 to 5, the usefulness of a set of features related to the management of students' submissions:

- Consulting of the history of submissions and grades of each student;
- Notification (automatic and/or manual) of students who did not submit their solutions;
- Manual alteration of the code submitted by students;
- Manual alteration of the results of the automatic assessment;
- Sending of e-mails to groups of students, based on assessment criteria.

The results are presented in table 4.10 and, as can be seen, the most popular feature is the fourth one, followed by the first and the second.

<b>Feature</b>	<b>Average</b>	$\sigma$	<b>Mode</b>
<b>Student History</b>	3.9	0.8	4
<b>Student Notification</b>	3.9	1.2	4
<b>Alteration of submitted code</b>	3.3	1.4	4
<b>Alteration of assessment results</b>	4.3	0.7	5
<b>Sending of e-mails</b>	3.8	1.1	3

Table 4.10: Usefulness of features related to the management of solutions

The respondents could also suggest some features in a text box. The received suggestions include the following features: exporting of the assessment results to different formats, direct links to the submissions and/or built-in environment for manual testing of the submissions.

#### 4.1.7 Extra Features

The last part of the survey had the purpose of evaluating the potential usefulness of a set of features that, although not essential for the functioning of the system, may add value to the CBA system. The participants were asked to rate, from 1 to 5, the following features:

- Maintenance of a database with all the exercises created throughout all the curricular years and accessible to new students;
- Ranking of the submissions of all the students, with statistics like runtime and memory used;
- Mechanism for detecting plagiarism among the submissions of the students;
- Built-in Q&A system for doubts clarification.

The rating results are presented in table 4.11. The participants considered the plagiarism detection mechanism to be a quite important feature. The Q&A system and the exercises database also received high ratings, while the maintenance of a ranking was only considered relatively important.

Feature	Average	$\sigma$	Mode
Exercises database	3.9	1.2	4
Submissions ranking	3.1	1.1	4
Plagiarism detection	4.6	0.6	5
Q&A system	4.1	0.8	4

Table 4.11: Usefulness of some suggested extra features

#### 4.1.8 Final Comments and Observations

At the end of the survey, the participants were asked to make additional comments and observations. In some of the comments it was referred that a human assessment of the code is indispensable as a complement to the automatic assessment. The following justifications were given:

- Being a valued and requested process in the labor market, it is important to optimize the submitted solutions. However, it is hard to create an exam exercise to allow for the automatic evaluation of the optimization process and hence a human inspection of the code is needed;
- The true assessment of knowledge implies evaluating the whole reasoning process and not only the final output and hence assessment of code style and quality should be handled by manual evaluation.

Some opinions expressed concern about the flexibility and usability of the system. It was referred that the system should not have too many configuration steps since it may lead to usability issues and make it unattractive.

## Analysis of Learning Needs

It was also referred that it should be possible to students to compile and validate their solutions against some test cases, before submitting their final answers. This will lead to the reduction of compilation and interpretation problems in the final solutions.

Finally, there was an opinion suggesting that the CBA system should be integrated with SIGEX, a system which is used at FEUP to support the realization of computer exams. SIGEX allows controlling the resources that students can access during exams and manages the distribution of the exam statement and the submission of solutions.

## Analysis of Learning Needs



## Chapter 5

# Specification of a New CBA System

This chapter contains the specification of the new CBA system, which has the goal of fulfilling the needs identified in the previous chapter. The first section contains an overview of the system, consisting of a description of the desired behavior and architecture, as well as the features that should be implemented. Then, a study that was conducted with the goal of evaluating the possibility of reusing one of the systems analyzed in chapter 2 is presented. Then, and because one of the requirements was to integrate the system with Moodle, there was the need of deciding which type of Moodle plugin to use. Since the base system could not fulfill all the requirements of the new system, there is a brief discussion of how and which features should be built on top of it. Finally, the architecture of the CBA system was detailed.

### 5.1 New System Proposal

The new CBA system should have a core component, behaving as an independent service, responsible for the automatic assessment of programming assignments. It can, possibly, be based on one of the systems analyzed in chapter 2 and should be flexible enough to communicate with other platforms, responsible for implementing the user interface. The platforms of interest for DEI teaching staff are:

- Moodle, which is the e-Learning platform used at FEUP. This should be the platform with higher priority;
- SIGEX, a system used to support the realization of computer exams;
- An eBook system, containing programming exercises;

The system should cover the learning needs identified in the previous chapter. In the next subsection, these needs are described via use case diagrams, which illustrate the main features that should be supported, by showing who can do what with the system. Then, the features are presented in more detail, in a subsection containing prioritized lists.

### 5.1.1 Use Cases

This subsection presents the main use cases of the CBA system. As can be seen in figure 5.1, they were grouped in three different packages: Assessments Management, Submissions Management and Assessments Solving. Two actors can be identified: the teaching staff, involved in the first two packages, and students, involved in the last package.

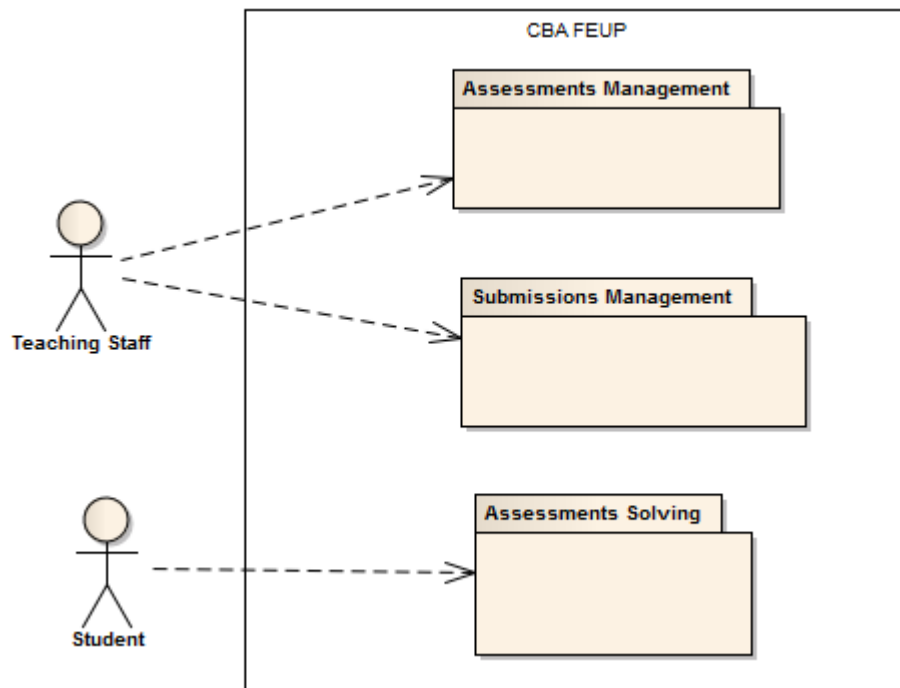


Figure 5.1: Use cases overview

#### 5.1.1.1 Assessments Management

The package Assessments Management, presented in figure 5.2, contains the use cases related to the management of assessments. Teaching staff can create, edit and remove assessments. Associated to the creation and edition of assessments there is the definition of assessment parameters, feedback detail, personalized feedback messages, skeleton solutions and test cases.

#### 5.1.1.2 Assessments Solving

The package Assessments Solving, which can be consulted in figure 5.3, includes all the use cases involving students. When solving an assessment, they can consult its statement, submit a solution and consult the submission grade and feedback.

## Specification of a New CBA System

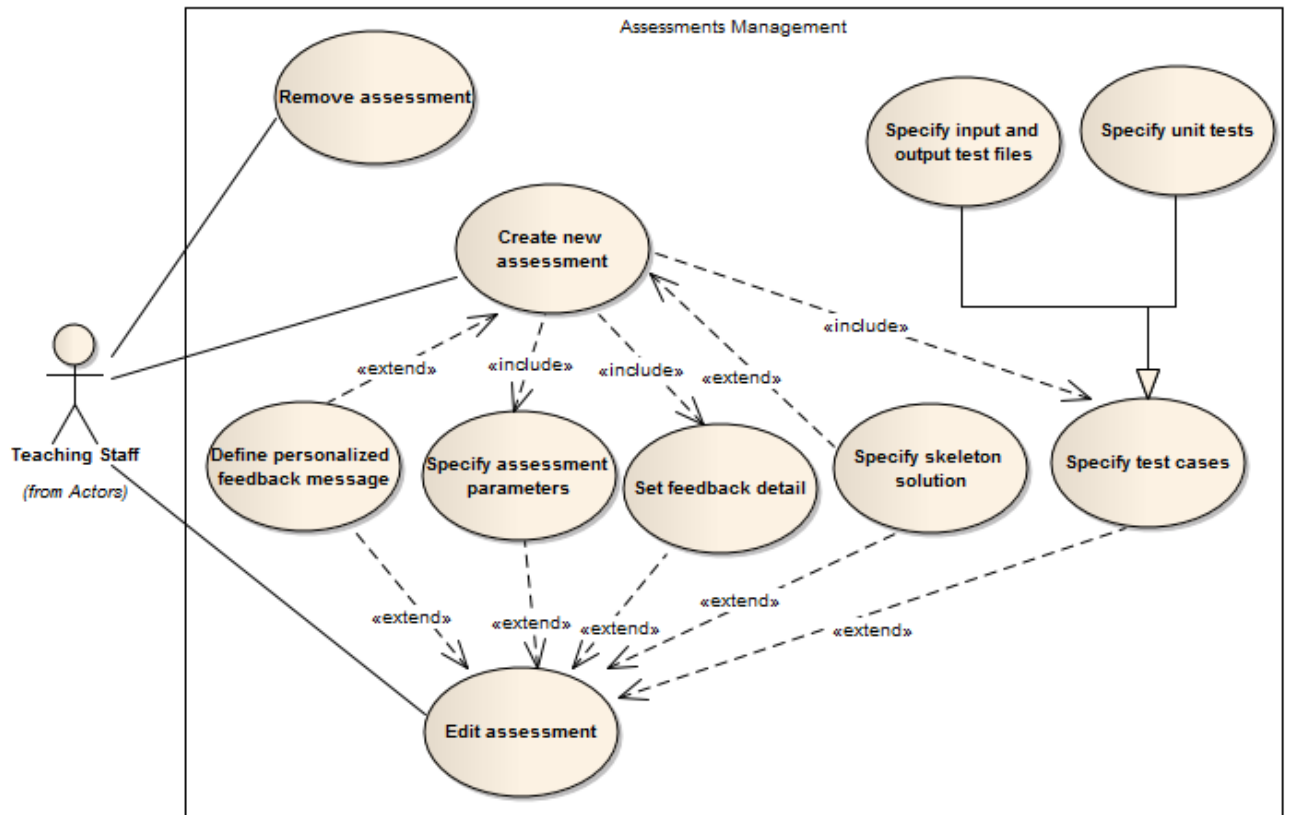


Figure 5.2: Assessments Management use cases

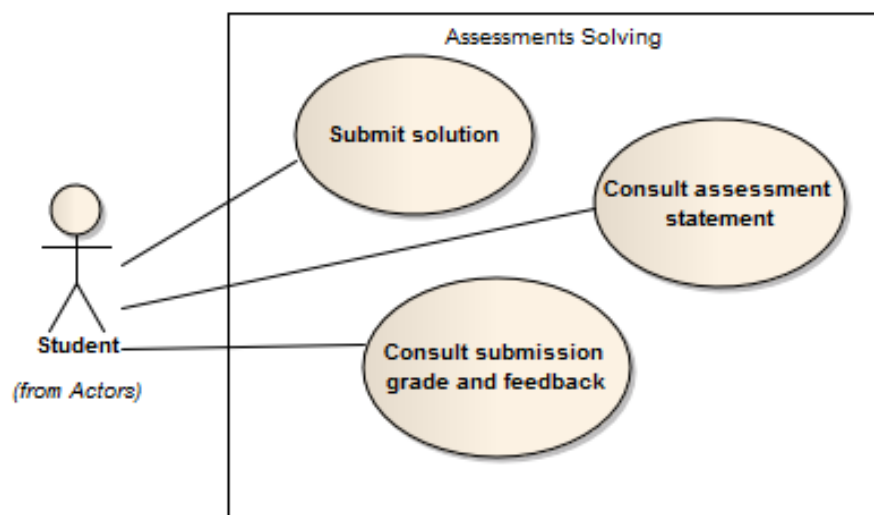


Figure 5.3: Assessments Solving use cases

### 5.1.1.3 Submissions Management

The package Submissions Management, shown in figure 5.4, contains the use cases related to the management of students' solutions. These use cases are based on the features discussed in 4.1.6 and include the sending of e-mail based on assessment criteria, the notification of students, the consulting of assignment submissions and student history, and the manual modification of automatic assessment results and submitted code.

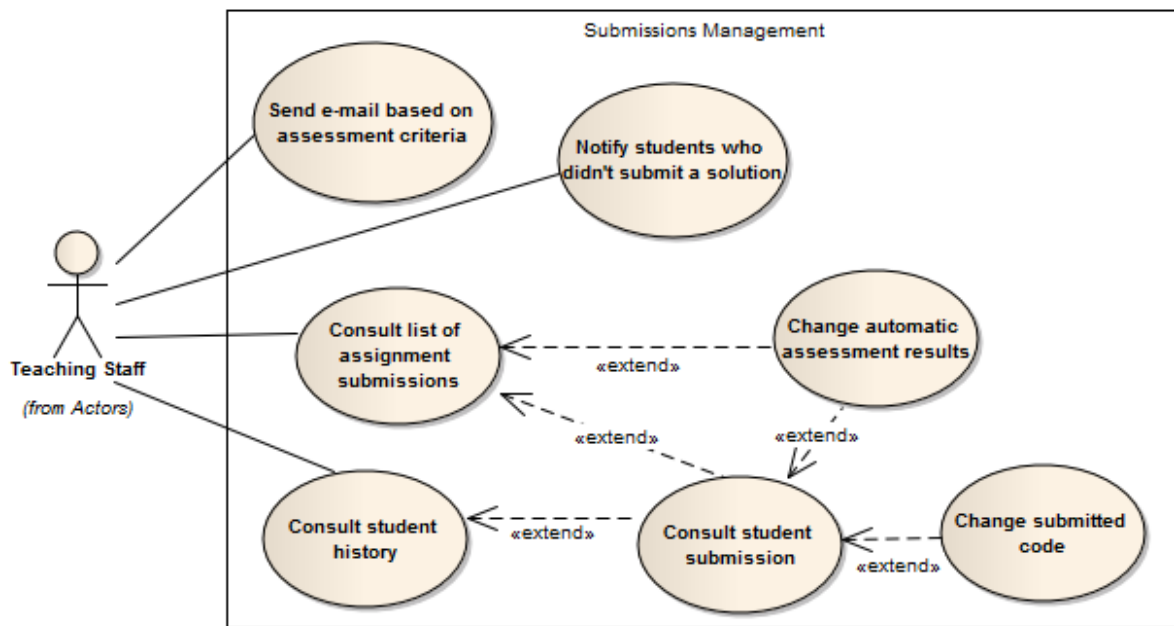


Figure 5.4: Submissions Management use cases

## 5.1.2 Features to Implement

The identification of the features to implement was based on the analysis made in section 2.9, on the results of the survey performed next to the DEI teaching staff and presented in section 4.1 and on the use cases presented in section 5.1.1. Each feature has an associated priority (high, medium or low) and an id.

### 5.1.2.1 Support for Assessment Creation

As can be observed in table 5.1, the system should support the realization of exams and tests with an high priority and the realization of self-learning exercises with medium priority.

## Specification of a New CBA System

Id	Assessment	Priority
F1.1	Exams	High
F1.2	Tests	High
F1.3	Exercises	Medium

Table 5.1: Types of supported assessments

Table 5.2 contains the assessment configuration parameters that should be implemented. Five of them were classified with high priority - start time, deadline, list of accepted programming languages, tolerance time and associated penalization, global duration of the assessment -, one with medium priority - maximum number of allowable solutions - and another one with low priority - duration of each individual question.

Id	Parameter	Priority
F2.1	Start time	High
F2.2	Deadline	High
F2.3	Accepted programming languages	High
F2.4	Tolerance time	High
F2.5	Duration	High
F2.6	Max number of submissions	Medium
F2.7	Duration of each individual question	Low

Table 5.2: Parameters for assessment configuration

### 5.1.2.2 Support for Test-Cases Definition

Two of the identified methodologies for test-cases definition were classified as having high priority: definition of text files with input and output test data and definition of unit tests. The usage of regular expressions for automatic generation of inputs and outputs was considered to have low priority.

Id	Methodology	Priority
F3.1	Text files	High
F3.2	Unit tests	High
F3.3	Regular expressions	Low

Table 5.3: Methodologies for test-cases definition

Table 5.4 contains the test-case assessing parameters that should be implemented. One was classified with high priority - definition of the weight of each test case in the assessment grade -, three were classified with medium priority - automatic static analysis of code quality and definition of the maximum running time and maximum allowable memory - and two were classified as having low priority - distinguishing student grades based on the time they take to solve each problem and associating a set of different solutions to

a test case, each one contemplating one common mistake and with a predefined penalization.

<b>Id</b>	<b>Parameter</b>	<b>Priority</b>
F4.1	Weight for assessment grade	High
F4.2	Static analysis of code quality	Medium
F4.3	Max runtime	Medium
F4.4	Max memory	Medium
F4.5	Distinguish students grades based on solving time	Low
F4.6	Identification and penalization of common mistakes	Low

Table 5.4: Parameters for test-case configuration

### 5.1.2.3 Support for Elaboration and Submission of Solutions

Table 5.5 contains the features related to the elaboration and submission of solutions that should be made available by the CBA system.

The utilization of a built-in editor and compiler for allowing the students to develop their assessment solutions directly in the CBA system proved to be an unpopular alternative. Therefore, a mechanism for allowing the uploading of solutions developed in a local environment is essential.

The implementation of a mechanism for allowing the teaching staff to define and make available skeleton solutions for the assessments was classified as having medium priority.

<b>Id</b>	<b>Feature</b>	<b>Priority</b>
F5.1	Mechanism for uploading solutions	High
F5.2	Skeleton mechanism	Medium

Table 5.5: Features for supporting the elaboration and submission of solutions

### 5.1.2.4 Feedback System

Table 5.6 presents the features related to the provision of feedback to students. The system should allow defining, with high priority and for a specific assessment, whether or not the feedback given to students is immediate. Moreover, and also with high priority, it should be possible to regulate the level of feedback that is automatically generated. Ideally, there should be three feedback levels: minimalist, moderated and detailed. The third feature concerning the feedback system is related to the provision of personalized feedback messages and has medium priority. The personalized feedback mechanism should allow associating manual messages to incorrect answers representing common mistakes

## Specification of a New CBA System

<b>Id</b>	<b>Feature</b>	<b>Priority</b>
F6.1	Toggle immediate/non-immediate feedback	High
F6.2	Regulation of the level of feedback (Minimalist, Moderated, Detailed)	High
F6.3	Personalized feedback messages	Medium

Table 5.6: Features related to the feedback system

### 5.1.2.5 Submissions Management

The features related to the management of submissions are shown in table 5.7. Four were classified as having high priority - consulting of statistics of assessment grades, manual alteration of the code submitted by students, consulting of the history of submissions and grades of each student and notification of students who did not submit their solutions -, two were classified with medium priority - sending of e-mails to group of students based on assessment criteria and manual alteration of the code submitted by students - and another two were classified with low priority - exporting of the assessment results to different formats and access to a built-in environment for manual testing of the submissions.

