

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Computer-Based Assessment System for E-learning Applied to Programming Education

João Cristóvão Xavier

Master in Informatics and Computing Engineering

Supervisor: António Fernando Coelho (Assistant Professor)

11th July, 2011

Computer-Based Assessment System for E-learning Applied to Programming Education

João Cristóvão Xavier

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Doctor Jorge Alves da Silva (Assistant Professor of FEUP¹)

External Examiner: Doctor João Paulo F. da Costa Moura (Assistant Professor of UTAD²)

Supervisor: Doctor António Fernando V. C. Castro Coelho (Assistant Professor of FEUP¹)

11th July, 2011

¹Faculdade de Engenharia da Universidade do Porto

²Universidade de Trás-os-Montes e Alto Douro

Abstract

The e-learning systems are, currently, extremely useful tools in education, helping to reduce the distance between students and teachers and providing better methods for monitoring the learning process. These systems can play a particularly important role in programming courses, due to students needing to solve a high number of exercises while learning from their mistakes.

The overall learning process can be improved by the use of an automated tool, since it can present the student the feedback he needs to refine his programming skill. Moreover, teachers will benefit from not having to carry these tasks manually, in addition to easing the process of making assessments.

This dissertation comprises an analysis of the prototype of a Computer-Based Assessment (CBA) system developed for supporting the programming components of the courses of DEI, FEUP and a study of advanced features for systems of this kind. While currently implementing a set of important features, the platform was deemed as insufficiently prepared to be actively used due to lacking more complex functionalities.

A study of state-of-the-art CBA systems was performed in order to compare the features they implement to the base CBA system developed, so that the main points which needed to be addressed could be identified. Between the advanced features outlined, the ones that distinguish themselves are the plagiarism detection and the static analysis of code quality. The former aims to motivate students to learn by themselves, while the latter to help students improve their programming style and skill. Additionally, a study on a range of common metrics which contribute the most for a student's learning followed, together with different plagiarism detection approaches. This analysis allowed to compile different solution perspectives, build an advanced module specification as well as a new system architecture.

The CBA system was used in a contest thrown by teachers of a course unit and a report on this case study was assembled, yielding important information on the system hardware limitations. Besides, the developed modules were implemented, tested and integrated with extensibility in mind, providing guidelines on how they can be expanded and how to improve a few incomplete advanced features.

Regarding the results, it was concluded that the CBA system can be used in a real working environment, providing it is installed in production servers and all users are warned that the platform is still in test. However, in order to further unleash the educational potential of the system, the advanced modules could benefit from some optimizations and extended functionalities.

Resumo

Actualmente, os sistemas de e-learning são ferramentas extremamente úteis no ensino, permitindo reduzir a distância entre estudantes e professores e fornecer melhores métodos para monitorizar o processo de aprendizagem. Estes sistemas podem desempenhar um papel particularmente importante em disciplinas na área da programação, devido à necessidade dos estudantes de resolver um número elevado de exercícios.

O processo de aprendizagem geral pode ser melhorado através do uso de uma ferramenta automática, visto permitir apresentar ao aluno o *feedback* necessário para refinar as suas capacidades de programação. Além disso, o corpo docente beneficia de não ter que desempenhar estas tarefas manualmente, bem como de uma maior facilidade na concepção de provas.

Esta dissertação é constituída pela análise do protótipo de um Sistema de Avaliação Automática (SAA) desenvolvido para suportar as componentes de programação das unidades curriculares do DEI, FEUP e ainda um estudo sobre módulos avançados para sistemas deste cariz. Apesar de actualmente implementar um conjunto de funcionalidades importantes, a plataforma não foi considerada como preparada para ser usada activamente visto carecer de ferramentas mais complexas.

Foi efectuado um estudo de sistemas deste tipo de forma a comparar as funcionalidades que estes implementam com as do SAA desenvolvido, para que os principais pontos de foco pudessem ser determinados. Entre as funcionalidades avançadas identificadas, as que sobressaem são a detecção de plágio e a análise estática da qualidade do código. Com a primeira espera-se que os alunos se sintam mais motivados em aprender por si, e com a última pretende-se ajudar os alunos a melhorar a sua técnica e estilo de programação. Estudaram-se ainda métricas de *software* comuns, tendo em particular conta as que mais contribuem para a aprendizagem de um aluno, juntamente com diferentes algoritmos de detecção de plágio. Esta análise permitiu delinear diferentes perspectivas de solução, construir uma especificação para cada módulo avançado, bem como uma nova arquitectura de sistema.

O SAA foi usado num concurso proposto por professores de uma unidade curricular e concebeu-se um relatório deste caso de estudo, contendo informação sobre as limitações do sistema a nível de *hardware*. Os módulos avançados desenvolvidos foram implementados, testados e integrados tendo em vista a extensibilidade, sendo facultadas directivas em como podem ser melhorados.

Tendo em conta os resultados, concluiu-se que o SAA pode ser usado activamente se for instalado em servidores de produção e se os utilizadores forem avisados que a plataforma ainda não se encontra na sua versão definitiva. No entanto, de modo a maximizar o potencial educativo deste sistema, os módulos avançados beneficiariam de algumas optimizações e funcionalidades adicionais.

Acknowledgements

I would like to thank everyone who have contributed to this dissertation, both directly and indirectly.

Firstly, my thanks go to my supervisor, Prof. António Coelho, for all the suggestions, ideas and the assistance in improving the structure of this document, essential to keep this thesis in a good track.

I would like to add further thanks for the suggestions and help of both Prof. Gabriel David and Prof. Filipe Correia in setting up a case study to test the platform.

My acknowledgements to Pedro Pacheco for providing me important documentation on his past work, which laid the foundation for this thesis.

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

My gratitude for the joint contribution with Enrique Kato and Ricardo Gonçalves in the article written for the Second International Conference on Serious Games Development and Applications and for sharing their opinions during our meetings.

A word of appreciation goes to André Silva for aiding me in choosing some of the technologies to use.

I cannot thank Joana Lima enough, who greatly helped me keep the whole thesis in the right course with constant reviewing, motivation and, most importantly, her love.

I would like to express my deep gratitude to my family for their love and support, at all times.

Finally, thanks to you who are involved in my life everyday and provide your insight and ideas, as well as a good pack of laughs. May we continue to share our experiences in the future and hopefully work together.

João Xavier

*“I do not fear computers.
I fear the lack of them.”*

Isaac Asimov

Contents

1	Introduction	1
1.1	Background and Motivation	1
1.2	Problem Statement	1
1.3	Goals	2
1.4	Expected Results	3
1.5	Related Work	3
1.6	Methodology	3
1.7	Document Structure	4
2	The Base CBA System	5
2.1	Features Implemented	5
2.1.1	Support for Assessment Creation	5
2.1.2	Support for Test Cases Definition	6
2.1.3	Support for Elaboration and Submission of Solutions	6
2.1.4	Feedback System	6
2.1.5	Submissions Management	6
2.1.6	Extra Features	6
2.2	Comparison with Other CBA Systems	7
2.3	System Architecture	11
2.4	Limitations of the Current Implementation	11
2.5	Summary	13
3	State of the Art	15
3.1	Static Analysis of Code Quality	15
3.1.1	Approaches Taken by Other CBA Systems	16
3.1.2	Metrics	19
3.1.3	Summary	22
3.2	Plagiarism Detection	23
3.2.1	Approaches Taken by Other CBA Systems	24
3.2.2	Algorithms	27
3.2.3	Summary	29
4	Integration of Advanced Functionalities in CBA Systems	33
4.1	Applicability	34
4.1.1	Requirements	34
4.1.2	Assignment versus Module	35
4.1.3	Plagiarism API	36

CONTENTS

4.2	Proposed Architecture	39
4.3	Advanced Module Specification	40
5	Implementation	43
5.1	Full System Architecture	44
5.1.1	Hardware Specifications	44
5.1.2	Software Used	44
5.1.3	Installation Process	45
5.1.4	Integration with Moodle 2.0	45
5.2	Integrating New Programming Languages in the CBA System	46
5.2.1	Input and Output	46
5.2.2	Connection Library	48
5.2.3	Running the Program	49
5.2.4	Moodle Integration	50
5.3	Throwing a Contest with DOMjudge	51
5.3.1	Before Starting	51
5.3.2	Mid-contest	52
5.3.3	Gathering Results	52
5.4	Static Analysis of Code Quality	52
5.4.1	Parsing Techniques	53
5.4.2	Metrics Development	54
5.4.3	Metrics Library	58
5.4.4	Integration with DOMjudge	59
5.4.5	Integration with Moodle	60
5.4.6	Extensibility	62
5.5	Plagiarism Detection	63
5.5.1	Choice	63
5.5.2	Crot Analysis	64
5.5.3	Adapting the Programming Assessment Module	66
5.5.4	Adapting Crot	68
6	Results	73
6.1	Implemented Features	73
6.2	Tests	74
6.2.1	SQL Contest	74
6.2.2	Metric Unit Testing	75
6.2.3	Plagiarism Detection Effectiveness	80
6.3	Summary	82
7	Conclusions and Future Work	83
7.1	Work Summary	83
7.2	Future Work	85
	References	87

CONTENTS

A	Installation Guide	93
A.1	Configuring the Main Automatic Assessment Server	93
A.2	Installing a Judgehost	94
A.3	Installing the Moodle Server	94
B	Programming Language Configuration Guide	95
B.1	DOMjudge Configuration	95
B.2	Moodle Configuration	96
B.3	Adding Support to SQL	97
B.3.1	Installation of the Oracle Call Interface Driver	97
B.3.2	Adding the Oracle Scripts to the Judgehosts	98
C	Configuration of the Static Analysis Module	99
D	Configuration of the Plagiarism Detection Module	101
E	Tests	103
E.1	Files Used for Metric Unit Testing	103
E.2	Submitted Files for Plagiarism Detection	104
F	Assessment Creation Using the Advanced Functionalities	121
F.1	Static Analysis of Code Quality	121
F.2	Plagiarism Detection	121
G	Communication Protocol	123

CONTENTS

List of Figures

2.1	Base CBA system architecture (Source: [Pac10] p.71)	12
3.1	Sample control flow graph (Source: [Wik11])	20
3.2	Simplified classification of plagiarism detection systems (Source: [Moz08] p.17)	27
3.3	Speed and reliability of different plagiarism detection schemes (Source: [Moz08] p.38)	30
4.1	Advanced uploading of files	36
4.2	Assignment grading screen	37
4.3	New CBA system architecture	39
5.1	Scoreboard of a finished SQL contest	53
5.2	Static analysis interface when no metrics are available in the Programming Assessment form	60
5.3	Collapsed static analysis interface in the Programming Assessment form	60
5.4	Expanded static analysis interface in the Programming Assessment form	61
5.5	Static analysis grade interface	62
5.6	Crot main steps for local and global search (Source: [BS09] p.783)	65
5.7	Crot assignment global similarity (Source: abridged from [But11])	65
5.8	Student disclosure in Programming Assessment	67
5.9	Number of submitted assessments in Programming Assessment	67
5.10	Options in the grading view of Programming Assessment	68
5.11	Optional settings in the grading view of Programming Assessment	68
5.12	Full grading view of Programming Assessment	69
5.13	Crot similarity report view in Programming Assessment	70
5.14	Crot compare view in Programming Assessment	71
6.1	Contest submission outcome chart	75
6.2	C program example #1	76
6.3	C program example #2	78
6.4	C program example #3	79
A.1	Adding an entry to the table <i>contest</i>	93
A.2	Adding an entry to the table <i>judgehost</i>	94
B.1	Adding an entry to the table <i>language</i>	95
B.2	Installing <i>oci8</i> using <i>pecl</i>	97

LIST OF FIGURES

B.3	Adding the <i>oci8</i> extension in the configuration files	98
F.1	Enabling Crot in a programming assessment	121

List of Tables

2.1	Comparison of CBA systems features: types of supported assessments. . .	7
2.2	Comparison of CBA systems features: parameters for assessment configuration.	7
2.3	Comparison of CBA systems features: methodologies for test-cases definition.	8
2.4	Comparison of CBA systems features: parameters for test-case configuration.	8
2.5	Comparison of CBA systems features: support for elaboration and submission of solutions.	9
2.6	Comparison of CBA systems features: feedback system.	9
2.7	Comparison of CBA systems features: management of submissions. . . .	10
2.8	Comparison of CBA systems features: extras.	10
5.1	Comparison of different plagiarism detection implementations.	64
6.1	Similarities between the modifications of the source file <code>binary.c</code>	81