<b>Id</b>	<b>Feature</b>	<b>Priority</b>
F7.1	Consult grades statistics	High
F7.2	Manual alteration of assessment results	High
F7.3	Student history	High
F7.4	Student notification	High
F7.5	Sending of e-mails	Medium
F7.6	Alteration of submitted code	Medium
F7.7	Exporting of assessment results	Low
F7.8	Built-in environment for manual testing of submissions	Low

Table 5.7: Features related to the management of submissions

### 5.1.2.6 Extra Features

Table 5.8 presents the features that were identified as being extra, i. e. features that were not considered essential for the functioning of the system but add value to it. Having a mechanism for detecting plagiarism among students submissions were classified with high priority. Having a built-in Q&A mechanism for doubts clarification and a database with all the exercises created throughout all the curricular years were considered to have medium priority. The implementation of a ranking with the submissions of all the students with statistics like runtime and memory used was classified with low priority.

## Specification of a New CBA System

<b>Id</b>	<b>Feature</b>	<b>Priority</b>
F8.1	Plagiarism detection	High
F8.2	Q&A system	Medium
F8.3	Exercise Database	Medium
F8.4	Submissions ranking	Low

Table 5.8: Extra features

## 5.2 Reusing an Existing System

Having the list of features to implement and the summary of the features available in each of the analyzed CBA systems, the next step was to decide whether or not one of the existing systems could be used as the basis for the desired one. To support the decision, three tables were created: table 5.9, table 5.10 and table 5.11. They present the features to implement with, respectively, high, medium and low priority already available in the existing CBA systems.

<b>Feature</b>	<b>CBA System</b>							
	<b>CourseMarker</b>	<b>BOSS</b>	<b>xLx</b>	<b>Mooshak</b>	<b>RoboProf</b>	<b>Submit!</b>	<b>GAME</b>	<b>DOMjudge</b>
F1.1 Exams	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
F1.2 Tests	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
F2.1 Start time	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
F2.2 Deadline	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
F2.3 Accepted programming languages	No	No	No	Yes	No	No	No	Yes
F2.4 Tolerance time	No	No	No	No	No	No	No	No
F2.5 Duration	No	No	No	No	No	No	No	No
F3.1 Text files	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
F3.2 Unit tests	No	Yes	Yes	No	No	No	No	No
F4.1 Weight for assessment grade	No	Yes	Yes	Yes	No	Yes	No	No
F5.1 Mechanism for uploading solutions	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
F6.1 Toggle immediate/non-immediate feedback	No	No	No	No	No	No	No	Yes
F6.2 Regulation of the level of feedback	Yes	No	No	No	No	No	No	No
F7.1 Consult grades statistics	Yes	No	No	No	No	Yes	No	No
F7.2 Manual alteration of assessment results	No	Yes	No	No	No	Yes	No	No
F7.3 Student history	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
F7.4 Student notification	Yes	Yes	No	No	No	No	No	No
F8.1 Plagiarism detection	Yes	Yes	No	No	Yes	No	Yes	No
<b>Total</b>	<b>11</b>	<b>12</b>	<b>8</b>	<b>9</b>	<b>7</b>	<b>10</b>	<b>8</b>	<b>9</b>

Table 5.9: Features with high priority



## Specification of a New CBA System

Feature	CBA System							
	CourseMarker	BOSS	xLx	Mooshak	RoboProf	Submit!	GAME	DOMjudge
F1.3 Exercises	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
F2.6 Max number of submissions	Yes	No	No	No	No	No	No	No
F4.2 Static analysis of code quality	Yes	Yes	No	No	No	Yes	Yes	No
F4.3 Max runtime	Yes	No	No	Yes	No	No	No	Yes
F4.4 Max memory	No	No	No	Yes	No	No	No	No
F5.2 Skeleton mechanism	Yes	No	No	No	No	No	No	No
F6.3 Personalized feedback messages	No	No	No	No	No	No	No	No
F7.5 Sending of e-mails	No	No	No	No	No	No	No	No
F7.6 Alteration of submitted code	No	Yes	No	No	No	No	No	No
F8.2 Q&A system	Yes	No	No	Yes	No	No	No	Yes
F8.3 Exercise Database	No	No	No	No	No	No	No	No
<b>Total</b>	<b>6</b>	<b>3</b>	<b>1</b>	<b>4</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>3</b>

Table 5.10: Features with medium priority

Feature	CBA System							
	CourseMarker	BOSS	xLx	Mooshak	RoboProf	Submit!	GAME	DOMjudge
F2.7 Duration of each individual question	No	No	No	No	No	No	No	No
F3.3 Regular expressions	No	No	No	No	No	No	No	No
F4.5 Distinguish grades based on solving time	No	No	No	No	No	No	No	No
F4.6 Penalization of common mistakes	No	No	No	No	No	No	No	No
F7.7 Exporting of assessment results	No	No	No	No	No	No	No	No
F7.8 Environment for manual testing of subm.	No	No	No	No	No	No	No	No
F8.4 Submissions ranking	No	No	No	Yes	No	No	No	Yes
<b>Total</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>

Table 5.11: Features with low priority

The tables show that the existing systems already implement a significant amount of high priority features and a few of the medium and low priority ones. Therefore, it was considered to be advantageous to reuse one system instead of implementing a new system from scratch.

By inspecting table 5.9 it is possible to conclude that the CourseMarker and BOSS systems are the ones that have a higher number of already implemented features. However, CourseMarker was ruled out since it is a paid system and cannot be extended with

the features that it lacks. The BOSS system was also ruled out as a viable alternative for the base system, despite being an open-source project. After installing and testing BOSS, it was noted that the core functionality of the system - the automatic assessment of code - was not as mature and user-friendly as desired.

Since the maturity of the automatic assessment of code components seemed to be a problem of the systems developed specifically to be used at Universities, the 2 systems created for programming contests, Mooshak and DOMjudge, were considered to be the more viable alternatives.

### 5.2.1 Mooshak versus DOMjudge

Mooshak and DOMjudge are both open source, already implement 9 out of 18 of the high priority features and 1 out of 7 of the low priority features. In terms of medium priority features, Mooshak has one more than DOMjudge. By analysing both systems' source code and documentation, the following observations were drawn:

- DOMjudge uses a MySQL database for keeping contest and submission data, while Mooshak uses the file system;
- DOMjudge has, in its architecture, a clear separation of the components responsible for running the contest and the components responsible for automatically assessing the submitted solutions. There are even two different programs that can (and should) be executed on different machines;
- DOMjudge has more documentation available, including detailed Administrator, Jury and Team manuals.

Based on this analysis, DOMjudge was chosen over Mooshak as the base system. Since, for the desired CBA system, the features related to the running of contests are irrelevant, the fact that DOMjudge makes a clear distinction between those features and the automatic assessment of code, allows for a more straightforward creation of an abstraction layer responsible for the communication with external platforms. This abstract layer can directly access DOMjudge's database and be independent of DOMjudge's source code. The components responsible for the automatic assessment of code have an autonomous behavior, use the same database, but execute on different machines.

Besides the robust automatic assessment of code component, DOMjudge has also proved to be a very safe system, being used with success in some important programming contests. Furthermore, and as referred in section 2.8.1, it allows for the definition of *validators* that may be useful for handling data that do not fit within the standard scheme of fixed test data.

### 5.3 Integration With Moodle

The easiest and most maintainable for adding new functionality to Moodle is by using one of the many plugin APIs available [Moo10b]. Three of the different types of plugin available were identified as viable possibilities for making the bridge between the server responsible for the automatic assessment and Moodle: a new activity module, a new type of assignment or a new question type. Each of the three alternatives has its up and down points. This section has the goal of evaluating those ups and downs and of justifying the type of plugin that was chosen. The decision was made based on 6 factors: implementation freedom, implementation effort, the number of configuration parameters already available in each of the alternatives, the existence of a pre-implemented upload mechanism, vulnerability to future changes in Moodle's structure and, finally, the way programming problems can be reused.

#### 5.3.1 Implementation Freedom

The creation of a new activity module allows for much more freedom in the definition of the functionalities needed on the Moodle side since almost all of the user interface can be implemented from scratch. The level of freedom decreases for the implementation of a new assignment type, since one has to follow the restrictions outlined by the structure of the assignment template. When it comes to the question type, the implementation level decreases even more: one has to follow the restrictions outlined by the structure of the question type template and has to cope with the quiz structure already implemented in Moodle. Therefore, and as can be seen in figure 5.12, the implementation freedom can be classified as low, medium and high for, respectively, the question type plugin, the assignment plugin and the activity module plugin.

	Question	Assignment	Module
<b>Implementation freedom</b>	Low	Medium	High

Table 5.12: Implementation freedom of the different plugin alternatives

#### 5.3.2 Implementation Effort

By analyzing the source code of existing question types, assignment types and activity modules, the implementation effort of the different alternatives was classified as shown on table 5.13. The implementation of a new activity module requires the definition of a much higher number of and more complex source files than the implementation of a new question or assignment.

	Question	Assignment	Module
<b>Implementation effort</b>	Medium	Medium	High

Table 5.13: Implementation effort of the different plugin alternatives

### 5.3.3 Configuration Parameters

Opting for a new question type allows for the use of a big set of assessment configuration parameters already implemented in Moodle's quizzes. The alternative of the assignment type also allows for the utilization of a pre-implemented set of configuration parameters. However, the amount is smaller than the available for quizzes. As expected, there are no configuration parameters pre-implemented in the case of the activity module, since everything has to be implemented from scratch. Table 5.14 summarizes these conclusions about the configuration parameters.

	Question	Assignment	Module
<b>No. configuration parameters</b>	High	Medium	N/A

Table 5.14: Number of pre-implemented configuration parameters for the different plugin alternatives

### 5.3.4 Existence of an Upload Mechanism

The creation of a new assignment type allows for the use of a pre-implemented mechanism for the upload of assignment solutions. To achieve this, one has only to extend the already available assignment types `Advanced uploading of files` or `Upload a single file`. The implementation of a new question type demands for the implementation of an upload mechanism. Although this problem can be outlined by making use of a text field where the students could paste their code, if multiple source code files are needed, then the upload mechanism is imperative. As expected, there is no pre-implemented upload mechanism in the case of the activity module, since everything has to be implemented from scratch. Table 5.15 summarizes these conclusions about the upload mechanism.

	Question	Assignment	Module
<b>Upload mechanism</b>	No	Yes	No

Table 5.15: Availability of a pre-implemented mechanism for upload of solutions in the different plugin alternatives

### 5.3.5 Vulnerability to Structural Changes

One of the main factors influencing the final decision about which type of plugin to chose is the vulnerability to future changes in Moodle's structure. It is important for the maintainability of the CBA system that changes in future versions of Moodle will not affect the normal behavior of the plugin. Obviously, it is not possible to make accurate previsions about this vulnerability factor, since it is not possible to predict the structure of future Moodle versions and also because the way the plugin is implemented, independently of its type, may also affect forward compatibility. However, some previsions can still be made.

As can be seen on table 5.16, the vulnerability of the question type, assignment type and activity module were respectively classified as Medium/High, Medium and Low. The question type is probably the most vulnerable since it is dependent on the current structure of quizzes and question types. If one of these two is changed the plugin may be affected. The assignment type may be vulnerable to future changes in the assignment template, while the activity module may be vulnerable to future changes in the activity module templates, which is less likely to happen due to the considerable amount of pre-existing modules that would have to be changed.

	Question	Assignment	Module
<b>Vulnerability to structural changes</b>	Medium/High	Medium	Low

Table 5.16: Vulnerability to future Moodle structural changes of the different plugin alternatives

### 5.3.6 Reuse of Problems

It is also important to analyze how the three different alternatives allow for the reuse of previously created problems. In the situation of the question type, different questions can be used for different assignments. This feature is already implemented in Moodle. However, when it comes to the assignment type or the activity module, there is no such feature already available. This should be implemented in the server side, where the problems are maintained. Table 5.17 summarizes these conclusions about the means for reusing problems.

	Question	Assignment	Module
<b>Reuse of problems</b>	Moodle	Server	Server

Table 5.17: Means for reusing problems of the different plugin alternatives

### 5.3.7 Making a Decision

The conclusions drawn about each of the plugin types can be summarized as follows:

- **Question type** - this is the option that requires less implementation effort and that takes more advantage of features already implemented in Moodle. However, this is probably the worst choice in terms of maintainability and implementation freedom;
- **Assignment** - the assignment alternative can be considered to lie somewhere in between the other two alternatives. It takes advantage of some already implemented features, but less than the ones available for the question type. The implementation freedom is higher than the question type but lower than the activity module;
- **Activity Module** - this alternative is the most viable one in terms of implementation freedom and maintainability. In spite of requiring more implementation effort, it is the one less coupled to Moodle's structure and the one more likely to support future changes in the CBA system structure;

Based on all the collected information, it was decided to give a higher priority to the implementation of a new activity module. The implementation freedom and maintainability factors played a very important role in the decision.

The implementation of a new question type was not discarded. However, it was given a lower priority than the activity module. The new question type has advantages in terms of configuration parameters and reuse of problem. Moreover, it will allow the creation of assignments with programming exercises combined with other types of exercises.

The creation of a new assignment type was ruled out because its main advantages are feature related. The already implemented features, that are of interest for the system, can also be implemented in the activity module.

## 5.4 From Programming Contests to University Assignments

Being specifically developed to be used in programming contests, the choice of DOMjudge to be the base system raises a number of conceptual issues. The entities involved in a programming context are not exactly the same ones involved in a teaching context and hence a mapping of entities between the two contexts is needed. For instance, the entities `student` and `assessment` can be represented by the DOMjudge entities `team` and `problem`. This means that there should be a layer responsible for linking and adapting DOMjudge's functionalities to be used by the Moodle plugin.

It is also important to note that DOMjudge does not provide all the functionalities specified for the final system. As presented in section 5.2, it supports 9 out of 18 features with high priority, 3 out of 11 features with medium priority and 1 out of 7 features with low priority. Some of the missing features can be exclusively implemented in the Moodle plugin, while others may be implemented by using additional external tools. For instance, features F3.2 - Unit tests, F4.2 - Static analysis of code quality

and F8.1 - Plagiarism detection may be implemented by linking out-of-the-shelf tools to the Moodle plugin for, respectively, performing unit tests, static analysis of code and detect plagiarism in source files.

By looking at the set of features not supported by DOMjudge, one can conclude that it mostly lacks the ones listed in section 5.1.2.5. Those are submission management related features that are associated to the teaching context and are not used in programming contests. Another characteristic of programming contests, in which DOMjudge is used, is that a problem is only considered to be correct if it passes all the test cases. In the teaching context it is important to accept partially correct programs capable of solving a part of the test cases, which may have different weights.

The fact that DOMjudge's automatic assessment of code relies on input and output processing may pose some issues in introductory programming courses, where students still do not master the handling of input and output data. However, this can be overcome with feature F5.2 - Skeleton mechanism. Teaching staff can provide skeleton code, responsible for handling the input and output data, allowing students to focus on writing the code for problem solving. Therefore it is reasonable to change the priority of this feature from Medium to High.

## 5.5 Architecture

This subsection introduces the system's architecture by the means of UML diagrams. The architecture of the base system, DOMjudge, is presented in first place. Then, the overall architecture of the system is discussed. The overall architecture was define by linking Moodle (the platform with higher priority as referred in 5.1) to the automatic assessment server and by adding some utilitarian external tools.

### 5.5.1 DOMjudge's Architecture

DOMjudge's architecture, briefly introduced in section 2.8.6, can be represented by the deployment model of figure 5.5.

As can be seen, there are three different kinds of physical nodes:

- **Client** - represents the machines where the teams compete. Teams can access the main server, hosted by the Domserver machine, either via a web browser or via a submit client which is a command line client for submitting solutions;
- **Domserver** - this node hosts the main server, which runs on Apache, receives submissions, runs a MySQL database for keeping the submissions and serves web pages to teams, jury and administrators;

## Specification of a New CBA System

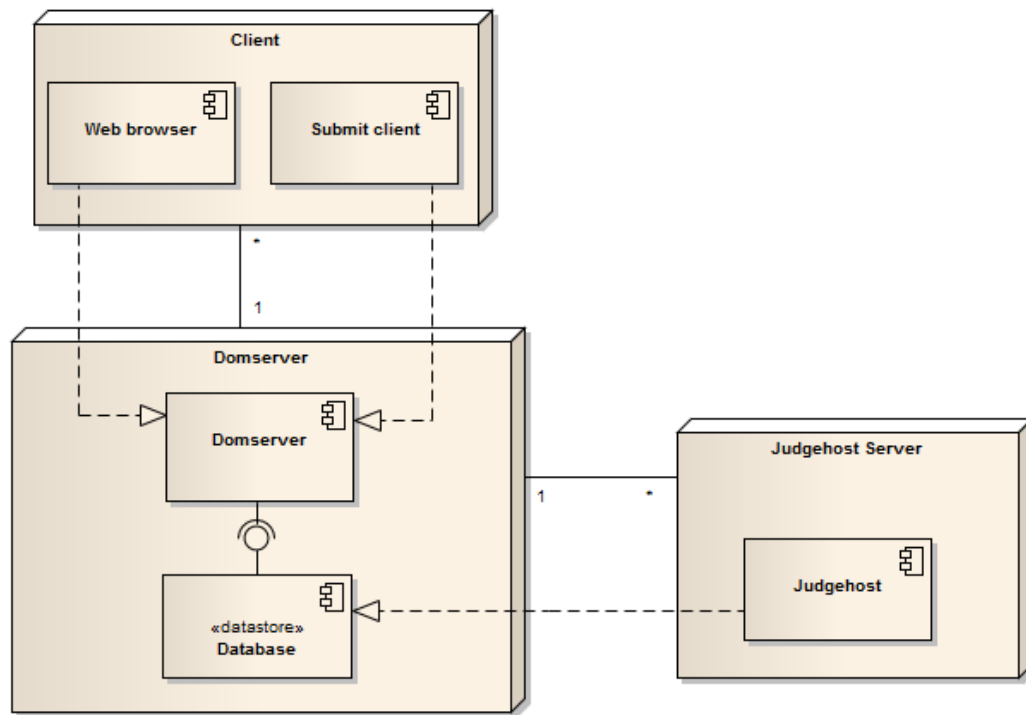


Figure 5.5: DOMjudge's architecture

- **Judgehost Server** - each judgehost server runs a script which retrieves submitted solutions from the database hosted by the Domserver, compiles and tests them and sends the result back to the main server.

### 5.5.2 Overall Architecture

The overall architecture of the CBA system, which was devised based on DOMjudge's architecture, is presented in the deployment model of figure 5.6.

Following there is a description of each of the nodes included in the model.