LIST OF TABLES

Abbreviations

ACM	Association for Computing Machinery
AJAX	Asynchronous Javascript And XML
API	Application Programming Interface
AST	Abstract Syntax Tree
BOSS	BOSS Online Submission System
CBA	Computer-Based Assessment
CICA	Centro de Informática Correia de Araújo
CMT	Complexity Measures Tool
CPU	Central Processing Unit
DBMS	Database Management System
DEI	Departamento de Engenharia Informática
FCUP	Faculdade de Ciências da Universidade do Porto
FEUP	Faculdade de Engenharia da Universidade do Porto
FSB	Front-Side Bus
GAME	Generic Automated Marking Environment
GB	GigaByte
GCC	GNU Compiler Collection
GNU	GNU's Not Unix!
GST	Greedy-String-Tiling
GUI	Graphical User Interface
HTML	HyperText Markup Language
ICPC	International Collegiate Programming Contest
LCI	Licenciatura em Ciência da Informação
LLVM	Low Level Virtual Machine
MOSS	Measure Of Software Similarity
OCI	Oracle Call Interface
OS	Operating System
PHP	PHP: Hypertext Preprocessor
RAM	Random Access Memory
RKR	Running-Karp-Rabin
SAA	Sistema de Avaliação Automática
SIGEX	SIstema de Gestão de EXames
SQL	Structured Query Language
UML	Unified Modelling Language
VM	Virtual Machine
XLX	eXtreme e-Learning eXperience
XML	eXtensible Markup Language
YAP	Yet Another Plague

ABBREVIATIONS

Chapter 1

Introduction

1.1 Background and Motivation

The key areas of this dissertation are e-learning and Information and Communication Technologies.

The e-learning systems are, currently, extremely useful tools in education. They not only enable the removal of distance problems but also communication problems, providing a variety of interaction possibilities between the students and the teachers, specially regarding the higher education.

Evaluation is a component which plays an important role in the whole learning process, and it is crucial that a student is frequently tested and evaluated while being able to obtain quick, detailed and constructive feedback.

In higher education, teachers cannot manage such a level of assessment because of the large number of students.

The motivation of this work is mainly due to the fact that this area of research is still unexplored and there are not many related pieces of work with the same specific goal. As a fond programming student, it is interesting to provide the rest of the student community an important tool which enables them to benefit from an efficient self-learning experience.

1.2 Problem Statement

Last year, in 2010, a Computer-Based Assessment system (CBA) was developed and applied to the teaching of programming, which integrates with the e-learning platform Moodle. This system is based on the programming contest jury system DOMjudge and performs the basic tasks of such a system. However, it lacks some important features,

such as the support for plagiarism detection, static analysis of code quality and feedback customization.

To be fully functional in the teaching activities of programming courses, this platform would greatly benefit if extended with advanced modules, higher customization possibilities and improved ease of use.

1.3 Goals

The main goal of this dissertation is the development of a set of advanced modules for a platform developed in the academic year 2009/10 for the automatic evaluation of course units concerning Programming in the Faculty of Engineering of the University of Porto (FEUP).

Among these advanced modules it is possible to outline the custom feedback, the development of more complex validators, essential for some programming problems, the automatic correction of projects with multiple source files, the static code analysis, the plagiarism detection and the development of more intuitive tools supporting the assessment creation.

To achieve this, it will be necessary to accomplish the following objectives:

- Conceive a state of art report concerning the e-learning technology, E-assessment and software testing, focusing on the most advanced functionalities and specificities of the teaching of programming;
- Define the set of the new modules and its basic functionality;
- Study the Computer-Based Assessment system already developed from the module integration point of view;
- Analyse and specify the requirements of the advanced modules of the Computer-Based Assessment system;
- Develop the specified modules and integrate them with the existing system;
- Test and evaluate the modules;
- Prepare a set of test cases;
- Conduct a series of experiments and a survey for validation of the Computer-Based Assessment platform.

1.4 Expected Results

With this work, it is expected to obtain a perfectly working tool with fully customizable advanced modules, ready to be integrated in Moodle or other systems with the same purpose.

It is hoped that this tool will be embraced by the students and will eventually take a consistent and important part on their study methods, complementing the teachers.

Ideally, the platform will be successful enough to start working the next school year with the course units subject to test during the dissertation. The platform is aimed to be sufficiently generic for being used in any course unit of the Faculty of Engineering of University of Porto which involves programming.

1.5 Related Work

CBA systems are used in a number of universities around the world. Since this thesis will be a contributing work to a platform already developed, the main related work is the underlying platform [[Pac10](#)].

Six other CBA systems were analysed in order to explore their functionalities and implementation methods, since some already feature a subset of the modules required (static analysis of code quality and plagiarism detection): CourseMarker [[HGST05](#)], BOSS [[JGB05](#)], Submit! [[PRS⁺03](#)], XLX [[HLVW02](#)], RoboProf [[Dal99](#)] and GAME [[BGNM04](#), [MBG07](#), [MBG09](#)].

Two automated judge systems which may serve as education platforms were taken into account: Mooshak [[LS03](#)] and DOMjudge [[EKW11a](#)], being the latter the basis of the developed platform.

Concerning more specifically the plagiarism detection, some existing tools were studied in order to analyse the performance of different algorithms: SIM [[GT99](#)], MOSS [[SWA03](#)], YAP3 [[Wis96](#)] and JPlag [[PMP00](#)].

1.6 Methodology

A case study will be done by the end of the first semester to test the intuitiveness of the platform in an assignment of the course unit Programming Fundamentals. A survey regarding its students and teachers will follow in order to obtain information about the ease of use of the platform.

Additionally, concerning the state of the art, an evaluation of similar CBA platforms will take place, focusing on the modules to be implemented.

When the modules are integrated in the platform and subsequently tested, a working prototype will be established as a validation method in both the case studies proposed.

During this validation phase, bugs that might eventually appear will be corrected and the interface will be improved.

1.7 Document Structure

This document is structured in seven chapters, being this first one the introduction.

The second chapter comprises an analysis to the base CBA system developed, containing a summary of the features implemented and the limitations of the implementation. Moreover, the system is compared at a functionality level to eight other CBA systems.

Chapter 3 approaches the weaknesses of the system and a study of other CBA systems which implement some of the missing features, as well as external tools which can serve the same functionality. The viability of implementation of each solution is discussed.

The fourth chapter consists of the specification of the advanced modules for the base CBA system and a discussion of the technologies being used in the current implementation.

Chapter 5 contains every detail about the advanced modules development and integration in the CBA system including the explanation of a case study.

The evaluation of the results achieved by this project is presented in chapter 6. The implemented features are listed, alongside with the tests conducted to each module.

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

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

Chapter 2

The Base CBA System

The base CBA system was the subject of the Master's Thesis of Pedro Pacheco (2010), a student of the Master in Informatics and Computing Engineering of the Faculty of Engineering of the University of Porto. It was developed based on the DOMjudge platform, after studying and identifying the characteristics of the following CBA systems: Course-Marker, BOSS, XLX, Mooshak, RoboProf, Submit!, GAME and DOMjudge.

Upon analysing the hypothesis of creating a new system from scratch or reusing an existing system, he concluded that the existing systems already implement a significant amount of high priority features and a few of the medium and low priority ones [[Pac10](#)], therefore considering to be advantageous to reuse the aforementioned system.

This chapter provides an overview of the platform developed during the second semester of the academic year of 2009/10, outlining its strengths and weaknesses, as well as a brief comparison with other CBA systems.

2.1 Features Implemented

The features to implement and their priority were based on a survey performed next to the Department of Informatics Engineering (DEI) teaching staff [[Pac10](#)]. This section summarizes these features, while detailing whether a particular objective was fulfilled. Further details can be found in the Results chapter of Pedro Pacheco's thesis [[Pac10](#)].

2.1.1 Support for Assessment Creation

The CBA system successfully implements the three types of assessments required: exams, tests and exercises.

Regarding the assessment configuration, the only objective which was not implemented was configuring the duration of each individual question. It is possible to change

the start time, deadline, accepted programming languages, tolerance time, duration and maximum number of submissions.

2.1.2 Support for Test Cases Definition

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 behaviour [Pac10]. It does not, however, support unit testing nor regular expressions.

The only parameter for test-case configuration implemented was the weight for assessment grade.

2.1.3 Support for Elaboration and Submission of Solutions

All the features associated to the support and elaboration of submissions were implemented.

2.1.4 Feedback System

The current feedback system provides three types of features: toggling between immediate and non-immediate feedback, personalization of feedback messages per test case and regulation of the level of feedback.

The level of feedback can be currently set to *Minimalist*, *Moderated* and *Detailed*. 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 [Pac10].

2.1.5 Submissions Management

Concerning submissions management, only the consulting of the history of submissions and grades of students was implemented.

2.1.6 Extra Features

None of the features classified as extra were implemented in the system.

2.2 Comparison with Other CBA Systems

This section presents a comparison of all the features of eight state-of-the-art CBA systems [Pac10] and the prototype developed.

It is possible to conclude, according to the table 2.1, that all the systems support the three types of assessment: exams, tests and exercises.

Table 2.1: Comparison of CBA systems features: types of supported assessments.

Feature	CBA System								
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge	Base CBA System
Exams	x	x	x	x	x	x	x	x	x
Tests	x	x	x	x	x	x	x	x	x
Exercises	x	x	x	x	x	x	x	x	x

Taking the parameters for assessment configuration into consideration, no system allows setting a duration for each individual question. However, the base CBA system implements a greater number of features than every other system examined, as it can be seen in the table 2.2.

Table 2.2: Comparison of CBA systems features: parameters for assessment configuration.

Feature	CBA System								
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge	Base CBA System
Start time	x	x	x	x	x	x	x	x	x
Deadline	x	x	x	x	x	x	x	x	x
Accepted programming languages				x				x	x
Tolerance time									x
Duration									x
Maximum number of submissions	x								x
Duration of each individual question									

The Base CBA System

When comparing the methodologies for defining test-cases (table 2.3), the prototype only implements the use of text files, in similarity to DOMjudge. When a submission run is given the input test data, the resulting output is compared to the reference output data. If they match exactly, the problem is judged to be correct [EKW10]. No studied CBA system implements test-case definition by the means of regular expressions.

Table 2.3: Comparison of CBA systems features: methodologies for test-cases definition.

Feature	CBA System								Base CBA System
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge	
Input-output tests	x	x		x	x	x	x	x	x
Unit tests		x	x						
Regular expressions tests									

Regarding the parameters for test-case configuration, the base CBA system only features the weight for assessment grade, as shown in table 2.4. It has lost its capacity inherited from DOMjudge to adjust the maximum runtime for each test-case. Four systems implement static analysis of code quality, which is one of the main limitations of the prototype.

Table 2.4: Comparison of CBA systems features: parameters for test-case configuration.

Feature	CBA System								Base CBA System
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge	
Weight for assessment grade		x	x	x		x			x
Static analysis of code quality	x	x				x	x		
Maximum runtime	x			x				x	
Maximum memory				x					
Distinguish students grades based on solving time									
Identification and penalization of common mistakes									

The Base CBA System

All the systems provide a mechanism for uploading solutions (table 2.5). However, the skeleton solution mechanism proved to be a bit unpopular, being only supported by the CourseMarker system [Pac10].

Table 2.5: Comparison of CBA systems features: support for elaboration and submission of solutions.

Feature	CBA System								Base CBA System
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge	
Mechanism for uploading solutions	x	x	x	x	x	x	x	x	x
Skeleton mechanism	x								x

The base CBA system differentiates itself well from the others through its feedback system (table 2.6). No other system allows neither the toggling of immediate / non-immediate feedback, regulation of the level of feedback nor personalized feedback messages.

Table 2.6: Comparison of CBA systems features: feedback system.

Feature	CBA System								Base CBA System
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge	
Toggle immediate/non-immediate feedback								x	x
Regulation of the level of feedback	x								x
Personalized feedback messages									x

Concerning the management of submissions, the prototype falls behind in this category, implementing only one out of eight features. Although, the other CBA systems also possess a narrow range of tools for this purpose, as outlined in the table 2.7.

The Base CBA System

Table 2.7: Comparison of CBA systems features: management of submissions.