- **Client** - represents the machines used by teaching staff and students to use the CBA system. These machines need to have a web browser to access the system's interface, which is generated by a Moodle server;
- **Moodle Server** - machine that hosts Moodle's server, running on XAMPP [Fri10]. The CBA system demands for the implementation of a new Moodle plugin that is also maintained in this machine. The plugin is responsible for the interaction with the user, regarding the CBA system features, as well as for the communication with the automatic assessment server;
- **Automatic Assessment Server** - this node can be seen as an extension of the Domserver node presented in DOMjudge's architecture: it hosts DOMjudge's main



## Specification of a New CBA System

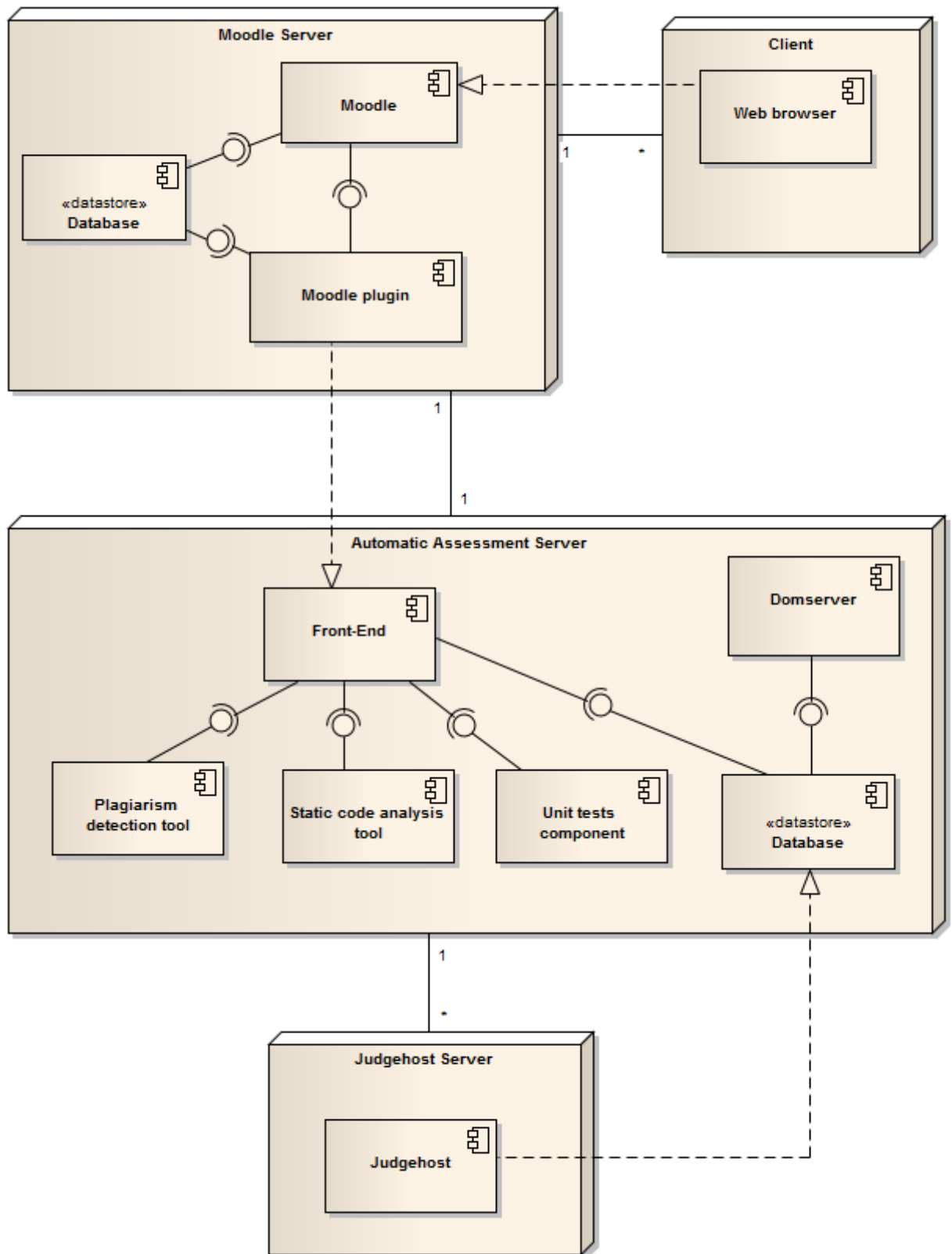


Figure 5.6: Overall architecture

server and the database for keeping the submissions, as well as some additional components. Among the additional components, there is a front end that is needed for linking the Moodle plugin with DOMjudge's main server. The other additional components include an external tool for dealing with plagiarism detection, an external tool for running static code analysis and a module responsible for supporting the assessment of solutions with unit tests;

- **Judgehost Server** - represents the same judgehost server presented in DOMjudge's architecture.

This architecture configuration is important for assuring the security and reliability of the whole system. Information regarding grades and course information is kept in the Moodle server and hence is not affected by potential problems in the automatic assessment server. Furthermore, the main automatic assessment server is kept apart from the servers responsible for running submitted code and therefore will not be affected by potential crashes caused by submissions containing malicious code (the code is run under a sandbox with restricted privileges and hence server crashes are unlikely to happen, but this separation adds extra security).

### 5.5.2.1 Front End Specification

The Front End is the component responsible for defining the communication protocol between the Moodle plugin and the DOMjudge's main server. Moreover, this Front End should add a new abstraction layer and be flexible enough to make it possible to easily link other systems such as Moodle2, SIGEX and the eBook system previously mentioned. The new abstraction layer should be responsible for mapping the entities between DOMjudge's programming context and Moodle's teaching context.

In order to allow the linking of the automatic assessment server to a number of distinct systems, it is desirable to implement the communication protocol using some well-known standard. Among the available alternatives, it was decided to use WSDL [[Conb](#)] in conjunction with SOAP [[Cona](#)] due to its simplicity and previous experiences of CICA with implementing this kind of protocol.

## Chapter 6

# Implementation

Using the specification of the previous chapter as guideline, a prototype of the new CBA system was implemented. In the prototype, the automatic assessment server was integrated with Moodle. Developed in PHP, it implements a subset of the listed features and consists of an activity module plugin able to communicate with the Front End component of the automatic assessment server. The new activity module was called `Programming Assessment`.

This chapter presents the most relevant implementation details. It includes the description of:

1. How DOMjudge entities were mapped to the entities used in the activity module;
2. The communication protocol and the interface provided by the Front End module;
3. The methodology used for extending DOMjudge's features;
4. How the automatic assessment results are gathered from the DOMjudge server to the Moodle server;
5. The mechanism for configuring test cases with attributes such as personalized feedback messages;
6. The skeleton file mechanism, that may be used to simplify the work of students in introductory programming courses;
7. The feedback system, responsible for the generation of feedback reports about the performance of students in the assessments.

## 6.1 Entity Mapping

This section explains how the entities involved in the DOMjudge domain were associated to the ones used by the Moodle plugin. Before explaining how the mapping was done, the entities of both domains are briefly described.

### 6.1.1 DOMjudge Entities

Figure 6.1 presents the main entities of the DOMjudge domain. Their description, as well as the meaning of their relationships, is as follows:

- **Contest** - represents a programming contest. Each DOMjudge server can only run one contest at a time and a contest can have several different programming problems;
- **Problem** - a programming problem can only be used in a single programming contest, can only have one test case and may have multiple submissions from multiple teams;
- **TestCase** - represents a test case used to test the correctness of programming problems. In a relational perspective, a programming problem can only have one test case. However, the input and output files specification may be made in such a way that it may include several logical test cases;
- **Team** - a team may be either a team competing in programming contests, an observer or a member of the organization. It may submit several solutions for the problems of the programming contests;
- **Submission** - represents a solution made by a team to solve one programming problem. A submission has one source file of a given programming language and gets a judging result after being evaluated by a judgehost;
- **Judging** - represents a judging result for a given submission. To be considered correct, the submission output has to completely match the solution output file;
- **Language** - represents a programming language.

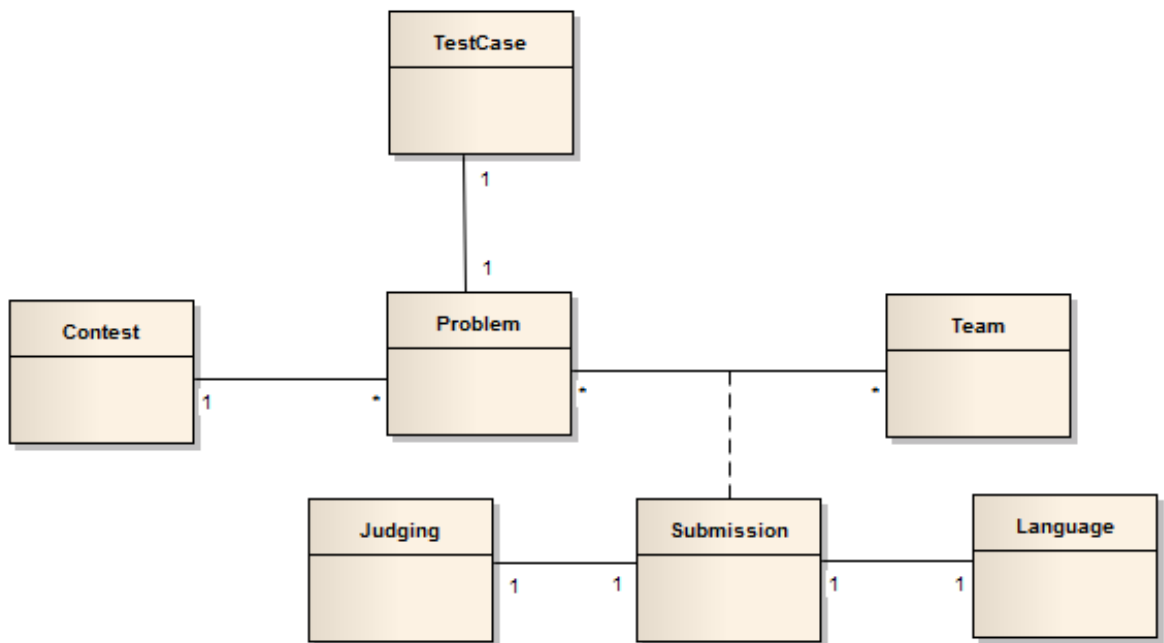


Figure 6.1: DOMjudge main entities

### 6.1.2 Moodle Plugin Entities

Figure 6.2 presents the main entities of the Moodle plugin domain. Their description, as well as the meaning of their relationships, is as follows:

- **ProgrammingAssessment** - a programming assessment is an assignment that may have several test cases and several submissions from multiple users;
- **TestCase** - represents a test case used to test the correctness of programming assessments. Each programming assessment may have several test cases, possibly with different weights;
- **User** - a user may be either a student, a teacher or an administrator. They may submit solutions to solve the available programming assessments;
- **Submission** - represents a solution made by a user to solve one programming assessment. A submission has one source file of a given programming language and gets a result after being evaluated by the automatic assessment server;
- **Result** - represents the automatic assessment result of a given submission. It is composed by the results in all the test cases of the associated programming assessment;
- **Language** - represents a programming language.

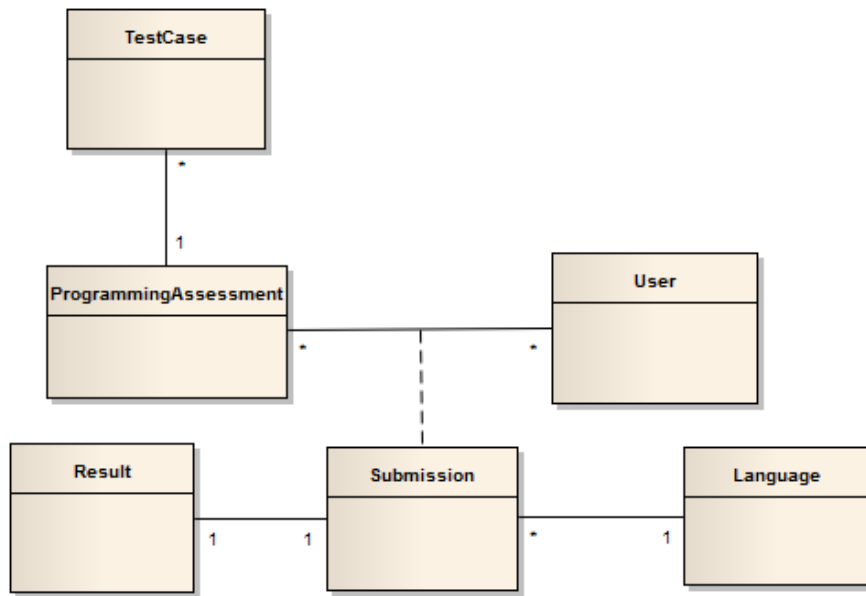


Figure 6.2: Moodle plugin main entities

### 6.1.3 Mapping

As can be concluded from the two previous sections, 6.1.1 and 6.1.2, there is not a direct match between all the entities of the two domains. Still, some of the associations were defined quite straightforwardly:

- Moodle’s `Language` is mapped to DOMjudge’s `Language` ;
- Moodle’s `User` is mapped to DOMjudge’s `Team`;
- Moodle’s `TestCase` is mapped to DOMjudge’s `TestCase`.

The main mapping issues arose when mapping the `ProgrammingAssessment` entity. Associating it to a `Contest` would mean that the main server would only be capable of hosting one assessment at a time. Associating it to a `Problem` would mean that an assessment could only have one `TestCase`. The adopted solution consists in mapping one `ProgrammingAssessment` to multiple `Problems`, one for each `TestCase`. With this solution, the main server can host multiple assessments at the same time.

The association between `ProgrammingAssessments` and multiple `Problems` influenced the mapping of `Submissions`: a Moodle `Submission` is associated to multiple DOMjudge `Submissions`, one per `TestCase` of the `ProgrammingAssessment`. Therefore, the `Result` of a `Submission` consists of all the associated `Judgings`.

Having `Problems` mapped to `ProgrammingAssessments`, it was decided to keep one single `Contest` in DOMjudge’s database. This `Contest` is used by all the `Problems`.

## 6.2 Front End and Communication Protocol

As specified in section 5.5.2.1, the Front End component was defined using WSDL in conjunction with SOAP. Its interface was described in a `.wsdl` file and the implementation was made with `PHP`. The Front End adds a new abstraction layer on top of DOMjudge, by hiding its domain. This is achieved by mapping the DOMjudge entities to the ones associated to the programming assessments, following the rules described in 6.1.3. With this new abstraction layer, it becomes easier to link other systems to DOMjudge's functionalities.

The description of the Front End interface can be found in appendix G and contains methods to:

- Get information about the programming languages supported by the automatic assessment server;
- Add, remove and update programming assessments;
- Add, remove and update test cases;
- Add users to the system;
- Add submissions and get their judging results;

## 6.3 Extending DOMjudge's Features

As referred in section 5.4, DOMjudge did not provide all the functionalities specified for the final system. When developing the prototype, it was decided to keep DOMjudge's source code unchanged and to implement all the non supported features only at the Moodle plugin level. For instance, the implementation of the parameters for assessment configuration listed in section 5.1.2 is completely independent of DOMjudge. This decision was mainly affected by the mapping of programming assessments to problems, which do not have temporal configurations (start time, deadline and tolerance time). Other relevant features that were implemented on top of DOMjudge are the test case configuration system with personalized feedback, described in section 6.5, as well as the skeleton file mechanism, described in section 6.6.

## 6.4 Polling of Automatic Assessment Results

After being submitted via the Moodle interface, the solutions for programming assessments are sent to the DOMjudge server. Since the automatic assessment process may last from a few seconds to a few minutes, depending on the number and complexity of the

test cases, the communication between Moodle and DOMjudge was implemented in an asynchronous fashion. This means that the Moodle plugin does not block, waiting for the judging results, after a submission is made. Instead, the results are polled by using the Cron mechanism, which is a Unix program that runs predefined tasks on a computer at regular intervals. It assists some of Moodle's modules to perform tasks on a scheduled basis [Moo10a].

In terms of implementation, a cron function, which is executed on a regular basis, was created for the Programming Assessment activity module. The interval between executions can be regulated, but its minimum value is restricted to one minute. The algorithm implemented for the cron function is as follows:

1. Fetch, from Moodle's database, all the submissions that still do not have a judging result;
2. For all the fetched submissions, query the DOMjudge server for the judging result;
3. For all the received answers, update Moodle's database with the results;
4. Update student's grades based on the new results.

### 6.5 Test Cases Configuration and Personalized Feedback

In the Moodle plugin, the mechanism for test case definition is based on the mechanism used in DOMjudge, which relies on the usage of plain input and output files. However, some modifications were made in order to adapt it to teaching staff needs. The implemented mechanism is able to detect some specific tags that can be used to define test case attributes.

With the tag system, the input of a test case has to start, mandatorily, with the line `#testcase`. Then, four optional attributes can be defined with lines starting with the symbol `#`:

- **weight** - defines the contribution of the test case to the grade of the assessment. If omitted, the weight is set to 1;
- **name** - sets the name of the test case (this is useful for identifying the test cases in the automatic reports containing the results and feedback of the assessment). If omitted, a number based on the index of the test case is used as name;
- **wrong** - defines a personalized feedback message that is shown to students whenever they fail the test case;
- **right** - defines a personalized feedback message that is shown to students whenever they get the test case right;



## Implementation

The output files do not have tagged attributes, however the start of a test case output needs to be signaled with the line `#testcase`.

Since the automatic assessment server uses separate files for the different test cases, the Moodle plugin strips out the lines with tags. Therefore, the code for handling input and output will not have to deal with the lines with reserved tags.

Following, there is an example of an input and output file for a hypothetical assessment with two test cases:

Listing 6.1: Sample input file

```
#testcase
#weight 50
#name gcd_test
#wrong http://en.wikipedia.org/wiki/Greatest_common_divisor
#right congratulations
gcd
12 18

#testcase
#weight 40
#name prime_test
#wrong http://en.wikipedia.org/wiki/Prime_number
prime
2
3
4
```

Listing 6.2: Sample output file

```
#testcase
6

#testcase
1
1
0
```

## 6.6 Skeleton File Mechanism

As explained in section 5.4, the skeleton file mechanism was specified to simplify the work of students in introductory programming courses. Teaching staff can provide skele-

ton code, which is not visible to students and is responsible for handling input and output data. This allows students to focus on writing the code for problem solving.

Following there is an example containing the outline of a possible skeleton file, written in pseudo code for a hypothetical assessment with two questions:

Listing 6.3: Sample skeleton file

```
function question1_solution(parameters) {
    //code for solving question 1
    return answer;
}

function question2_solution(parameters) {
    //code for solving question 2
    return answer;
}

// studentcode

function read_question1_input() {
    //code for reading input of question 1
    return input;
}

function read_question2_input() {
    //code for reading input of question 2
    return output;
}

function test_question1(parameters) {
    write_line(question1_solution(parameters)
               == question1_student(parameters));

    // alternatively , we could just output the result
    // of question1_student(parameters)
}

function test_question2(parameters) {
    write_line(question2_solution(parameters)
               == question2_student(parameters));
```

```

        // alternatively , we could just output the result
        // of question2_student(parameters)
    }

function main() {
    question = read_line();

    if (question == "question1") {
        parameters = read_question1_input();
        test_question1(parameters);
    } else if (question == "question2") {
        parameters = read_question2_input();
        test_question2(parameters);
    }
}

```

The `main` function starts by reading a line from the input file which specifies the question to test. Then, the parameters for the question in cause are read and the method responsible for testing the question is invoked. The testing method can follow (at least) two approaches:

- Output the result of a direct comparison between staff's solutions and student's solution;
- Output the result of student's solution;

The output files will have different configurations, depending on the approach that is followed. In the case of the first approach, the output files should contain the boolean `true` for each answer, while in the second one the output files should contain the results produced by staff's solutions.