Feature	CBA System							
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge
Consult grades statistics	x					x		
Manual alteration of assessment results		x				x		
Student history	x	x	x	x		x	x	x
Student notification	x	x						
Sending of e-mails								
Alteration of submitted code		x						
Exporting of assessment results								
Built-in environment for manual testing of submissions								

The current system does not implement any feature classified as extra, even though plagiarism detection was labelled as high priority. In conformity with the maximum runtime feature analysed in the table 2.4, the prototype lost two functionalities native to DOMjudge. Every other system implements at most two of the four features presented in the table 2.8.

Table 2.8: Comparison of CBA systems features: extras.

Feature	CBA System							
	CourseMarker	BOSS	XLX	Mooshak	RoboProf	Submit!	GAME	DOMjudge
Plagiarism detection	x	x			x		x	
Q&A system	x			x				x
Exercise database								
Submissions ranking				x				x

It is possible to conclude that the base CBA system does not limit itself to replicate the functionalities of DOMjudge, albeit losing some of them, but also extends its features,

providing two innovative mechanisms not found in other CBA systems [Pac10]. The number of features implemented makes the prototype a good starting point for developing a more complex and powerful system, mature enough to be used in any programming course in the Faculty of Engineering of the University of Porto.

2.3 System Architecture

The system is divided in three main components, as depicted in the high-level UML diagram of figure 2.1:

- Moodle Server;
- Automatic Assessment Server;
- Judgehost Server.

The Automatic Assessment Server is constituted by two essential parts, the DOMserver and the front-end which is used by Moodle for interaction. The DOMserver is the central element of the DOMjudge architecture, containing the database where all contest, problem, submission and user information is stored. The front-end is meant to be added to this server to provide additional functionality by interacting directly with the DOMjudge database and enabling the use of theoretical advanced modules such as the plagiarism detection and static code analysis tools.

The actual marking component of the system is the Judgehost server, whose task is polling the central DOMjudge server database for unjudged submissions, running them and updating the database entries with the compilation and execution results. There can be multiple Judgehosts linked to the DOMserver, allowing more submissions to be graded in parallel.

Lastly, the Moodle Server contains a regular Moodle installation with the plugin "Programming Assessment". This plugin is the end-user interface with the CBA system. A teacher can create, configure, update and delete programming exercises and students are able to submit their solutions to the system. The plugin intercommunicates with the Automatic Assessment Server solely via web services provided by the front-end, as previously mentioned.

2.4 Limitations of the Current Implementation

Despite having a complete system specification and a functional prototype that implements a significant amount of features [Pac10], the CBA system as it currently is could be further improved into a more mature solution to serve as a learning tool. It still has room for improvement in some areas and there are a few important features missing:

The Base CBA System

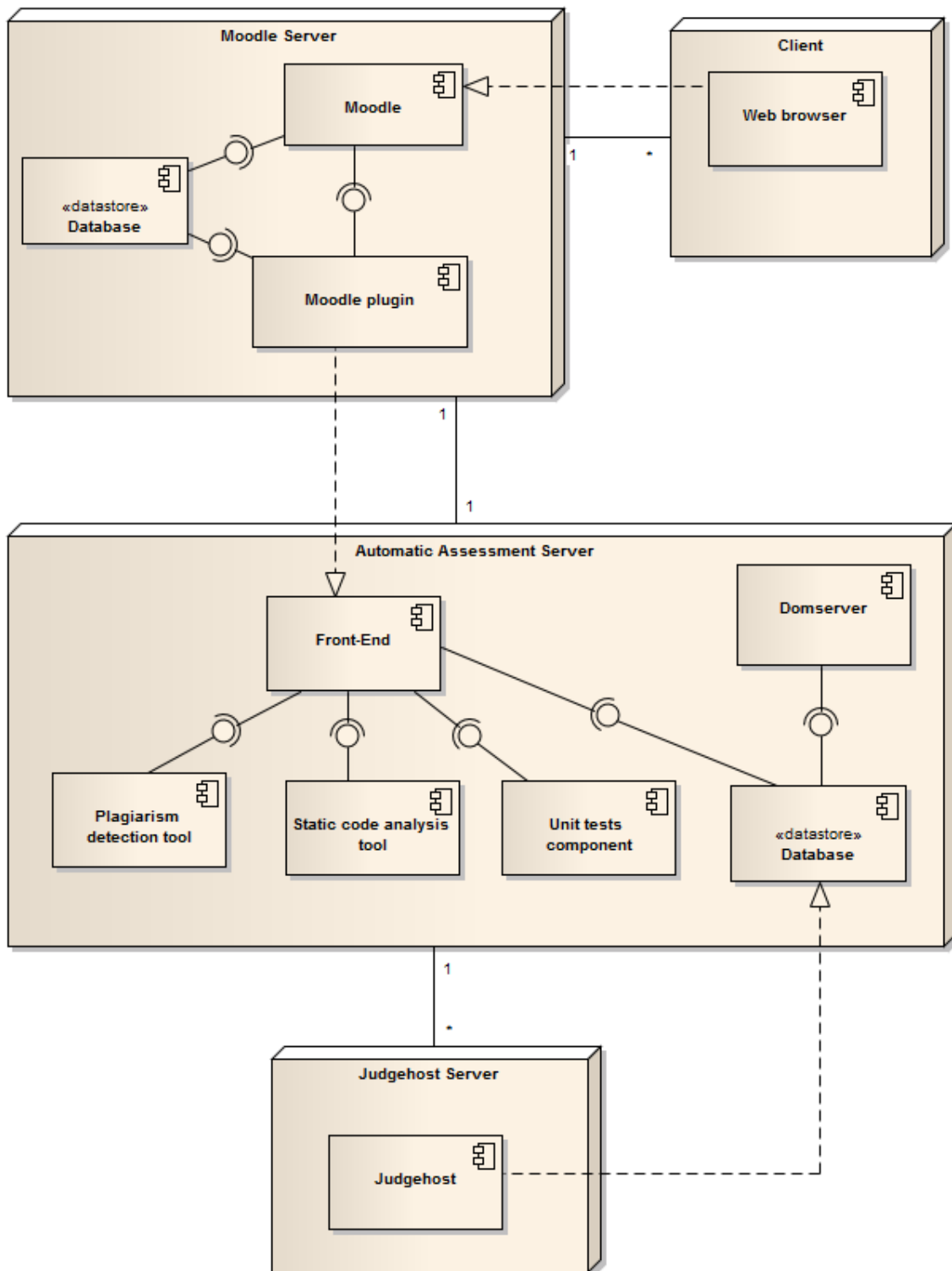


Figure 2.1: Base CBA system architecture (Source: [Pac10] p.71)

- **Static analysis of code quality:** while not considered a high priority in the development of the base CBA system, it is one of the goals of this thesis. The Automatic

Assessment Server (refer to figure 2.1) was designed keeping in mind the possibility of integrating an external static code analysis tool, although it was not implemented;

- **Plagiarism detection:** one of the highest priority goals which was not accomplished. This feature can benefit from the current architecture in a similar fashion as the previous described one;
- **Feedback level regulation:** although the system presents, according to table 2.6, the most advanced feedback system, there is still room for improvement. The detail level, for instance, can be further customizable to reflect details generated by the static code analysis tool, instead of being limited to three predetermined levels;
- **Support for assessment creation:** the tools currently used to create assessments should be more polished and intuitive, enabling teachers to easily set up assignments;
- **Use of additional DOMjudge features:** due to the programming contests nature that characterizes DOMjudge, it would be interesting to adapt more of its features, namely the team ranking, which can be used to set up some small competitions to foster students' motivation. Likewise, it would be profitable to take advantage of other features, such as the use of *validators*, which can be extremely useful for problems that do not fit within the standard scheme of fixed input and/or output [Pac10];
- **Interactivity:** can enhance the overall user experience. For instance, the interface for submitting solutions and consulting their results can be improved by using more JavaScript and AJAX [Pac10].

2.5 Summary

In the present chapter the major strengths and weaknesses of the base CBA system were identified.

This study provided a deeper insight into the base CBA system architecture by breaking it down to components (Moodle Server, Automatic Assessment Server and Judgehost Server) and describing them in detail. An analysis of each implemented feature and a comparison with the features implemented in other CBA systems: CourseMarker, BOSS, XLX, Mooshak, Roboprof, Submit!, GAME and DOMjudge, was encompassed as well.

Pedro Pacheco implemented most of the basic features with highest importance inherent to any system of this kind, establishing a solid ground level for the work described in this dissertation.

Nonetheless, the current implementation has some limitations, which were exposed and a brief tutorial was given on how they can be overcome. These limitations and the

The Base CBA System

future work delineated by Pacheco set the goal meant to be achieved during this thesis: the implementation of advanced modules for CBA systems.

The advanced modules outlined can be listed as:

- Static analysis of code quality;
- Plagiarism detection;
- Feedback level regulation;
- Support for assessment creation;
- Automatic correction of projects with multiple source files.

Chapter 3

State of the Art

This chapter approaches the functionalities deemed important by a past survey directed to the DEI teaching staff [[Pac10](#)] which the base CBA system currently does not implement and are labelled as *advanced* in this type of systems.

Some of the features the prototype is lacking are already implemented in a number of CBA systems studied in the chapter [2](#) or available as external tools. Other features, namely the feedback level configuration, the support for assessment creation and the automatic correction of projects with multiple source files are more technology dependent and are not covered at the same level as the static analysis of code quality and plagiarism detection.

3.1 Static Analysis of Code Quality

The process of statically analysing the quality of code can help to give a finer level of detail to grading allowing students more insight into how to improve their programming skills [[MY99](#)].

In order to evaluate a student's performance in programming, there are some metrics to take into consideration: programming skill, complexity, programming style and programming efficiency [[MY99](#)]. These metrics can only be calculated by a thorough examination of the code, or automatically, with a static analysis of the code.

This analysis can be as powerful as to be used to fully grade a program. Such feat was achieved by Wang's semantic similarity-based grading of student programs. The automatic grading of a student program is achieved by calculating semantic similarities between the student program and each model program. The higher the largest semantic similarity is, the higher the mark [[WSWM07](#)].

This section aims to present how other CBA systems implement their own code analysis tools as well as providing a comprehensive study on the most widely used software metrics. Lincke, Lundberg and Löwe [LLL08] present a detailed comparative analysis between a variety of different software metric tools and plugins at disposal. Although a high number of tools of this type exist, they will not be covered since it is not intended to integrate external software for this purpose.

3.1.1 Approaches Taken by Other CBA Systems

CourseMarker

CourseMarker has a set of marking tools at its disposal: Typographic tool, Dynamic tool, Feature tool, Flowchart tool, Object-oriented tool and CircuitSim tool [HGST05].

The available static marking tools are comprised of tools that assess the typographic and structure correctness, the algorithmic complexity and other measures related to the exercise features of the program. Static metric tools analyse the student's source code for typography, complexity, program structure, and features specific to the exercise.

Some of the typographic metrics used in the system are listed below [Tsi02]:

- Percentage of blank lines;
- Average characters per line;
- Percentage of average white space per line;
- Average identifier length;
- Percentage of names with good length;
- Percentage of comment lines;
- Percentage of characters in comments.

When considering the program complexity, the following metrics are used for comparison with the model solution [Tsi02]:

- Methods of types and data declarations;
- Reserved words, include files and literals;
- Assignment statements and complexity of expressions;
- Library functions and function calls;
- Operators, conditional statements and loops (including their depth);

- Maximum depth of braces, square brackets and round brackets.

Taking into account the program structure, the metrics grade the occurrence and absence of possible problematic aspects of the source code. The most notable metrics of this type are [Tsi02]:

- Variable assigned / defined / declared but never used;
- Variable undeclared / re-declared previously as a different kind of symbol;
- Variable used before set;
- Value computed / returned by a function never (or sometimes) used;
- Statement not reached / with no effect;
- Assignment of integer from pointer lacks a cast;
- Float or double assigned to integer data type;
- Comparison is always a set result due to limited range of data type;
- Data definition lacks type or storage class.

The feature metrics are used in a per exercise basis, to ensure the submitted solution uses a particular concept. Some examples of these features are as follows [Tsi02]:

- Numeric denotations which should have been set as constants;
- Language specific features (use of switch statements, overloading, *inlining*, ...);
- Non-use of preferred denotations (" ≤ 99 " instead of " < 100 ", for example).

To sum up, the CourseMarker CBA system features rich static analysis tools, concerning all aspects mentioned: structure, variables, metrics and comments. Nevertheless, some of the metrics presented are language-specific and cannot be applied to every programming language.