To finish the description of the skeleton file mechanism, it is necessary to explain the meaning of the line `//studentcode`. Whenever a student makes a submission, the submitted source file is merged with the skeleton code. The `//studentcode` is replaced by the submission code and the resulting file is compiled and tested. Therefore, the submitted code is expected to contain the methods `question1_student` and `question2_student` invoked in the skeleton source.

## 6.7 Feedback System

Besides the implementation of personalized messages for test cases, already described in section 6.5, the implemented feedback system also includes the possibility of toggling between immediate and non-immediate feedback and the regulation of the level of feedback given to students.

When the feedback of an assessment is set to be immediate, submissions are sent to the automatic assessment server right after being uploaded to Moodle. The feedback report is automatically generated right after the judging result is polled from DOMjudge's server and stored in Moodle's database. When the feedback is set to be non-immediate, submissions are also sent to the automatic assessment server and the result is polled and stored in Moodle's database. However, the generation of the feedback reports is not automatic and needs to be triggered either by a teacher or a course manager.

The feedback system can be configured to have one of three different levels of detail: minimalist, moderated or detailed. The different feedback levels affect the amount of information that is generated and presented to the user in the feedback report. All the feedback information is obtained from the judging results of the automatic assessment server. The information given for each level is as follows:

- **Minimalist** - global assessment grade and result in each test case;
- **Moderated** - minimalist information plus the input used in each test case;
- **Detailed** - moderated information plus the expected and obtained output in each test case.

Furthermore, the system presents special feedback information to the user whenever a submission receives a compile error or runtime error message.

# Chapter 7

## Results

This chapter presents an evaluation of the results achieved by the system specification and by the implemented prototype, which was installed and configured in a test server running on three CICA's (FEUP's computer centre) virtual machines. Being a test server, a single machine is used for running both the Moodle server and the main DOMjudge server. The two other machines run two judgehosts. The prototype results are evaluated by outlining which features were implemented. Then, the validation of the system is discussed. The chapter ends with some general considerations about the system's features, including an analysis of how it follows the best practices identified in section 3.1.9.

An installation guide is provided in the appendix B, while the instructions for accessing the test server are available in appendix D. By default, the DOMjudge server supports the following programming languages: Bash, C, C++, Haskell, Java, Pascal and Perl. Scheme, which is used in a MIEIC introductory programming course, was also configured in the test server. In fact, the system allows for an easy incorporation of new programming languages, mainly due to DOMjudge's automatic assessment mechanism. The steps needed for configuring a new programming language can be consulted in appendix C.

### 7.1 Implemented Features

This section details which features are available in the system prototype. The features were grouped using the same configuration of section 5.1.2.

#### 7.1.1 Support for Assessment Creation

As shown in table 7.1, the system supports the creation of the three types of assessment identified. The meaning of each type can be recalled from 2.9.1.1:

- **Exams** - traditional (programming) exams that have to be solved by all the students at the same time, within a given time limit;

- **Tests** - programming tests where students are divided in groups. All the students in each group have to solve the test at the same time, within a given time limit;
- **Exercises** - self-learning programming exercises that can be solved by all the students, without time limit;

Different time parameters (start time, deadline and tolerance time) can be used to configure assessments with the presented characteristics. A guide explaining the creation of new assessments can be found in appendix D.

Id	Assessment	Priority	Implemented?
F1.1	Exams	High	Yes
F1.2	Tests	High	Yes
F1.3	Exercises	Medium	Yes

Table 7.1: Implementation of the types of supported assessments

Table 7.2 shows the parameters for assessment configuration that are supported by the prototype. It is important to note that it is possible to associate a specific penalization for late submissions, in percentage, to the tolerance time.

Id	Parameter	Priority	Implemented?
F2.1	Start time	High	Yes
F2.2	Deadline	High	Yes
F2.3	Accepted programming languages	High	Yes
F2.4	Tolerance time	High	Yes
F2.5	Duration	High	Yes
F2.6	Max number of submissions	Medium	Yes
F2.7	Duration of each individual question	Low	No

Table 7.2: Implementation of the parameters for assessment configuration

### 7.1.2 Support for Test-Cases Definition

As seen in table 7.3, the prototype supports one of the identified test-cases definition methodologies: the use of text files with input and output test data, which is the methodology behind DOMjudge's behavior.

Id	Methodology	Priority	Implemented?
F3.1	Text files	High	Yes
F3.2	Unit tests	High	No
F3.3	Regular expressions	Low	No

Table 7.3: Implementation of the methodologies for test-cases definition

From the parameters for test-case configuration presented in table 7.4, only the one with high priority, the weight for assessment grade, was implemented. It is also possible to associate personalized feedback messages to test cases, but this feature was included in the feedback system discussed in section 7.1.4.

Id	Parameter	Priority	Implemented?
F4.1	Weight for assessment grade	High	Yes
F4.2	Static analysis of code quality	Medium	No
F4.3	Max runtime	Medium	No
F4.4	Max memory	Medium	No
F4.5	Distinguish students grades based on solving time	Low	No
F4.6	Identification and penalization of common mistakes	Low	No

Table 7.4: Implementation of the parameters for test-case configuration

### 7.1.3 Support for Elaboration and Submission of Solutions

The two features, associated to the support and elaboration of submissions, were successfully implemented (table 7.5). The solution upload mechanism, which is essential for the system usage, was one of the first implemented features. A guide on how to submit solutions can be consulted in the appendix F

Id	Feature	Priority	Implemented?
F5.1	Mechanism for uploading solutions	High	Yes
F5.2	Skeleton mechanism	High	Yes

Table 7.5: Implementation of the features for supporting the elaboration and submission of solutions

### 7.1.4 Feedback System

As shown in table 7.6, the three features related to the feedback system were implemented. Therefore, it is possible to toggle between immediate and non-immediate feedback, to choose one of three feedback levels and to associate personalized feedback messages to test cases.

Id	Feature	Priority	Implemented?
F6.1	Toggle immediate/non-immediate feedback	High	Yes
F6.2	Regulation of the level of feedback (Minimalist, Moderated, Detailed)	High	Yes
F6.3	Personalized feedback messages	Medium	Yes

Table 7.6: Implementation of the features related to the feedback system

### 7.1.5 Submissions Management

As concluded by observing table 7.7, not much effort was put into the implementation of the features related to submissions management. The prototype only supports the consulting of the history of submissions and grades of students.

<b>Id</b>	<b>Feature</b>	<b>Priority</b>	<b>Implemented?</b>
F7.1	Consult grades statistics	High	No
F7.2	Manual alteration of assessment results	High	No
F7.3	Student history	High	Yes
F7.4	Student notification	High	No
F7.5	Sending of e-mails	Medium	No
F7.6	Alteration of submitted code	Medium	No
F7.7	Exporting of assessment results	Low	No
F7.8	Built-in environment for manual testing of submissions	Low	No

Table 7.7: Implementation of the features related to the management of submissions

### 7.1.6 Extra Features

As shown in table 7.8, none of the features classified as extra is currently supported by the prototype.

<b>Id</b>	<b>Feature</b>	<b>Priority</b>	<b>Implemented?</b>
F8.1	Plagiarism detection	High	No
F8.2	Q&A system	Medium	No
F8.3	Exercise Database	Medium	No
F8.4	Submissions ranking	Low	No

Table 7.8: Implementation of the extra features

## 7.2 Prototype Validation

It is important to refer that the implemented prototype should be validated by DEI teaching staff and students. This can be done, for instance, with pilot tests involving small groups of students of programming courses. Then, surveys can be used to collect opinions and feedback from staff and students. This validation process has still not been made due to the limited amount of time available for the project. Moreover, the end of the implementation phase coincided with the end of the semester, which is a busy academic period with assignment deadlines and exams preparation.



### 7.3 Summary

The prototype implements 14 out of 19 of the features with high priority, 3 out of 10 of the features with medium priority and 0 out 7 of the features with low priority. In the process of linking the Moodle plugin to DOMjudge, some features already supported by the latter were lost, i. e., the appropriate user interface is not implemented by the plugin. Those features include two medium priority features, F4.3 - Max runtime and F8.2 - Q&A system, as well as one low priority feature, F8.4 - Submission ranking.

It is important to underline that the prototype does not simply link DOMjudge's features to Moodle, but also adds new functionalities useful in the academic context. Therefore it is possible, by using Moodle's interface, to create ACM-ICPC like exercises as well as academic programming assessments suitable for introductory programming courses. Examples of the two types of exercises can be found on the test server. The academic assessment example was based on the data of a Scheme test given to students of an introductory programming course of MIEIC. The steps followed for adapting the test to the new system can be consulted in appendix E.

In terms of innovation, the main improvements of the new system in relation to the already existing CBA systems are the possibility of defining test cases with attributes, such as personalized feedback messages, and the skeleton file mechanism. This mechanism is quite useful for introductory programming courses, where students do not master input and output handling. This mechanism was used in the Scheme assessment example of appendix E.

Finally, and making a bridge between the new system's features and the best-practices of the application of CBA systems in e-Learning presented in section 3.1.9, it is possible to make the following conclusions in relation to the new CBA system:

- **BP1** - It uses dynamic tests in the automatic assessment process. Not considered to be essential, automatic static tests were not included in the prototype. However, a component responsible for performing automated static analysis of source code was considered in the specification of the complete architecture of the system;
- **BP2** - Since none of the test case definition techniques was considered to have notorious benefits over the others, the three of them were included in the system's specification. However, the prototype only supports the use of text files, which is the methodology used by DOMjudge;
- **BP3 and BP4** - The skeleton file mechanism is a good system for avoiding problems with control characters, as well as for not forcing students of introductory programming courses to have to handle the input and output of test cases;

## Results

- **BP5** - It is able of giving relevant feedback to students, based on DOMjudge's automatic assessment results and on personalized feedback messages. The feedback detail and the number of allowed submissions can be combined to match one of the configurations that has shown to provide good results;
- **BP6** - Being a major concern in the University context, a component for detecting plagiarism among students was considered while specifying the overall architecture of the system. However, the prototype does not support plagiarism detection;
- **BP7** - System's security is assured at the authenticity and integrity level by Moodle, while DOMjudge uses a sandbox for protecting the system from malicious code;
- **BP8** - The prototype does not support any kind of competition-like assessments. However, a submission ranking feature was considered in the specification and future versions of the system may take advantage of DOMjudge's competition-related features to implement it.

## Chapter 8

# Conclusions and Future Work

### 8.1 Work Summary

The goals of this dissertation were to specify and develop a prototype of a CBA system for supporting the programming components of the courses of DEI. The system should be able to help reducing the amount of work needed for marking and grading programming assignments, by automating the assessment process. It has a core component, behaving as an independent service, responsible for the automatic assessment of programming assignments and flexible enough to be integrated with other platforms such as Moodle, SIGEX and an eBook system. In order to achieve the proposed goals, the following steps were followed:

- A study of eight state-of-the-art CBA systems was performed, with the goal of understanding what features are available. The features were grouped in nine different topics for a better comprehension. The studied systems were firstly presented one by one and then their features were summarized and directly compared;
- An analysis of five cases studies was then performed to understand how CBA systems are currently being used to support the e-Learning process in academic institutions. From this analysis emerged nine best-practices;
- The learning needs of DEI's teaching staff were analyzed by the means of an online survey. The survey structure was based on the topics used to group the features of the state-of-the-art CBA systems;
- Then, a specification for the new CBA system was developed. It has a proposal containing use case diagrams and a list of the features to implement, with different priorities and grouped in topics, using the same division of the state-of-the-art analysis. The specification also contains a study which led to the decision of using DOMjudge as the basis for the final system. The type of Moodle plugin to implement was also determined: with high priority, an activity module and, with less

priority, a new question type. Since DOMjudge could not fulfill all the requirements of the new system, a brief discussion was conducted to explain how and which features would be built on top of it. Finally, the architecture of the CBA system was defined. The overall architecture was obtained by combining DOMjudge's architecture with Moodle and some external tools;

- Having the complete system specification, a prototype of the CBA system was developed. It implements a subset of the specified functionalities and consists of a Moodle activity module plugin integrated with DOMjudge's automatic assessment server. Moodle and DOMjudge communicate using WSDL in conjunction with SOAP. The prototype was then set up on a test server running on CICA's virtual machines. The server supports programming assessments in DOMjudge's default languages (Bash, C, C++, Haskell, Java, Pascal and Perl), as well in Scheme, a functional programming language used in a MIEIC introductory programming course;
- Finally, the results achieved by the project were assessed. A significant amount of the specified features was implemented: 14 out of 19 with high priority and 3 out of 10 with medium priority. It was concluded that the prototype does not simply link DOMjudge's features to Moodle, but also adds new functionalities, useful in the academic context. Moreover, the system has two innovative mechanisms not found in other CBA systems: the possibility of defining test cases with attributes, such as personalized feedback messages, and a skeleton file mechanism. It was also concluded that the system allows to follow the previously identified best-practices of the application of CBA systems in e-Learning.

Some documents, regarding the configuration and usage of the system were created and included as appendix, at the end of this dissertation report. These documents include:

- **Appendix B** - a guide with some tips for installing the implemented prototype;
- **Appendix C** - a step-by-step guide for configuring a new programming language in the CBA system;
- **Appendix D** - a guide on how to create and configure a new programming assessment;
- **Appendix E** - a guide with the steps followed for adapting to the new CBA system a Scheme test, which was given to students of an introductory programming course of MIEIC and automatically corrected by the system referred in [1.1](#);
- **Appendix F** - a guide explaining the user interface for submitting solutions to programming assessments.

## 8.2 Future Work

Despite having a complete system specification and a functional prototype that implements a significant amount of features and that is available on a test server, there are still further steps that have to be followed in order to obtain a mature system to be used in real tests and exams:

- The five missing high priority features are of great importance for the system and hence should be implemented. Four of those features are related to the management of submissions and will contribute for increasing the control that teaching staff have over the submitted solutions and over the automatic assessment results. The fifth missing feature is related to the detection of plagiarism. This can be, in principle, implemented by integrating one off-the-shelf external tool with the system's prototype;
- It would be interesting to take advantage of some additional DOMjudge features. For instance, the use of *validators* can be extremely useful for problems that do not fit within the standard scheme of fixed input and/or output. Moreover, the programming contests related features, namely the team ranking, can be used to set up some small competitions to foster students' motivation;
- Some usability aspects can be improved. For instance, the interface for submitting solutions and consulting their results can be improved by using more JavaScript and AJAX;
- Finally, the system needs to be stress-tested and validated by teaching staff and students. This can be done, for instance, with pilot tests involving groups of students of programming courses. Then, surveys can be used to collect opinions and feedback from staff and students.

## Conclusions and Future Work

# References

- [AI10] ACM-ICPC. The ACM-ICPC International Collegiate Programming Contest Web Site, 2010. <http://cm.baylor.edu/welcome.icpc>. Last accessed: June 22, 2010.
- [BBFH95] Steve Benford, Edmund Burke, Eric Foxley, Colin Higgins. The Ceilidh System for the Automatic Grading of Students on Programming Courses. *Proceedings of the 33rd ACM Southeast Conference*, Clemson University, South Carolina, pages 176-182, 17th-19th March 1995.
- [BGNM04a] Michael Blumenstein, Steve Green, Ann Nguyen, and Vallipuram Muthukkumarasamy. GAME: A Generic Automated Marking Environment for Programming Assessment. itcc, vol. 1, pp.212, *International Conference on Information Technology: Coding and Computing (ITCC'04)* Volume 1, 2004.
- [BGNM04b] Michael Blumenstein, Steven Green, Ann Nguyen, and Vallipuram Muthukkumarasamy. An experimental analysis of GAME: a generic automated marking environment. *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, Leeds, United Kingdom, June 28-30, 2004.
- [Blo56] Benjamin S. Bloom. Taxonomy of Educational Objectives: The Classification of Educational Goals, *volume Handbook I: Cognitive domain*. New York, 1956.
- [Cam97] Tracy Camp. The incredible shrinking pipeline. *Commun. ACM*, 40(10):103–110, 1997.
- [CAMF<sup>+</sup>03] Janet Carter, Kirsti Ala-Mutka, Ursula Fuller, Martin Dick, John English, William Fone, and Judy Sheard. How shall we assess this?. *Working group reports from ITiCSE on Innovation and technology in computer science education*, p.107-123, Thessaloniki, Greece, 2003.
- [CL01] Fintan Culwin and Thomas Lancaster. Plagiarism issues for higher education. *Inf. Security*, 21(2):36 – 41, 2001.
- [Cona] World Wide Web Consortium. SOAP Specifications. <http://www.w3.org/TR/soap/>. Last accessed: June 22, 2010.
- [Conb] World Wide Web Consortium. Web Services Description Language (WSDL). <http://www.w3.org/TR/wsdl>. Last accessed: June 22, 2010.

## REFERENCES

- [Cum08] Stephen Cummins. Changing Programming Feedback Using Web 2.0 Technologies. Technical Report TR-TEL-08-05, Durham University, 2008.
- [Dal99] Charlie Daly. RoboProf and an introductory computer programming course. *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, p.155-158, Cracow, Poland, June 27-30, 1999.
- [DH04] Charlie Daly and Jane M. Horgan. An automated learning system for Java programming. *IEEE Transactions on Education*, 47(1):10 – 17, 2004.
- [DLO05] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, v.5 n.3, p.4-es, September 2005.
- [DS97] Norman R. Draper and Harry Smith. Applied Regression Analysis. Wiley, New York, 1997.
- [edu10] EduJudge, 2010. <http://www.edujudge.eu/>. Last accessed: June 22, 2010.
- [EKW09a] Jaap Eldering, Thijs Kinkhorst, and Peter van de Werken. DOMjudge - Programming Contest Jury System, 2009. <http://domjudge.sourceforge.net/>. Last accessed: June 22, 2010.
- [EKW09b] Jaap Eldering, Thijs Kinkhorst, and Peter van de Werken. DOMjudge team manual, 2009.
- [EKW10a] Jaap Eldering, Thijs Kinkhorst, and Peter van de Werken. DOMjudge Administrator’s Manual, 2010.
- [EKW10b] Jaap Eldering, Thijs Kinkhorst, and Peter van de Werken. DOMjudge Jury Manual, 2010.
- [FHST98] Eric Foxley, Colin Higgins, Pavlos Symoniedes, and Athanasios Tsintisifas. Security issues under Ceilidh’s WWW interface. *Proc ICCE’98*, Vol 1 pp235-240, Beijing, China, 14-17 Oct, 1998. Springer, ISBN: 7-04-007336-6.
- [FHST01] Eric Foxley, Colin Higgins, Pavlos Symeonidis, and Athanasios Tsintisifas. The CourseMaster Automated Assessment System – a next generation Ceilidh. *Proceedings of the Workshop on Computer Assisted Assessment to Support the ICS Disciplines*, April 5-6 2001.
- [FHTS00] Eric Foxley, Colin Higgins, Athanasios Tsintisifas, and Pavlos Symoniedes. The Ceilidh-CourseMaster System, An Introduction. *Proceedings of the 4th Java in the Curriculum Conference*, South Bank University, UK, January 24, 2000.
- [Fri10] Apache Friends. XAMPP, 2010. <http://www.apachefriends.org/en/xampp.html>. Last accessed: June 22, 2010.