BOSS

BOSS provides a set of simple program metrics, such as number of comments, percentage of methods declared abstract, etc., which can support the marker in assessing the subjective attributes of a program. The software is extensible and inclusion of other metrics is straightforward [JGB05].

Due to the lack of documentation on the program metrics used, the exact characteristics of the static analysis tool of BOSS system are unknown.

Submit!

Submit! has incorporated programming style checking scripts which the students can freely use, based on what is identified in the University of Macquarie as their *ideal* programming style. The guidelines were chosen taking into account the most common problems in student code and the ease of implementing the automated tests, and can be enumerated as follows [PRS⁺03]:

- All program lines should be less than 78 characters;
- Should use space instead of tabs;
- There must be space left between a closing parenthesis ')' and an opening curly brace '{';
- Do not have code on the same line following a curly brace '{';
- Do not use a whole block in a single line (an opening curly brace '{', followed by code, followed by closing curly brace '}');
- There should be space between the indicator for comments (usually '//' or '/*') and the actual comments.

Concerning the static analysis of code quality, Submit!'s approach is merely at a style level, allowing students to produce tidy, understandable and maintainable code.

GAME

The GAME CBA system, in its initial state, used to calculate a mark for source code structure, evaluating the following parameters [BGNM04]:

1. The number of block comments compared to the number of functions;
2. The valid declaration of variables (both global and local);
3. The number of "magic numbers" (number denotations which should have been set as constants).

However, this system has been extended (GAME-2, in 2007 [MBG07] and GAME-2+, in 2009 [MBG09]) to adopt fuzzy logic for analysing algorithms and meaningful comments.

The fuzzy logic underlying the analysis for the algorithm accuracy is the calculation of a ratio of some parameters of a student's algorithm to the instructor's algorithm:

- Number of iterations;

- Number of conditions;
- Number of assignments;
- Number of inline comments;
- Number of arrays.

Considering the meaningful comments, the grade is automatically zero if more than one fourth of words present in a comment are member of the set: $\{;, \{, \}, [,], =, :=, ==, !=, \text{if}, \text{else}, \text{import}, \text{while}, \text{do}, \text{switch}, \text{for}, \text{white space}\}$ [MBG09]. The number of words (NOW) in each comment is calculated, as well as the number of noun and preposition articles ("NumberOfNounArticles") and the number of conjunction words present ("NumberOfConjunctionWords"). A fuzzy membership function is used to calculate, for each variable, a value from 0 to 1 [MBG09]. It is then possible to obtain the "MeaningfulProportion", given by the equation 3.1, used to give partial marks on meaningful comments.

$$\text{MeaningfulProportion} = \left(\frac{\text{NumberOfNounArticles}}{\text{NOW}} \right), \left(\frac{\text{NumberOfConjunctionWords}}{\text{NOW}} \right). \quad (3.1)$$

3.1.2 Metrics

Metrics are of two types: raw and computed. Raw metrics are simple counts of things like lines of code and inheritance depth. Computed metrics take one or more raw metrics and combine them to form another measurement [Wig02].

Whereas the section 3.1.1 covers a wide array of these metrics, specially raw ones, this section details a few computed metrics and the extent to which they can be used.

McCabe's Cyclomatic Complexity

The McCabe cyclomatic complexity metric is one of the more widely-accepted software metrics for measuring the complexity of a program and it is intended to be independent of language and language format [MY99, Gmb10b].

This approach measures and controls the number of paths through a program. The cyclomatic number $v(G)$ of a graph G with n vertexes, e edges, and p connected components is given by the equation 3.2 [McC76].

$$v(G) = e - n + p. \quad (3.2)$$

The figure 3.1 illustrates a control flow graph of a simple program. In this case, the program begins executing at the red node, enters a loop and then executes a conditional

statement, ending in the blue node. In order to apply the cyclomatic number equation, the graph needs to be strongly connected (having a path joining any pair of arbitrary distinct vertexes) [McC76]. In this example, $e = 10$, $n = 8$ and $p = 1$. Therefore, $v(G) = 10 - 8 + 1 = 3$.

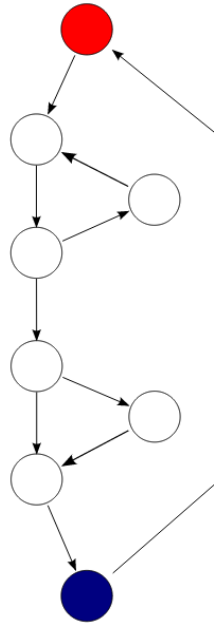


Figure 3.1: Sample control flow graph (Source: [Wik11])

Empirical evidence suggests that 10 is a reasonable upper limit for cyclomatic complexity per method. When the complexity exceeded 10, programmers had to either recognize and modularize subfunctions or redo the software [McC76].

Halstead Complexity Measures

Halstead's measures, which are best known as *software science*, are based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand. These measures were introduced in 1977 and have been used and experimented with extensively since that time. They are one of the oldest measures of program complexity. [Gmb10a].

The major components of software science are [KST⁺86]:

- η_1 , the number of unique operators;
- η_2 , the number of unique operands;
- N_1 , the total number of operators;
- N_2 , the total number of operands.

Halstead derived a number of measures from these components using a set of formulas for their calculation. The Testwell CMT++ and CMTJava (Complexity Measures Tool) documentation provides extensive information on the implementation of the Halstead metrics, as well as their most typical values [Gmb10a]:

The **program length** (N) is the sum of the total number of operators and operands in the program (3.3).

$$N = N_1 + N_2. \quad (3.3)$$

The **vocabulary size** (η) is the sum of the number of unique operators and operands, as shown in the equation 3.4.

$$\eta = \eta_1 + \eta_2. \quad (3.4)$$

Halstead's **volume** (V) describes the size of the implementation of an algorithm. The computation of V is based on the number of operations performed and operands handled in the algorithm. It is given by the equation 3.5.

$$V = (N_1 + N_2) * \log_2(\eta_1 + \eta_2). \quad (3.5)$$

The **difficulty level** or error proneness (D) of the program is proportional to the number of unique operators in the program (3.6).

$$D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2}. \quad (3.6)$$

The **program level** (L) is the inverse of the difficulty level of the program (3.7).

$$L = \frac{1}{D}. \quad (3.7)$$

The most extensively studied measure is an estimate of the **effort**, E , required to implement a program [KST⁺86]. Its equation can be expressed through the proportionality of the volume and difficulty level of a program (3.8).

$$E = V * D. \quad (3.8)$$

The **time to implement** or understand a program (T) is proportional to the effort. The equation 3.9 gives the approximated time to implement a program in seconds, whose value was calculated through empirical experiments by Halstead.

$$T = \frac{E}{18}. \quad (3.9)$$

Halstead's **delivered bugs** (B) is an estimate for the number of errors in the implementation and it is given by the equation 3.10. Delivered bugs in a file should be less than 2.

$$B = \frac{E^{\frac{2}{3}}}{3000}. \quad (3.10)$$

3.1.3 Summary

The wide array of metrics covered in this section fall into the categories of complexity, style and overall good practices of programming.