## REFERENCES

- [GMFA09] Ginés García-Mateos and José Luis Fernández-Alemán. A course on algorithms and data structures using on-line judging. *ACM*, 45-49, 2009.
- [GVN02] Moumita Ghosh, Brijesh Kumar Verma, and Ann Nguyen. An Automatic Assessment Marking and Plagiarism Detection System, *Proceedings of International Conference on Information Technology and Applications*, Charles Sturt University, 2002.
- [HGST05] Colin A. Higgins, Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintsifas. Automated assessment and experiences of teaching programming. *Journal on Educational Resources in Computing (JERIC)*, v.5 n.3, p.5-es, September 2005.
- [HHST03] Colin Higgins, Tarek Hegazy, Pavlos Symeonidis, and Athanasios Tsintsifas. The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies*, v.8 n.3, p.287-304, September 2003.
- [HLVW02] Bodo Husemann, Jens Lechtenbörger, Gottfried Vossen, and Peter Westerkamp. XLX - A Platform for Graduate-Level Exercises. *Proceedings of the International Conference on Computers in Education*, p.1262, December 03-06, 2002.
- [Hol60] Jack Hollingsworth. Automatic graders for programming classes. *Communications of the ACM*, v.3 n.10, p.528-529, Oct. 1960.
- [HST02] Colin Higgins, Pavlos Symeonidis, and Athanasios Tsintsifas. The marking system for CourseMaster. *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, Aarhus, Denmark, June 24-28, 2002.
- [HTS02] Colin A. Higgins, Athanasios Tsintsifas, and Pavlos Symeonidis. CourseMaster marking programs and diagrams, *Proceedings of the Dealing with Plagiarism in ICS Education Conference*, Warwick, April 11-12 2002.
- [JGB05] Mike Joy, Nathan Griffiths, and Russell Boyatt. The boss online submission and assessment system. *Journal on Educational Resources in Computing*, 5(3):2, 2005.
- [JL95] Mike Joy and Michael Luck. On-line submission and testing of programming assignments. *Innovations in Computing Teaching*, SEDA, 1995.
- [JL98] Mike Joy and Michael Luck. Effective electronic marking for on-line assessment. *SIGCSE Bull.*, 30(3):134–138, 1998.
- [JL99] M. Joy and M. Luck. Plagiarism in Programming Assignments. *IEEE Transactions on Education*, 42(1):129–133, 1999.
- [Lau93] Diana Laurillard. Rethinking University Teaching: A Conversational Framework for the Effective Use of Learning Technologies, 1993.
- [Lea09] José Paulo Leal. Mooshak, 26 May 2009 2009. <http://mooshak.dcc.fc.up.pt/>. Last accessed: June 22, 2010.

## REFERENCES

- [LS03] José Paulo Leal and Fernando Silva. Mooshak: a Web-based multi-site programming contest system. *Software—Practice and Experience*, 33(6):567–581, 2003.
- [MKKN05] Lauri Malmi, Ville Karavirta, Ari Korhonen, and Jussi Nikander. Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *Journal on Educational Resources in Computing*, 5(3):7, 2005.
- [Moo10a] MoodleDocs. Cron, 2010. <http://docs.moodle.org/en/Cron>. Last accessed: June 22, 2010.
- [Moo10b] MoodleDocs. Moodle developer documentation, 2010. <http://docs.moodle.org/en/Development>. Last accessed: June 22, 2010.
- [Mue08] Department of Information Systems of the University of Muenster. xLx - eXtreme e-Learning eXperience, 2008. <http://dbis-group.uni-muenster.de/projects/xlx/>. Last accessed: June 22, 2010.
- [Pla94] P. J. Plauger. Fingerprints. *Embedded Systems Programming*, pages 84–87, 1994.
- [PRS<sup>+</sup>03] Yusuf Pisan, Debbie Richards, Anthony Sloane, Helena Koncek, and Simon Mitchell. Submit! a web-based system for automatic program critiquing, 2003. *Proceedings of the fifth Australasian computing education conference on Computing education*, Society, Inc, pages 59-68, Darlinghurst, Australia, 2003.
- [RML08] Miguel Revilla, Shahriar Manzoor, and Rujia Liu. Competitive Learning in Informatics: The UVa Online Judge Experience. *Olympiads in Informatics*, Vol. 2, pages 131–148, 16-23 August 2008.
- [Rob99] Robby Robson. WWW-Based Course-support Systems: The First Generation. *International Journal of Educational Telecommunications*, pages 271–282, 1999.
- [Row87] Derek Rowntree. Assessing Students: How Shall We Know Them? Nichols Pub Co., 1987.
- [SVW06] Joachim Schwieren, Gottfried Vossen, and Peter Westerkamp. Using Software Testing Techniques for Efficient Handling of Programming Exercises in an E-Learning Platform. *Electronic Journal of e-Learning*, 4(1):87–94, 2006.
- [TLN<sup>+</sup>04] Djamshid Tavangarian, Markus E. Leybold, Kristin Nölting, Marc Röser, and Denny Voigt. Is e-Learning the Solution for Individual Learning? *Electronic Journal of e-Learning*, 2(2), 2004.
- [uva10] UVa Online Judge, 2010. <http://uva.onlinejudge.org/>. Last accessed: June 22, 2010.

## REFERENCES

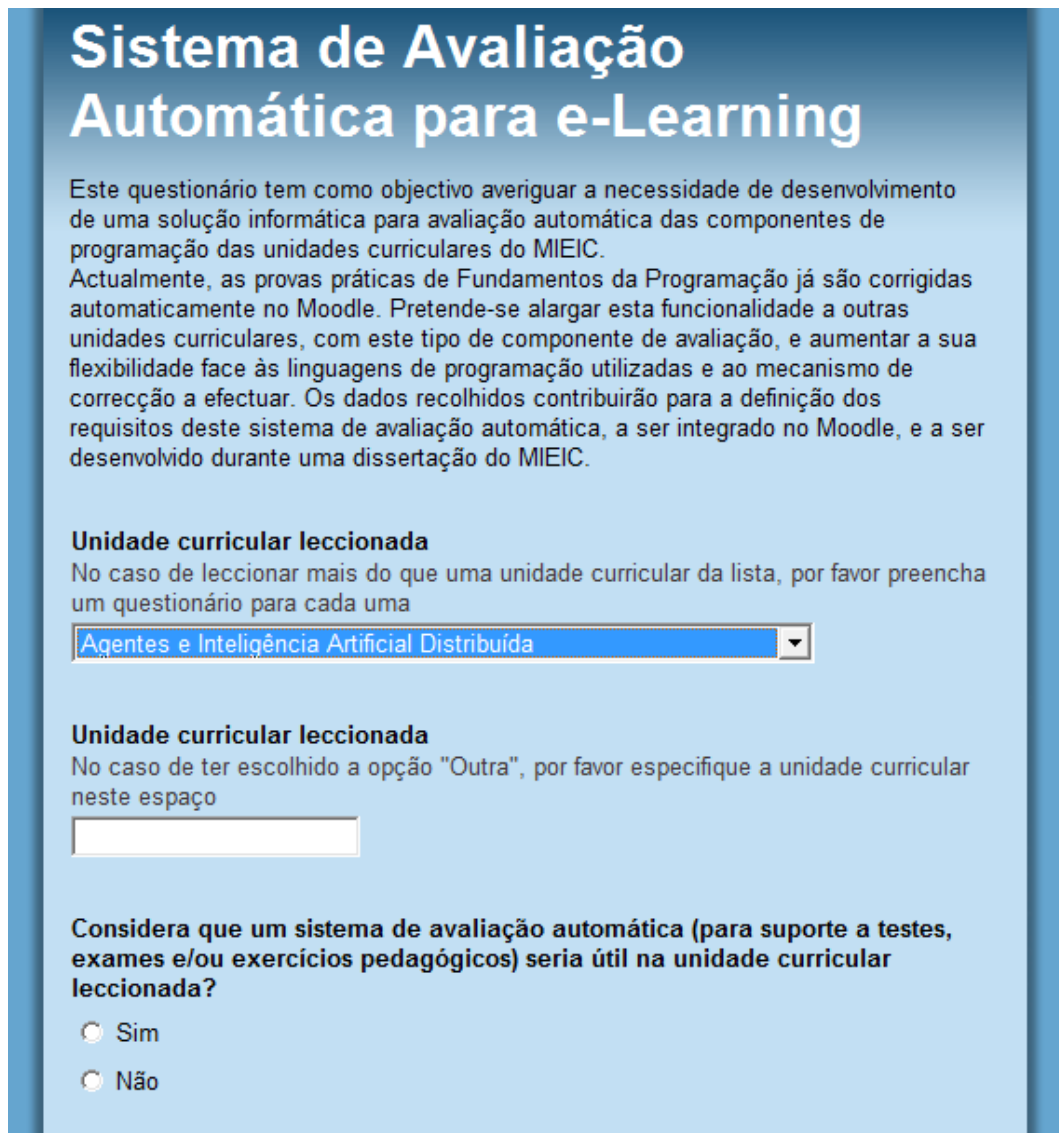
- [VH03] Anne Venables and Liz Haywood. Programming students NEED instant feedback!. *Proceedings of the fifth Australasian conference on Computing education*, p.267-272, Adelaide, Australia, February 1, 2003.
- [War09] The University of Warwick. *BOSS Online Submission System*. 2009. <http://www.dcs.warwick.ac.uk/boss/>. Last accessed: June 22, 2010.
- [YH03] Chye-Foong Yong and Colin Higgins. Automatically creating personalised exercises based on student profiles. *SIGCSE Bull.*, 35(3):236–236, 2003.

## REFERENCES

# Appendix A

## Survey

Survey available online at <http://tinyurl.com/376yb9f>.



**Sistema de Avaliação Automática para e-Learning**

Este questionário tem como objectivo averiguar a necessidade de desenvolvimento de uma solução informática para avaliação automática das componentes de programação das unidades curriculares do MIEIC.

Actualmente, as provas práticas de Fundamentos da Programação já são corrigidas automaticamente no Moodle. Pretende-se alargar esta funcionalidade a outras unidades curriculares, com este tipo de componente de avaliação, e aumentar a sua flexibilidade face às linguagens de programação utilizadas e ao mecanismo de correcção a efectuar. Os dados recolhidos contribuirão para a definição dos requisitos deste sistema de avaliação automática, a ser integrado no Moodle, e a ser desenvolvido durante uma dissertação do MIEIC.

**Unidade curricular leccionada**

No caso de leccionar mais do que uma unidade curricular da lista, por favor preencha um questionário para cada uma

**Unidade curricular leccionada**

No caso de ter escolhido a opção "Outra", por favor especifique a unidade curricular neste espaço

**Considera que um sistema de avaliação automática (para suporte a testes, exames e/ou exercícios pedagógicos) seria útil na unidade curricular leccionada?**

☐ Sim

☐ Não

Figure A.1: Survey page 1 - Initial page

**Criação de Novas Provas (2/8)**

**Em que tipo de provas considera que o sistema a desenvolver será útil?**  
1 = nada útil, 5 = bastante útil

	1	2	3	4	5
Exames (tempo limitado e síncrono com os restantes estudantes)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Auto-testes (tempo limitado e assíncrono)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Exercícios de auto-aprendizagem (sem limite de tempo e assíncrono)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Ao criar uma nova prova, que parâmetros gostaria de ter disponíveis?**  
Ex: Início da prova; duração e tolerância; número máximo de submissões possíveis; linguagens de programação válidas para as submissões, ...

Figure A.2: Survey page 2 - Creation of new assessments

**Definição de Casos de Teste e Avaliação de Submissões (3/8)**

Num sistema de avaliação automática, as submissões para um problema/prova são geralmente avaliadas através de um ou mais casos de teste, que especificam um determinado input e o output respectivo. Diferentes casos de teste podem representar diferentes pesos para a avaliação global. Para além da utilização dos casos de teste, as submissões podem também ser avaliadas utilizando parâmetros como tempo de execução, memória utilizada e qualidade do código.

Figure A.3: Survey page 3 - Test case definition and automatic assessment

**Assinale qual(is) a(s) funcionalidades(s) para definição dos casos de teste que gostaria de ver suportadas**

☐ Criação manual (ou semi-automática) de ficheiros de texto com os inputs e outputs esperados

☐ Definição de expressões regulares, utilizadas para gerar inputs e outputs automaticamente

☐ Criação de testes unitários (permite uma maior flexibilidade na definição dos testes)

☐ Validação dos casos de teste e respectiva cotação

☐ Outra:

**A cada caso de teste podem ser associados diferentes parâmetros. Na sua opinião, que importância atribui à possibilidade de definição dos seguintes parâmetros?**  
 1 = nada importante, 5 = bastante importante

	1	2	3	4	5
Peso na nota final do problema/prova	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Tempo de runtime limite para resolução	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Memória limite	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Análise da qualidade estática do código (indentação, complexidade McCabe, quantidade de comentários, ...)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Outros parâmetros**  
 Utilize este espaço se pretender indicar outros parâmetros adicionais

Figure A.3: Survey page 3 - Test case definition and automatic assessment (cont)

**Elaboração e Submissão de Soluções (4/8)**

**Qual o método que considera mais adequado para os estudantes elaborarem as soluções para os problemas?**  
 Se considerar que ambos os métodos são vantajosos, por favor seleccione a opção "Other" e, por favor, justifique a sua opinião

☒ Directamente no sistema de avaliação, através de editor+compilador integrados  
☐ Localmente, num editor à escolha do estudante  
☐ Outra:

**Qual a utilidade que atribui a um mecanismo para suporte a esqueletos/esboços de soluções?**  
 Este mecanismo permitiria, por exemplo, colocar no enunciado electrónico de um teste o esqueleto para a solução de um problema. Ao clicar no esqueleto da solução, um editor, integrado ou local, contendo o código respectivo seria aberto. (1 - nada útil, 5 - bastante útil)

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure A.4: Survey page 4 - Creation and submission of solutions

**Sistema de Feedback (5/8)**

Num sistema de avaliação automática, após a submissão de uma solução, os estudantes recebem habitualmente um relatório com feedback relativo à (in)correção do código fonte. O nível de detalhe do relatório pode ser bastante variável: desde uma simples mensagem indicando se o programa está ou não correcto até uma lista de mensagens indicando, para cada caso de teste, a sua correcção, memória utilizada, tempo de execução, input, output obtido e esperado.

**Que nível(is) de feedback gostaria de ver disponibilizado(s) pelo sistema?**

☐ Minimalista (para cada submissão indicar apenas a nota obtida)  
☐ Intermédio (para cada submissão indicar a nota obtida e, para cada caso de teste, se foi aceite ou não. Não sendo aceite, podem ser apresentadas razões como Resposta Errada, Tempo Limite Excedido, Erro de Runtime, etc)  
☐ Detalhado (para cada submissão indicar a nota obtida. Para cada caso de teste indicar se foi aceite ou não, e respectivo motivo, input utilizado, output esperado e obtido e resultados dos restantes parâmetros de avaliação)  
☐ Outra:

Figure A.5: Survey page 5 - Feedback system



**Qual a utilidade que atribui a um mecanismo que possibilite associar mensagens de feedback personalizadas a casos de teste/problemas?**

Exemplo: considere-se um problema em que se pede para arredondar o resultado às unidades. Num dos casos de teste a solução esperada é 1.6, ou seja 2. Com este mecanismo, poder-se-á configurar o sistema para emitir mensagens personalizadas no caso da solução obtida ser 1.6 (o estudante esqueceu-se de arredondar o resultado) ou 1 (possivelmente, o resultado foi truncado). 1 = nada útil, 5 = bastante útil

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure A.5: Survey page 5 - Feedback system (cont)

**Gestão de Submissões (6/8)**

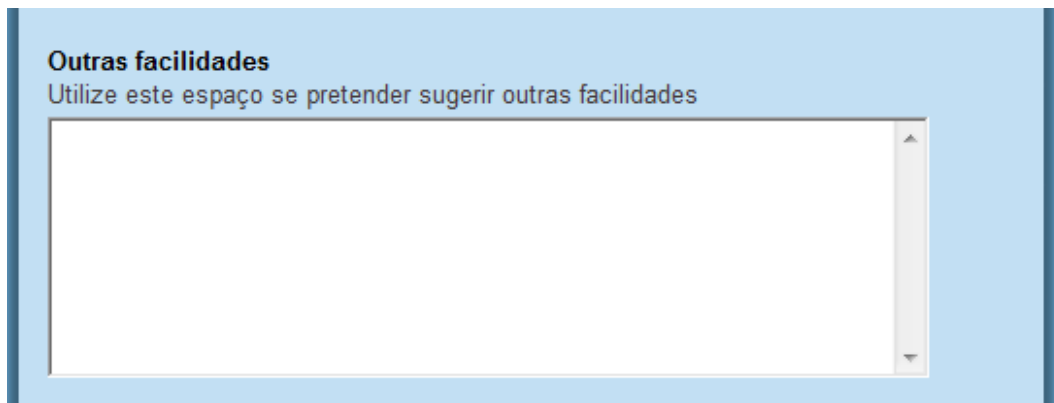
O sistema de avaliação deverá ter uma área onde os docentes poderão gerir as submissões efectuadas pelos estudantes

**Qual a utilidade que atribui à implementação das seguintes facilidades:**  
1 = nada útil, 5 = bastante útil

	1	2	3	4	5
Consulta do histórico de submissões e classificações de cada estudante	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Notificação (automática e/ou manual) de estudantes que não submeteram soluções	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Alteração manual de código submetido	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Alteração manual dos resultados da avaliação automática	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Enviar mensagens de e-mail a conjuntos de estudantes, de acordo com critérios relativos à avaliação	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure A.6: Survey page 6 - Submissions management

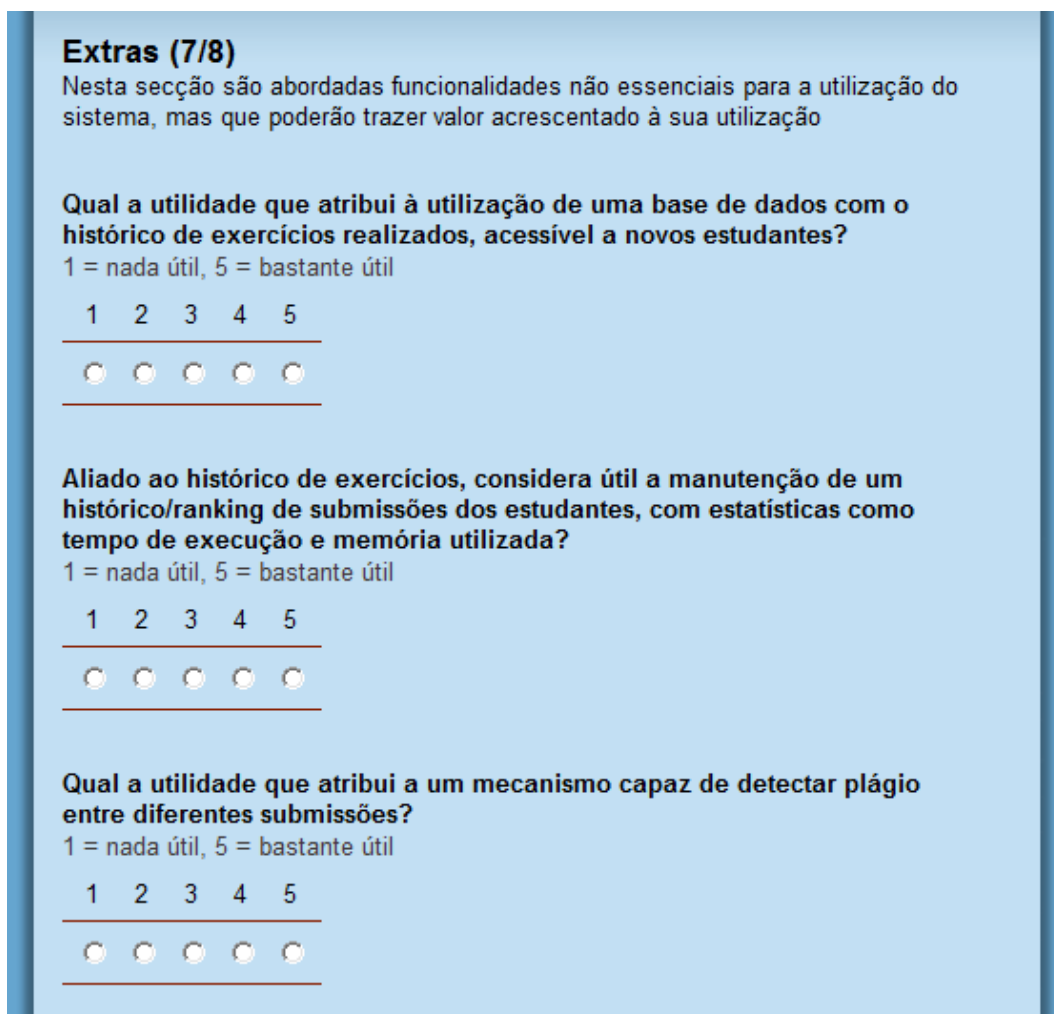
## Survey



**Outras facilidades**

Utilize este espaço se pretender sugerir outras facilidades

Figure A.6: Survey page 6 - Submissions management (cont)



**Extras (7/8)**

Nesta secção são abordadas funcionalidades não essenciais para a utilização do sistema, mas que poderão trazer valor acrescentado à sua utilização

**Qual a utilidade que atribui à utilização de uma base de dados com o histórico de exercícios realizados, acessível a novos estudantes?**  
1 = nada útil, 5 = bastante útil

1 2 3 4 5

☐ ☐ ☐ ☐ ☐

**Aliado ao histórico de exercícios, considera útil a manutenção de um histórico/ranking de submissões dos estudantes, com estatísticas como tempo de execução e memória utilizada?**  
1 = nada útil, 5 = bastante útil

1 2 3 4 5

☐ ☐ ☐ ☐ ☐

**Qual a utilidade que atribui a um mecanismo capaz de detectar plágio entre diferentes submissões?**  
1 = nada útil, 5 = bastante útil

1 2 3 4 5

☐ ☐ ☐ ☐ ☐

Figure A.7: Survey page 7 - Extra features

## Survey

**Qual a utilidade que atribui a um mecanismo de perguntas e respostas?**  
Com este mecanismo, os estudantes podem utilizar o sistema para efectuar perguntas relativas aos diferentes problemas propostos. As respostas serão também dadas pelo docente através do sistema. (1 = nada útil, 5 = bastante útil)

1   2   3   4   5

---

☐ ☐ ☐ ☐ ☐

---

Figure A.7: Survey page 7 - Extra features (cont)

**Obrigado pela sua participação! (8/8)**

**Observações**  
No caso de ter observações/comentários adicionais, por favor utilize este espaço

Figure A.8: Survey page 8 - Final page

## Survey

## Appendix B

# Installation Guide

This appendix contains a guide with some useful tips for installing the implemented prototype. These tips are based on the experience of installing the system on a test server running on Debian in CICA's virtual machines.

The appendix is divided in three sections, one for each physical node (excluding the client) identified in the system's overall architecture. Therefore, there is one section explaining the installation of the main automatic assessment server, another one explaining how to install a judgehost and, finally, a section describing the installation of the Moodle server.

### B.1 Installing the Main Automatic Assessment Server

In order to configure the main automatic assessment server, you need to first install DOMjudge's domserver. To do so, it is recommended to follow the instructions of the administrator's manual [EKW10a]. In the first place, it is necessary to have all the listed software requirements installed. Since the domserver runs on Apache and MySQL, XAMPP [Fri10] was installed. Then, and assuming you are using XAMPP, the `configure` script can be run as:

Listing B.1: Executing the configure script

```
sudo configure --prefix=/opt/lampp/htdocs/domjudge
```

Then, compile and install the domserver:

Listing B.2: Compile and install the domserver

```
sudo make domserver && sudo make install-domserver
```

While configuring the test server, there were some problems with the creation of the file `dbpasswords.secret` located in the folder `etc` of the domserver. It had to be manually created. A possible configuration for this file (with unsafe passwords) may be:

Listing B.3: File `dbpasswords.secret`

```
# Format: '<role>:<db_host>:<db_name>:<user>:<password>'
team:localhost:domjudge:domjudge_team:123
```

```
jury:localhost:domjudge:domjudge_jury:123
public:localhost:domjudge:domjudge_public:123
plugin:localhost:domjudge:domjudge_plugin:123
```

Then, DOMjudge's database has to be installed. This is performed, as indicated in the admin manual, by running the `dj-setup-database` script located in the folder `bin` of `domserver`:

Listing B.4: DOMjudge database installation

```
dj-setup-database [-u <admin_user>] [-p <passw>|-r] install
```

Having the database correctly installed, the default contests should be removed from the table `contest` and a new contest, to be used by the CBA system, needs to be added (see figure B.1).

contestname Descriptive name	activatetime Time contest becomes visible in team/public views	starttime Time contest starts, submissions accepted	freezetime Time scoreboard is frozen	endtime Time after which no more submissions are accepted	unfreezetime Unfreeze a frozen scoreboard at this time
progassessment	2000-01-01 00:00:00	2000-01-01 00:00:00	NULL	2100-01-01 00:00:00	NULL

Figure B.1: Setting up the contest table

The database installation finishes the `domserver` installation process. Then, the Front End component of the automatic assessment server needs to be configured. To do so, two source files are needed: `frontend.php` and `frontend.wsdl`, which can be obtained from <http://paginas.fe.up.pt/~ei05058/frontend.zip>. After downloading the zip file, copy the `frontend` folder, containing the two source files, to the following location: `/opt/lamp/htdocs/domjudge/`.

## B.2 Installing a Judgehost

It is also recommended to follow the instructions of DOMjudge's administrator manual while configuring a judgehost. In the first place, it is necessary to have all the listed software requirements installed. Then, the configure script needs to be executed. Since the judgehosts run submitted code with non-root privileges, the judgehost should be installed to a folder not needing root privileges.

Listing B.5: Executing the configure script

```
sudo configure --prefix=$HOME/domjudge
```

Then, compile and install the judgehost:

Listing B.6: Compile and install the judgehost

```
sudo make judgehost && sudo make install-judgehost
```

Then, a user with minimal privileges needs to be added. This user is used for executing submitted code. On Debian this can be made with:

Listing B.7: Adding user with minimal privileges

```
useradd -d /nonexistent -g nogroup -s /bin/false domjudge-run
```

Having the judgehost installed, a new entry has to be added to the table `judgehost` of the database of the main automatic assessment server. Figure B.2 presents the example of an entry for a judgehost running on a machine named Ubuntu.

hostname	Resolvable hostname of judgehost	active	Should this host take on judgments?
ubuntu			1

Figure B.2: Adding a new judgehost to DOMjudge

To execute the judgehost script, run the `judgedaemon` file located in the `bin` folder. The script can be scheduled to run every time the machine gets started, by using the Linux crontab mechanism with the `@reboot` keyword.

### B.3 Installing the Moodle Server

Instructions for installing Moodle and Moodle plugins can be found in its official site, <http://moodle.org/>.

In the test server, XAMPP was used. Moodle source code was downloaded from the official site and copied to the `htdocs` folder of the XAMPP installation. The activity module plugin was developed for version 2.0 and it is probable that some features do not work on previous version (however, this was not confirmed). The source code of the Programming Assessment activity module can be downloaded from <http://paginas.fe.up.pt/~ei05058/progassessment.zip> and should be copied to the `mod` folder of Moodle.





## Appendix C

# Programming Language Configuration Guide

DOMjudge supports, by default, the following programming languages: Bash, C, C++, Haskell, Java, Pascal and Perl. As explained in the Administrator's Manual [EKW10a], in order to configure a new language, one needs to install a compiler, create a shell-script named `compile_<lang>.sh` and place it in the `lib/judge` folder of the judgehosts.

This appendix contains a step-by-step guide for configuring a new programming language, PLT Scheme, which is the language used for introducing MIEIC students to programming. This involves not only configuring the language in DOMjudge but also configuring it in the Moodle plugin.

### C.1 DOMjudge Configuration

#### C.1.1 Step 1

The first step is to install an appropriate compiler. This can be done by downloading and running one of the installers available on <http://download.plt-scheme.org/> or, in Debian/Ubuntu by running the command `sudo apt-get install plt-scheme`. This will install MzScheme, which can be used to generate executables from scheme source code.

#### C.1.2 Step 2

The second step is to create a new entry in the table `language` of DOMjudge's database (figure C.1).

#### C.1.3 Step 3

The third step consist in editing the configuration file `domserver-config.php`, located in the folder `etc` of the domserver. The constant `LANG_EXTS` has to be updated with the new language data:

Listing C.1: Editing the constant `LANG_EXTS`

```
define('LANG_EXTS', 'C,c C++,cpp,cc,c++ Java,java Pascal,
                    pas,p, Haskell,hs,lhs Perl,pl Bash,sh Scheme,scm');
```

langid Unique ID (string)	name Descriptive language name	extension Filename extension for this language	allow_submit Are submissions accepted in this language?	allow_judge Are submissions in this language judged?	time_factor Language-specific factor multiplied by problem run times
bash	Bash	sh	0	1	1
c	C	c	1	1	1
cpp	C++	cpp	1	1	1
haskell	Haskell	hs	0	1	2
java	Java	java	1	1	1.5
pascal	Pascal	pas	0	1	1
perl	Perl	pl	0	1	1
scheme	Scheme	scm	1	1	5

Figure C.1: Programming Language Configuration - Step 2

## C.1.4 Step 4

The final step consists in the creation of the compilation script. The file `compile_scheme.sh` can be defined as follows:

Listing C.2: Scheme compile script

```
#!/bin/sh

# Scheme compile wrapper-script for 'test_solution.sh'.
# See that script for syntax and more info.

SOURCE="$1"
DEST="$2"

mzc --exe $DEST $SOURCE

exit 0
```

## C.2 Moodle Configuration

### C.2.1 Step 1

To configure the programming language in the Moodle plugin, a single step is enough. It consists in editing the file `languages_config.php` by adding the character used to create comments in the desired language. This information is used to process skeleton code files:

Listing C.3: Editing the file `languages_config.php`

```
<?php

$progassessment_languages_comments = array(
    "cpp" => "//",
    "c" => "//",
```

```
"java" => "//" ,  
"scheme" => ";"  
);  
define('STUDENT_CODE_IDENTIFIER', "studentcode");  
?>
```



## Appendix D

# Assessment Creation Guide

This appendix aims to be a guide on how to create and configure a new programming assessment. It is assumed that the reader has some previous knowledge about using Moodle and creating activity modules.

The creation/edition form is divided in 8 sections, 7 of which are specific of the programming assessment activity module. The last section is related to common module settings and is not explained in this guide.

The following is a description of how to access a server that was created for testing purposes. Then, two different guides are presented: one with a programming assessment with skeleton code and the other without it. For each one there is an explanation on how to fill each of the programming assessment configuration sections. Some sample data for a simple C++ assessment is also provided in order to allow the reader to create a sample assessment of his/her own.

It is important to note that the server responsible for the automatic assessment of code, DOMjudge, is (as most of the systems that automatically assess code) based on the usage of input and output files. The programs receive specific input, through standard input, and have to produce answers to the standard output. These answers are then compared with the expected output. The result of this comparison determines the correction of the code.

The specification and format of the input and output files is responsibility of the exercise developer. Therefore, the input and output files presented during the guide are just mere examples and could have had a completely different format.

### D.1 Accessing the Test Server

The test server is available at <http://domserver.fe.up.pt/moodle/>. This server already contains some sample assessments (including the ones described in this appendix). You need to be connected to FEUP's network (you can use, for instance, a VPN connection). A teacher account can be used by logging in with username `teacher` and password `feup2010`. Some student accounts are also available, with usernames ranging from `student1` to `student20` (with password equal to the username). Using the teacher account, enter the course `Programming`, turn editing on and use the *Add an activity* box to add a new programming assessment.

## D.2 Assessment with Skeleton Code

Creating a programming assessment with skeleton code is, probably, going to be the recommended procedure when creating assignments for introductory programming courses. The skeleton code can be used for handling input and output data, allowing students to focus only in writing the code responsible for the problem solving. A complete solution for the proposed problem can be downloaded from <http://domserver.fe.up.pt/math.cpp>.

### D.2.1 General Settings

Figure D.1 presents the general settings for the sample programming assessment of this guide.

**General**

\* Hide Advanced

Name\* Math assignment

Description\*   
 Font family Font size Paragraph Styles   
 B I U ABC x<sub>2</sub> x<sup>2</sup>   
 1) Write the function `int gcd(int a, int b)` which calculates the greatest common divisor of the numbers `a` and `b`.   
 2) Write the function `bool is_prime(int a)` which tests the primality of number `a`. You may assume that `a` is less than 1000.   
 Path: p   
 HTML format

Description file Choose a file...

Available from 8 June 2010 11 30 Enable

Due date 28 June 2010 11 30 Enable

Duration 2 hours Enable

Tolerance date\* 30 June 2010 11 30 Enable

Penalty for late submissions\* 20 %

Figure D.1: Programming assessment general settings

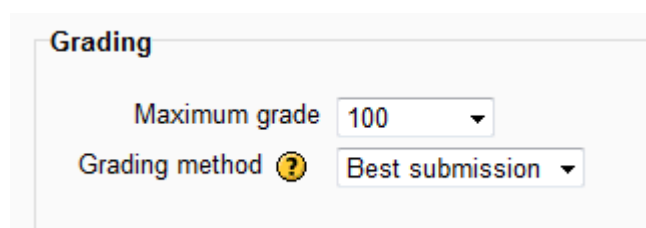
These settings comprise two mandatory fields: name and description. Optionally, the description can be provided in a description file, in a format such as a pdf, which can be downloaded by students. Then, the assessment time settings have to be specified:

- **Available from** - start date of the assessment. In the example it was set for 11h30 of the 8th June 2010;
- **Due date** - regular due date for the submission of solutions. In the example it was set for 11h30 of the 28th June 2010;
- **Duration** - duration of the assessment. After opening the assessment solution, students have this amount of time for submitting solutions. In the example it was set for 2 hours;
- **Tolerance date** - both this and the next setting can be hidden/show with the button `Hide/Show Advanced`. Submissions after the due date and before the tolerance date are still accepted, but are considered to be late and may be penalized;
- **Penalty for late submissions** - represents the penalty applied to late submissions. The penalty is multipliable (e.g. with a penalty of 10%, a late submission graded 90 has a final grade of  $90 * 0.9 = 81$ ).

### D.2.2 Grading

Figure D.2 presents the grading settings for the sample programming assessment. As can be seen, 2 setting need to be specified:

- **Maximum grade** - represents the maximum grade that can be obtained in the assessment. It can range between 1 and 100 or, alternatively, be set to the value `No grade`;
- **Grading method** - this setting can either be `Last submission`, where the last submitted solution is the one that counts for the final grade or, as used in the example, `Best submission`, where the solution with the highest grade is the one that is used.



The image shows a user interface for setting grading parameters. It is titled "Grading" in a blue header. Below the header, there are two settings, each with a label and a dropdown menu. The first setting is "Maximum grade" with a value of "100". The second setting is "Grading method" with a value of "Best submission". There is a small yellow question mark icon next to the "Grading method" label.

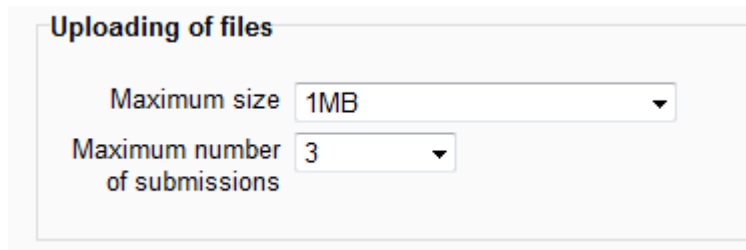
Figure D.2: Programming assessment grading settings

### D.2.3 Uploading of Files

Figure D.3 presents the uploading of files settings for the sample programming assessment. As can be seen, 2 setting need to be specified:

- **Maximum size** - represents the maximum size of the files that are going to be uploaded by students;

- **Maximum number of submissions** - represents the maximum number of attempts that can be performed by the student. It can range between 1 and 10 or, alternatively, be set to `Unlimited`;



**Uploading of files**

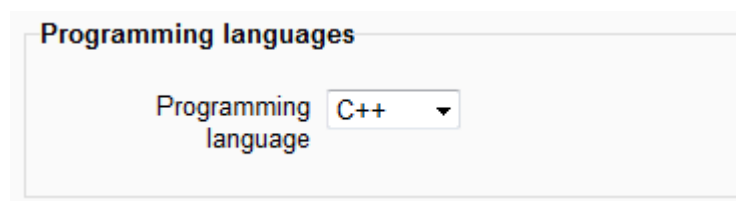
Maximum size

Maximum number of submissions

Figure D.3: Programming assessment uploading of files settings

#### D.2.4 Programming Languages

Figure D.4 presents the programming languages settings for the sample programming assessment. This section simply involves the choice of the programming language in which the assessment has to be solved. The list of available languages is dynamically loaded from the automatic assessment server.



**Programming languages**

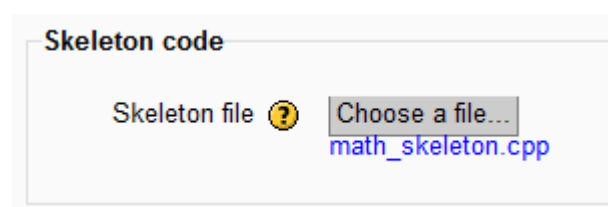
Programming language

Figure D.4: Programming assessment programming languages settings

#### D.2.5 Skeleton Code

The address [http://domserver.fe.up.pt/math\\_skeleton.cpp](http://domserver.fe.up.pt/math_skeleton.cpp) contains the file used in this example as skeleton code (figure D.5). This skeleton was obtained from the code for solving the whole assessment. The functions responsible for solving the assessment were replaced by the comment line `//studentcode`, while the code responsible for handling the input and output was kept.

Whenever a submission is made, the code is merged with the skeleton code. The `//studentcode` line is replaced by the submission code and the resulting file is compiled and tested.



**Skeleton code**

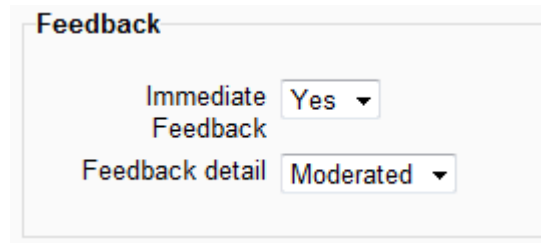
Skeleton file ?  [math\\_skeleton.cpp](#)

Figure D.5: Programming assessment skeleton code



### D.2.6 Feedback

Figure D.14 presents the feedback settings for the sample programming assessment.



The screenshot shows a 'Feedback' section with two settings:

- Immediate Feedback:** A dropdown menu currently set to 'Yes'.
- Feedback detail:** A dropdown menu currently set to 'Moderated'.

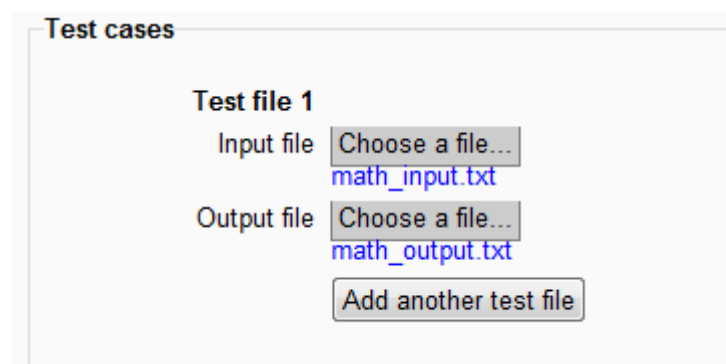
Figure D.6: Programming assessment feedback settings

As can be seen, 2 setting need to be specified:

- **Immediate feedback** - this parameter defines whether or not students should receive the results of their solutions immediately after submitting their source files. When set to **No**, the generation of the feedback reports needs to be triggered by a teacher or a course manager;
- **Feedback detail** - represents the feedback detail level given to students. The different feedback levels affect the amount of information that is generated in the feedback report. The information given is as follows:
  - **Minimalist** - global assessment grade and result in each test case;
  - **Moderated** - minimalist information plus the input used in each test case;
  - **Detailed** - moderated information plus the expected and obtained output in each test case.

### D.2.7 Test Cases

This section is used for uploading the input and output files for testing students' solutions (figure D.7).



The screenshot shows a 'Test cases' section with the following details for 'Test file 1':

- Input file:** A button 'Choose a file...' with the text 'math\_input.txt' below it.
- Output file:** A button 'Choose a file...' with the text 'math\_output.txt' below it.
- Add another test file:** A button at the bottom.

Figure D.7: Programming assessment test cases

In the sample assessment, all the test cases are specified in one input and one output file. However, they could have been split into different files.

The sample files `math_input.txt` and `math_output.txt` can be downloaded, respectively, from [http://domserver.fe.up.pt/math\\_input.txt](http://domserver.fe.up.pt/math_input.txt) and [http://domserver.fe.up.pt/math\\_output.txt](http://domserver.fe.up.pt/math_output.txt). Following, there is an explanation of the format of the input and output files.

### D.2.7.1 Input File

The sample input file contains 3 test cases. A test case has to start, mandatorily, with the line `#testcase`. Then, four optional attributes can be defined with lines starting with the symbol `#`:

- **weight** - defines the contribution of the test case to the grade of the assessment. If omitted, the weight is set to 1;
- **name** - sets the name of the test case (this is useful for identifying the test cases in the automatic reports with the results and feedback of the assessment). If omitted, a number based on the index of the test case is used as name;
- **wrong** - defines a personalized feedback message that is shown to students whenever they fail the test case;
- **right** - defines a personalized feedback message that is shown to students whenever they get the test case right;

Since the automatic assessment server uses separate files for the different test cases, the system strips out the lines that define the test cases attributes. Therefore, the skeleton code for handling input will not have to deal with the lines started with `#`.

### D.2.7.2 Output File

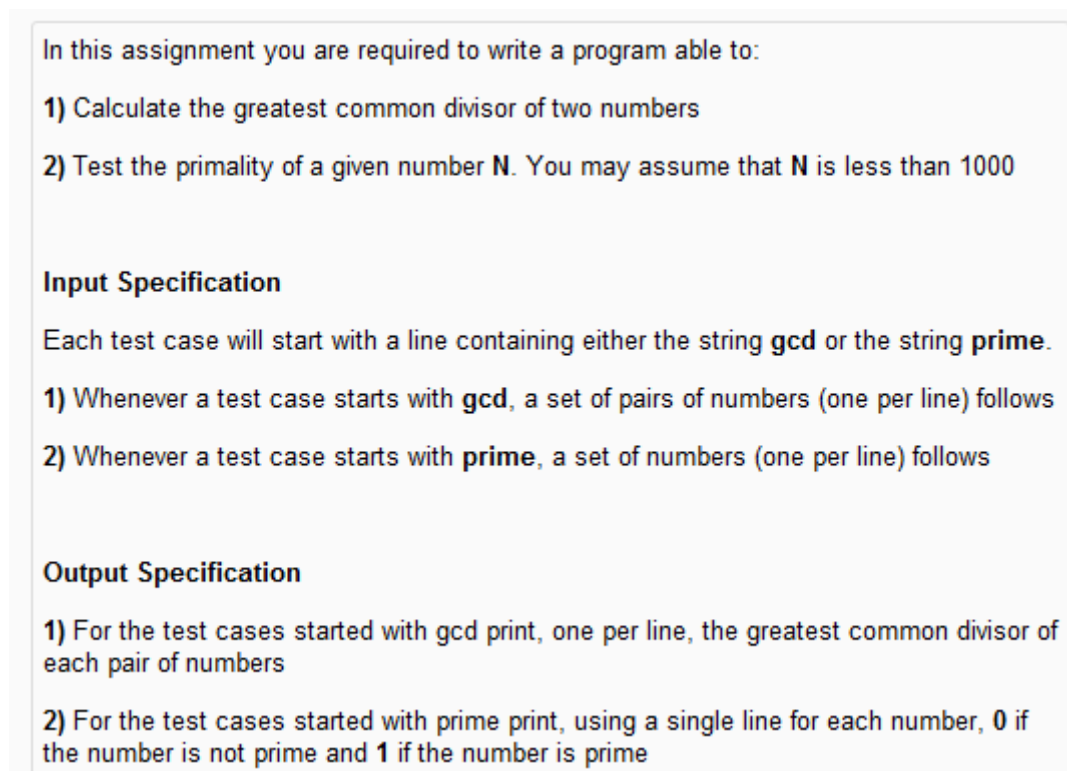
The output file contains the answers for each of the 3 test cases listed in the input file. The start of a test case output is signaled with the line `#testcase`. For the same reasons given for the input file, the system will strip out the `#testcase` lines and split the different outputs in separated files.

### D.3 Assessment without Skeleton Code

The creation of an assessment without skeleton code follows the methodology used in the problems of ACM ICPC programming contests. In such an assessment, the students will have to handle input and output data. The sample solution for this assessment can be obtained from <http://domserver.fe.up.pt/math2.cpp>.

#### D.3.1 General Settings

With the exception of the description, the general settings for this assessment are the same ones used in the previous example (section D.2.1). Since the students will now have to handle input and output data, the input and output specifications were added to the description. Moreover, students do not need to follow a strict specification, i. e. they are not required to have functions named `gcd` and `is_prime`. Therefore, the reference to these names was removed and the description became more general, as can be seen in figure D.8.



In this assignment you are required to write a program able to:

- 1) Calculate the greatest common divisor of two numbers
- 2) Test the primality of a given number **N**. You may assume that **N** is less than 1000

**Input Specification**

Each test case will start with a line containing either the string `gcd` or the string `prime`.

- 1) Whenever a test case starts with `gcd`, a set of pairs of numbers (one per line) follows
- 2) Whenever a test case starts with `prime`, a set of numbers (one per line) follows

**Output Specification**

- 1) For the test cases started with `gcd` print, one per line, the greatest common divisor of each pair of numbers
- 2) For the test cases started with `prime` print, using a single line for each number, `0` if the number is not prime and `1` if the number is prime

Figure D.8: Programming assessment description

In order to complement the input and output specification, it is also recommended to add some examples of input and output. In this example two sample test cases are provided, as shown in figure D.9

```

Sample Input 1

gcd
1 1
10 15

Sample Output 1

1
5

Sample Input 2

prime
2
3
4

Sample Output 2

1
1
0
    
```

Figure D.9: Programming assessment description - sample input and output

### D.3.2 Grading

Figure D.10 shows the grading setting used for this example.

**Grading**

Maximum grade


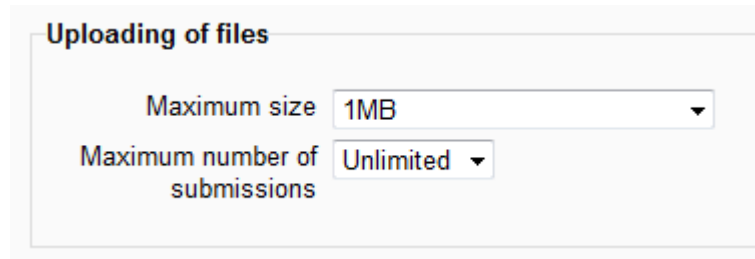
Grading method 

Figure D.10: Programming assessment grading settings

### D.3.3 Uploading of Files

Figure D.11 shows the uploading of files setting used for this example.



**Uploading of files**

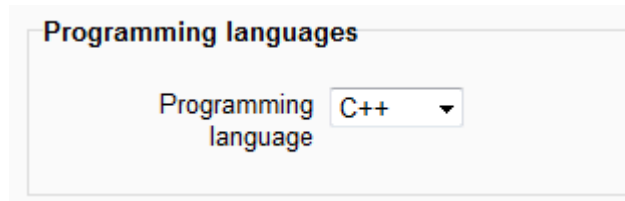
Maximum size 1MB ▼

Maximum number of submissions Unlimited ▼

Figure D.11: Programming assessment uploading of files settings

### D.3.4 Programming Languages

The programming language of this assessment is also C++ (figure D.12).



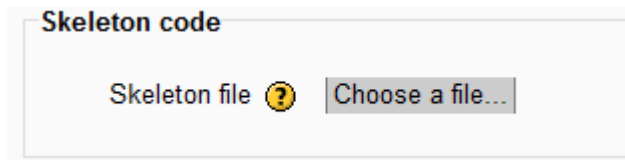
**Programming languages**

Programming language C++ ▼

Figure D.12: Programming assessment programming languages settings

### D.3.5 Skeleton Code

As expected, the skeleton code was left empty for this assessment (figure D.13).



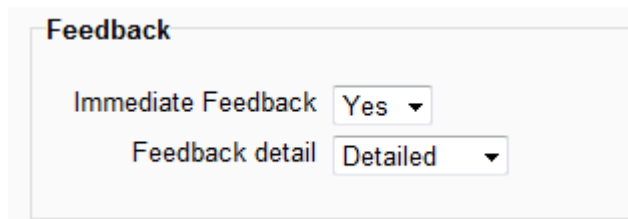
**Skeleton code**

Skeleton file ? Choose a file...

Figure D.13: Programming assessment skeleton code

### D.3.6 Feedback

Figure D.14 presents the feedback settings for this programming assessment.



**Feedback**

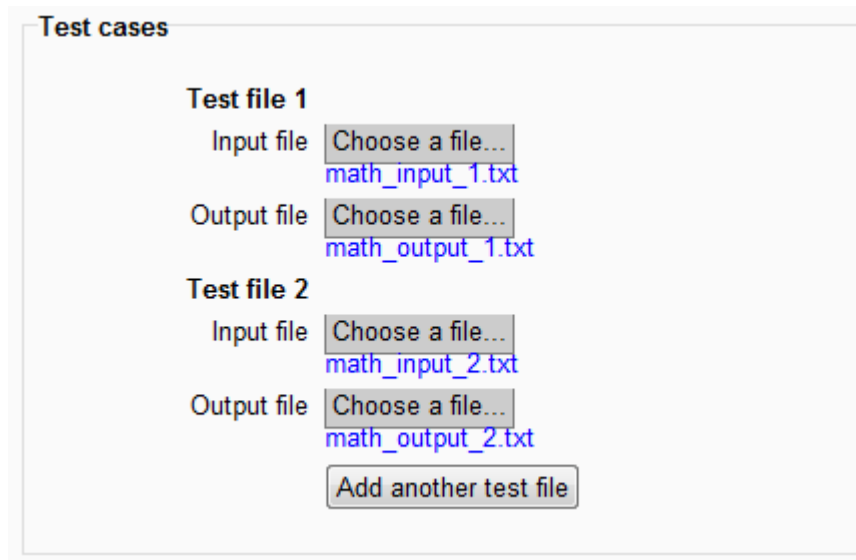
Immediate Feedback Yes ▼

Feedback detail Detailed ▼

Figure D.14: Programming assessment feedback settings

### D.3.7 Test Cases

The test cases settings are shown in figure D.15. The used input and output files can be downloaded from [http://domserver.fe.up.pt/math\\_test\\_cases.zip](http://domserver.fe.up.pt/math_test_cases.zip).



The screenshot shows a window titled "Test cases" with a light gray background. Inside, there are two sections for configuring test files. The first section, "Test file 1", has an "Input file" label followed by a button "Choose a file..." and the text "math\_input\_1.txt" below it. The "Output file" label is followed by a button "Choose a file..." and the text "math\_output\_1.txt" below it. The second section, "Test file 2", has an "Input file" label followed by a button "Choose a file..." and the text "math\_input\_2.txt" below it. The "Output file" label is followed by a button "Choose a file..." and the text "math\_output\_2.txt" below it. At the bottom of the window, there is a button labeled "Add another test file".

Figure D.15: Programming assessment test cases

## Appendix E

### A Scheme Assessment

In order to test the implemented prototype and evaluate its adequacy to the teaching of programming languages, a scheme assessment, based on a test given to students of an introductory programming course of MIEIC, was created. This test was automatically corrected by the system referred in 1.1 and its description (in Portuguese) can be downloaded from <http://paginas.fe.up.pt/~ei05058/pp2.pdf>, while the source code used to correct students' solutions can be obtained from [http://paginas.fe.up.pt/~ei05058/avaliador\\_pp2.scm](http://paginas.fe.up.pt/~ei05058/avaliador_pp2.scm).

As can be seen from the source code, in the old system there was no need to deal with input and output data. It is also possible to notice a considerable amount of replicated code throughout the correcting code. A procedure from teaching staff's solution is replicated whenever it is needed in a test case. This is useful when a test case depends on procedures from previous questions. In these situations, teaching staff's solution is used instead of student's solutions, assuring that students are not penalized for incorrect answers in previous questions.

In spite of mandatorily requiring the manipulation of input and output data, the new system introduces flexibility in the way the exercises can be assessed. The input and output data, as well as the correcting code, can be specified in many different ways. Following there is an explanation of two possible approaches. These approaches have input and output data with different configurations and hence the skeleton code responsible for handling the data is also different. However, they have similarities:

- The skeleton code starts with the teaching staff solution. The solution procedures are global in all the test cases and therefore there is no need for code replication. However, since the solution code is combined with students' code in a single file which is then compiled, the solution procedures cannot have the same name as students' procedures (in the examples, the solution procedures end with `-sol`). Teaching staff is responsible for using the appropriate procedures in the code for handling input and output and assess solutions;
- The usage of DOMjudge to automatically assess solutions requires the compilation of the code. This is currently being done with MzScheme. This package requires the code to be included in a module. To be compatible with DOMjudge, the module should be named `source` and use the language `mzscheme`. The MzScheme package is a subset of the PLT Scheme package and therefore some language constructions cannot be used. For instance, the usage of `else` in a `cond` expression

is recognized as unbound identifier. For more information on this, please consult <http://download.plt-scheme.org/doc/4.2.4/pdf/mzscheme.pdf>.

Two assignments, one for each approach and called `Scheme Assessment v1` and `Scheme Assessment v2`, were set up and can be accessed on the test server referred in [D.1](#).

## E.1 Approach 1

The configuration files used in this example can be downloaded from [http://paginas.fe.up.pt/~ei05058/scheme\\_assessment\\_v1.zip](http://paginas.fe.up.pt/~ei05058/scheme_assessment_v1.zip). The file `skeleton.scm` represents the skeleton code, while the file `student_code.scm` represents a possible student solution (which is, in this case, equal to the solution used in the skeleton code). The test cases were split among 7 input and output files, one for each question of the assessment.

In this approach, the code responsible for handling the input and test solutions was conceived to be generic, i.e., the same code is used to test a single procedure in different test cases. For instance, the code used to correct question 5 is prepared to receive a player name, a predefined number of cards, a suit name and then test the procedure `naipe-jogador`. The usage of generic code implies the need for relatively elaborated input files, containing the actual data used for the tests.

In order to understand the data flow that occurs when running a test case, let's take a look at what happens when the third test of the file `pp2_input5.txt` is run. The input data for this test is as follows:

Listing E.1: Input data of the third test of the file `pp2_input5.txt`

```
"pergunta5 "
"Ana"
2
"as" "ouros"
10 "espadas"
"paus"
```

The first code to run in the program is the procedure `main`:

Listing E.2: Procedure `main`

```
(define main
  (lambda ()
    (let ((str (read)))
      (cond ((eof-object? str) (exit 0))
            ((string=? str "pergunta1") (pergunta1))
            ((string=? str "pergunta2") (pergunta2))
            ((string=? str "pergunta3") (pergunta3))
            ((string=? str "pergunta4") (pergunta4))
            ((string=? str "pergunta5") (pergunta5))
            ((string=? str "pergunta6") (pergunta6))
            ((string=? str "pergunta7") (pergunta7))))))
```



```
(main))
```

The first input line will lead to the invocation of procedure `pergunta5`:

Listing E.3: Procedure `pergunta5`

```
(define pergunta5
  (lambda()
    (let ((nome (read))
          (n_cartas (read)))
      (pergunta5_aux (cria-jogador-sol nome) 0 n_cartas))))

(define pergunta5_aux
  (lambda(jogador cartas_lidas total_cartas)
    (cond ((= cartas_lidas total_cartas)
           ((let ((naipe (read)))
              (display (naipe-jogador jogador naipe))
              (newline)
              (main))))
          ((let ((valor (read))
                  (naipe (read)))
              (pergunta5_aux
               (adiciona-carta-jogador-sol jogador
                (cria-carta-sol valor naipe))
               (+ cartas_lidas 1)
               total_cartas)))))))
```

The auxiliary procedure `pergunta5_aux` reads the 2 cards specified in the input and then tests the procedure `naipe-jogador` with the value "paus".

## E.2 Approach 2

The configuration files used in this example can be downloaded from [http://paginas.fe.up.pt/~ei05058/scheme\\_assessment\\_v2.zip](http://paginas.fe.up.pt/~ei05058/scheme_assessment_v2.zip). The filenames are the same ones used for the first approach.

In opposition to the previous approach, the code responsible for handling the input and test solutions is not generic. Now, each test case has a procedure responsible for preparing its data and test students' solutions. This leads to the simplification of input and output data, while the size of the skeleton code increases.

In order to understand the data flow that occurs when running a test case, let's take a look at what happens when the third test of the file `pp2_input5.txt` (the same used for the first approach) is run. The input data for this test is as follows:

Listing E.4: Input data of the third test of the file `pp2_input5.txt`

```
"pergunta5 "
3
```

This data simply triggers the procedure `pergunta5_3`, responsible for the third test case of question 5:

Listing E.5: Procedure `pergunta5`

```
(define pergunta5
  (lambda ()
    (let ((teste (read)))
      (cond ((= teste 1) (pergunta5_1))
            ((= teste 2) (pergunta5_2))
            ((= teste 3) (pergunta5_3))
            ((= teste 4) (pergunta5_4))))))

(define pergunta5_3
  (lambda ()
    (display (equal? (naipe-jogador-sol
                      (adiciona-carta-jogador-sol
                       (adiciona-carta-jogador-sol
                        (cria-jogador-sol 'Ana)
                        (cria-carta-sol 'as 'ouros))
                       (cria-carta-sol 10 'espadas)) 'paus)
                     (naipe-jogador
                      (adiciona-carta-jogador-sol
                       (adiciona-carta-jogador-sol
                        (cria-jogador-sol 'Ana)
                        (cria-carta-sol 'as 'ouros))
                       (cria-carta-sol 10 'espadas)) 'paus)))
      (newline)))
```

Procedure `pergunta5_3` is responsible for creating a new player, adding the two same cards that were added in the previous approach and comparing the results of teaching staff solution with students' solutions. This methodology pretty much resembles the usage of unit tests. However, the comparison results have to be displayed in the standard output.

## Appendix F

# Solution Submission Guide

This appendix contains a guide explaining the user interface for submitting solutions to programming assessments. The guide uses the Scheme assessment defined in the first approach of appendix E and available on the test server described in D.1.

The submission interface is divided in multiple sections that can be expanded and collapsed. When accessing the Scheme assessment, a student, who still have not submitted a solution, is confronted with the information of figure F.1.

▼Programming Assessment: Scheme Assessment v1

Available from: Friday, 11 June 2010, 12:45 AM

Due date: Thursday, 30 December 2010, 12:45 AM

Duration: No restrictions

Maximum grade: 100

Grading method: Last submission


Programming languages: Scheme

Maximum number of submissions: Unlimited

Immediate Feedback: Yes

Feedback detail: Detailed

▼Description

 pp2.pdf

►Submission

►Compilation playground

Figure F.1: Solution submission general view

The first section of the interface contains general information regarding the assessment. In this example, the description was given in a pdf file which can be downloaded by students. Having read the assessment description and solved the problem in a local workspace, the student can then compile the code on the compilation playground section (see figure F.2). The file used in this example can be obtained in the zip file located at [http://paginas.fe.up.pt/~ei05058/scheme\\_assessment\\_v1.zip](http://paginas.fe.up.pt/~ei05058/scheme_assessment_v1.zip).

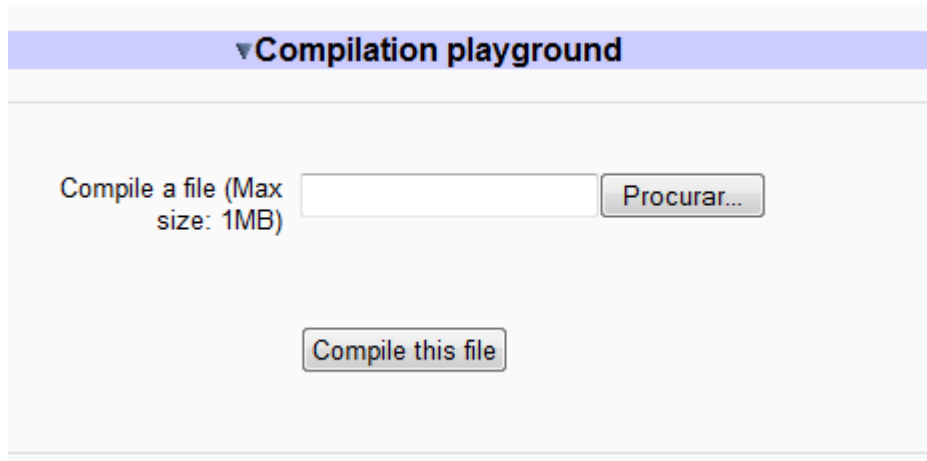


Figure F.2: Compilation playground

As can be seen in figure F.3 the uploaded code compiled successfully. When a compiler error occurs, instead of the `Compilation was successful` message, the compiler output is shown. The compilation result does not immediately appear. It may take from a few seconds to a couple of minutes to show up and the page needs to be manually refreshed. This should be improved in future versions of the system, by auto refreshing the compilation section with JavaScript.

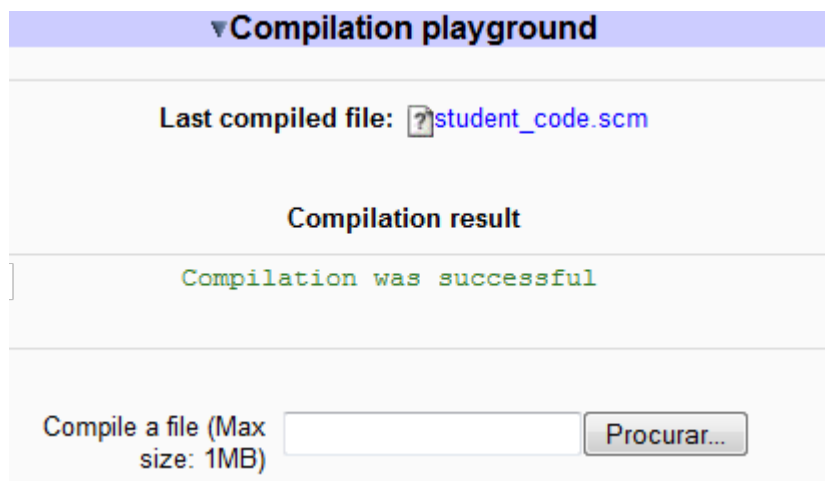
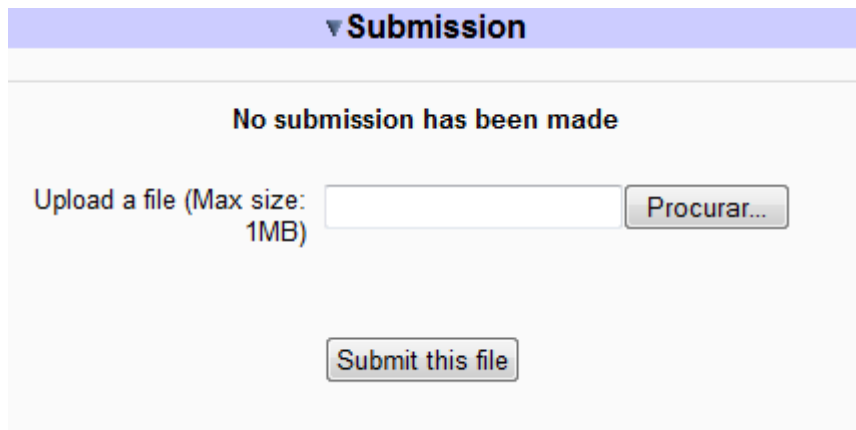


Figure F.3: Compilation playground - successful compilation

A successful compilation message does not assure the correction of the solution and hence it is recommended to test the code locally with some test data. Having tested the code, a submission can be made in the submit section of the interface (see figure F.4).



The screenshot shows a web interface for solution submission. At the top, there is a purple header bar with the text "▼ Submission". Below this, a message states "No submission has been made". Underneath the message, there is a text input field with the label "Upload a file (Max size: 1MB)". To the right of the input field is a button labeled "Procurar...". Below the input field and button, there is a button labeled "Submit this file".

Figure F.4: Solution submission

Similarly to what happens in the compilation process, the submission result does not immediately appear. It may take from a few seconds to a few minutes, depending on the number and complexity of test cases used in the assessment.

When the submission results are available, a new section with title Feedback appears on the interface (figure F.5). Whenever more than one submission has been made, it is possible to choose what submission to show. The feedback report is composed by the grade achieved in the assessment, plus the results in the different test cases. The amount of information presented for each test case depends on the feedback detail. In the example the feedback was set to be detailed and then, for each test case, it is presented the result of the automatic assessment, the input and output test data and the obtained output.

Whenever a teacher or a course manager accesses the submission page of a programming assessment, besides all the previously presented sections, he/she is also presented with a section containing information about the input and output data of the test cases used to test the submitted solutions. As can be seen in figure F.6, for each test case, the weight, input and output files are shown.

Feedback

Select submission

student\_code.scm - 15:03:23 25-06-2010 \*

View this submission

Currently showing feedback for submission: student\_code.scm - 15:03:23 25-06-2010

Grade

100 out of 100

Full test cases results

Test case teste cria-carta

Weight:

1

Result:

correct

Input:

input.txt

Expected output:

output.txt

Obtained output:

obtained\_output.txt

Figure F.5: Submission feedback

▼ Test case teste cria-carta (Weight: 1)

Input

```
"pergunta1"
"as" "paus"
```

input.txt

Output

```
(as . paus)
```

output.txt

▶ Test case teste cria-carta (Weight: 1)

▶ Test case teste cria-carta (Weight: 1)

▶ Test case teste cria-carta (Weight: 1)

▶ Test case teste cria-carta (Weight: 2)

Figure F.6: Input and output data

## Solution Submission Guide



## Appendix G

# Communication Protocol

Listing G.1: Front End Service Description

```
<definitions name="ProgAssessment"
  targetNamespace=
    "http://domserver.fe.up.pt/domjudge/frontend/frontend.wsdl"
  xmlns:tns=
    "http://domserver.fe.up.pt/domjudge/frontend/frontend.wsdl"
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:xsd1=
    "http://domserver.fe.up.pt/domjudge/frontend/frontend.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

<xsd:complexType name="ArrayOfstring">
  <xsd:complexContent>
    <xsd:restriction base="soapenc:Array">
      <xsd:attribute ref="soapenc:arrayType"
        wsdl:arrayType="xsd:string[]" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<!-- ##### setupProgassessmentModule #####-->

<message name="setupProgassessmentModuleResponse">
  <part name="result" type="xsd:int" />
</message>

<!--##### getLanguages #####-->

<message name="getLanguagesResponse">
  <part name="result" type="ArrayOfstring" />
</message>
```

```

<!--##### getAllLanguagesInfo #####-->

<message name="getAllLanguagesInfoResponse">
  <part name="result" type="ArrayOfstring"/>
</message>

<!--##### addNewAssessment #####-->

<message name="addNewAssessmentRequest">
  <part name="id" type="xsd:int"/>
  <part name="name" type="xsd:string"/>
  <part name="timeLimit" type="xsd:int"/>
</message>

<message name="addNewAssessmentResponse">
  <part name="Result" type="xsd:int"/>
</message>

<!--##### removeAssessment #####-->

<message name="removeAssessmentRequest">
  <part name="id" type="xsd:int"/>
</message>

<!--##### updateAssessment #####-->

<message name="updateAssessmentRequest">
  <part name="id" type="xsd:int"/>
  <part name="name" type="xsd:string"/>
  <part name="timeLimit" type="xsd:int"/>
  <part name="nTestCases" type="xsd:int"/>
</message>

<!--##### addTestCase #####-->

<message name="addTestCaseRequest">
  <part name="probid" type="xsd:string"/>
  <part name="input" type="xsd:string"/>
  <part name="output" type="xsd:string"/>
  <part name="id" type="xsd:int"/>
</message>

<message name="addTestCaseResponse">
  <part name="id" type="xsd:int"/>
</message>

```

```

<!--##### removeTestCase #####-->

<message name="removeTestCaseRequest">
  <part name="id" type="xsd:int"/>
</message>

<!--##### updateTestCase #####-->

<message name="updateTestCaseRequest">
  <part name="id" type="xsd:int"/>
  <part name="input" type="xsd:string"/>
  <part name="output" type="xsd:string"/>
</message>

<!--##### addParticipant #####-->

<message name="addParticipantRequest">
  <part name="login" type="xsd:string"/>
  <part name="name" type="xsd:string"/>
</message>

<!--##### participantExists #####-->

<message name="participantExistsRequest">
  <part name="login" type="xsd:string"/>
</message>

<message name="participantExistsResponse">
  <part name="result" type="xsd:int"/>
</message>

<!--##### addSubmission #####-->

<message name="addSubmissionRequest">
  <part name="participantLogin" type="xsd:string"/>
  <part name="assessmentId" type="xsd:int"/>
  <part name="testCasesIds" type="xsd:Array"/>
  <part name="language" type="xsd:string"/>
  <part name="sourceCode" type="xsd:string"/>
</message>

<message name="addSubmissionResponse">
  <part name="ids" type="xsd:Array"/>
</message>

<!--##### addCompileSubmission #####-->

```

```

<message name="addCompileSubmissionRequest">
  <part name="participantLogin" type="xsd:string"/>
  <part name="assessmentId" type="xsd:int"/>
  <part name="language" type="xsd:string"/>
  <part name="sourceCode" type="xsd:string"/>
</message>

<message name="addCompileSubmissionResponse">
  <part name="id" type="xsd:int"/>
</message>

<!--##### getSubmissionResult #####-->

<message name="getSubmissionResultRequest">
  <part name="id" type="xsd:int"/>
</message>

<message name="getSubmissionResultResponse">
  <part name="result" type="ArrayOfstring"/>
</message>

<!--##### Port #####-->

<portType name="ProgAssessmentPortType">

  <operation name="setupProgassessmentModule">
    <output message="setupProgassessmentModuleResponse"/>
  </operation>

  <operation name="getLanguages">
    <output message="getLanguagesResponse"/>
  </operation>

  <operation name="getAllLanguagesInfo">
    <output message="getAllLanguagesInfoResponse"/>
  </operation>

  <operation name="addNewAssessment">
    <input message="addNewAssessmentRequest"/>
    <output message="addNewAssessmentResponse"/>
  </operation>

  <operation name="removeAssessment">
    <input message="removeAssessmentRequest"/>
  </operation>

```

```

<operation name="updateAssessment">
  <input message="updateAssessmentRequest"/>
</operation>

<operation name="addTestCase">
  <input message="addTestCaseRequest"/>
  <output message="addTestCaseResponse"/>
</operation>

<operation name="removeTestCase">
  <input message="removeTestCaseRequest"/>
</operation>

<operation name="updateTestCase">
  <input message="updateTestCaseRequest"/>
</operation>

<operation name="addParticipant">
  <input message="addParticipantRequest"/>
</operation>

<operation name="participantExists">
  <input message="participantExistsRequest"/>
  <output message="participantExistsResponse"/>
</operation>

<operation name="addSubmission">
  <input message="addSubmissionRequest"/>
  <output message="addSubmissionResponse"/>
</operation>

<operation name="addCompileSubmission">
  <input message="addCompileSubmissionRequest"/>
  <output message="addCompileSubmissionResponse"/>
</operation>

<operation name="getSubmissionResult">
  <input message="getSubmissionResultRequest"/>
  <output message="getSubmissionResultResponse"/>
</operation>
</portType>

<!--##### Bindings #####-->

<binding name="ProgAssessmentBinding"

```

```

    type="ProgAssessmentPortType">

<soap:binding style='rpc'
    transport='http://schemas.xmlsoap.org/soap/http'/>

<operation name="setupProgassessmentModule">
    <soap:operation soapAction=
        "http://domserver.fe.up.pt/domjudge/frontend"/>
    <output> <soap:body use="literal"/> </output>
</operation>

<operation name='getLanguages'>
    <soap:operation soapAction=
        "http://domserver.fe.up.pt/domjudge/frontend"/>
    <output> <soap:body use="literal"/> </output>
</operation>

<operation name='getAllLanguagesInfo'>
    <soap:operation soapAction=
        "http://domserver.fe.up.pt/domjudge/frontend"/>
    <output> <soap:body use="literal"/> </output>
</operation>

<operation name='addNewAssessment'>
    <soap:operation soapAction=
        "http://domserver.fe.up.pt/domjudge/frontend"/>
    <input> <soap:body use="literal"/> </input>
    <output> <soap:body use="literal"/> </output>
</operation>

<operation name='removeAssessment'>
    <soap:operation soapAction=
        "http://domserver.fe.up.pt/domjudge/frontend"/>
    <input> <soap:body use="literal"/> </input>
</operation>

<operation name='updateAssessment'>
    <soap:operation soapAction=
        "http://domserver.fe.up.pt/domjudge/frontend"/>
    <input> <soap:body use="literal"/> </input>
</operation>

<operation name='addTestCase'>
    <soap:operation soapAction=
        "http://domserver.fe.up.pt/domjudge/frontend"/>
    <input> <soap:body use="literal"/> </input>
    <output> <soap:body use="literal"/> </output>

```

```

</operation>

<operation name='removeTestCase'>
  <soap:operation soapAction=
    "http://domserver.fe.up.pt/domjudge/frontend"/>
  <input> <soap:body use="literal"/> </input>
</operation>

<operation name='updateTestCase'>
  <soap:operation soapAction=
    "http://domserver.fe.up.pt/domjudge/frontend"/>
  <input> <soap:body use="literal"/> </input>
</operation>

<operation name='addParticipant'>
  <soap:operation soapAction=
    "http://domserver.fe.up.pt/domjudge/frontend"/>
  <input> <soap:body use="literal"/> </input>
</operation>

<operation name='participantExists'>
  <soap:operation soapAction=
    "http://domserver.fe.up.pt/domjudge/frontend"/>
  <input> <soap:body use="literal"/> </input>
  <output> <soap:body use="literal"/> </output>
</operation>

<operation name='addSubmission'>
  <soap:operation soapAction=
    "http://domserver.fe.up.pt/domjudge/frontend"/>
  <input> <soap:body use="literal"/> </input>
  <output> <soap:body use="literal"/> </output>
</operation>

<operation name='addCompileSubmission'>
  <soap:operation soapAction=
    "http://domserver.fe.up.pt/domjudge/frontend"/>
  <input> <soap:body use="literal"/> </input>
  <output> <soap:body use="literal"/> </output>
</operation>

<operation name='getSubmissionResult'>
  <soap:operation soapAction=
    "http://domserver.fe.up.pt/domjudge/frontend"/>
  <input> <soap:body use="literal"/> </input>
  <output> <soap:body use="literal"/> </output>
</operation>

```

```
</binding>

<service name='ProgAssessmentService'>
  <port name='ProgAssessmentPort'
        binding='ProgAssessmentBinding'>
    <soap:address
      location=
        'http://domserver.fe.up.pt/domjudge/frontend/frontend.php' />
    </port>
  </service>

</definitions>
```