Four of the CBA systems studied in chapter 2 implement a form of static analysis of code quality: CourseMarker, BOSS, Submit! and GAME. CourseMarker contains the most complex solution, since its static metrics tool marks typography, complexity, program structure, and features specific to the exercise. BOSS and Submit! take mostly into account the programming style of a student, whereas GAME features a different approach, consisting of fuzzy logic to grade algorithm complexity and meaningful comments.

Concerning individual software metrics, special attention was devoted to two different complexity metrics: McCabe cyclomatic complexity and Halstead complexity measures.

The most important metrics are the ones which directly help a student to either overcome or prevent problems in his code. For instance, they may aid the student to find useless or repeated code which can be refactored to methods. The computed McCabe cyclomatic complexity number and the Halstead measures give a detailed insight about one's code complexity. In fact, it has been proven that both measures are highly correlated [KST⁺86].

The static analysis of the code quality can play an important role in the evaluation of assessments as well, which enforces the requirement that all the metrics used need to be robust and resilient to cheating. A good example to this strategy is the fuzzy logic used in the GAME-2+ system [MBG09] to decide whether the comments are meaningful.

There are many more different metrics which have not been fully covered, but most are extracted without a need for thorough code parsing. Redish and Smyth's solution approach for a program style analysis [RS86] features the gathering of a high number of simple metrics, and comparing a model solution to the submitted solution. Each of these metrics is given a weight and a significance, i.e., whether having a higher metric score comparing to the model solution should be valued.

Most approaches to static code analysis in education are specific to a single source file. However, there are special metrics that can be applied when it comes to bigger projects, such as coupling metrics, which indicates the degree of independence between various modules in the program [MY99]. It is important that such a tool provides at least an

analysis for every source file of a project so that students spend less time trying to find faulty code.

3.2 Plagiarism Detection

Strategies which are based on society's change of attitude against copying or theft without any doubt are the most significant means to fight against plagiarism; however, the implementation of these methods is a challenge for society as a whole. Education institutions need to focus on techniques to detect similarities in code [LGG07].

There are two principal program comparison techniques, through the calculus of attribute counts (software metrics) and by comparing programs according to their structure [JL99].

Software metrics have been developed for the purpose of measuring software quality and guiding the software development process. Nonetheless, they can be also used in the detection of plagiarism in student programming assessments [Wha90]. In order to be useful in similarity detection, a software metric must present four important properties [Wha90]:

1. Significance of representation;
2. Robustness;
3. Ease of comparison;
4. Language generality.

These metrics can be sorted as fixed-size metrics and variable-size metrics, although they have been proven to be quite inadequate [Wha90], which supports the statement that software complexity measures must be critically evaluated to determine the ways in which they can best be used [KST⁺86].

In order to compare the structure of programs, a structure profile must be extracted in such a way that redundant simple statements are discarded. The tokens are then subject to a profile matching algorithm, although it might differ from language to language. Plague author Geoff Whale claims that the success of the structure profile is due primarily to its retention of detail relevant to the application at hand and its exclusion of all incidental material [Wha90].

Possible Solutions

According to the table 2.8, there are four CBA systems which implement a plagiarism detection method: CourseMarker, BOSS, RoboProf and GAME. This section comprises a study on the approach taken by each system.

There is a large number of external tools [LC03] that could be explored. To narrow down the choice range, it was opted to study some of the free tools presented in Chudá's paper which implement three different types of plagiarism detection algorithms [CK10]: SIM, MOSS, YAP3 and JPlag.

3.2.1 Approaches Taken by Other CBA Systems

CourseMarker

The CBA system CourseMarker features a plagiarism detection tool that after scanning all of the student programs and performing a variety of tests, will report any similarity between submitted work and inform the students' tutors. However, the tools used by this system are less sophisticated and thorough than other externally available tools recommended to assist on plagiarism detection [HGST05].

The University of Nottingham has promoted weekly exercises alongside larger exercises (with reports) and online exams. Students are informed that they can assist each other when working on a solution for any of the weekly exercises, while the larger exercises must be a personal work. While these types of assignments can motivate students to copy, this problem is counteracted by producing, at regular intervals, "similarity lists" of students generated by using external plagiarism software. These lists are then published on the web and dishonest students are encouraged to confess. If two students appear on several "similarity lists", their work can be cross-checked to establish whether they are regularly (over) collaborating [HGST05].

The authors claim that displaying these lists publicly has greatly helped reducing the amount of plagiarism detected the more frequently the lists appear.

BOSS

At the University of Warwick, a plagiarism detection software, known as SHERLOCK, was developed concurrently with BOSS and, until 2002, was a separate tool [JGB05].

When creating the SHERLOCK system, Joy and Luck identified the following requirements [JL99]:

- The program comparison algorithm must be reliable - ideally a structure comparison method;
- The program comparison algorithm must be simple to change for a new language;
- The interface must be efficient (preferably with graphical output) to enable the rapid isolation of potential instances of plagiarism;
- The output must be clear enough to present as evidence to the students and to a third party.

The approach adopted as a structure comparison method was labelled as *incremental comparison*, in which a pair of programs is compared five times [JL99]:

1. In their original form;
2. With the maximum amount of white space removed;
3. With all comments removed;
4. With the maximum amount of white space removed and with all comments removed;
5. Translated to a file of *tokens*.

These *tokens* are values, such as *name*, *operator*, *begin*, *loop-statement*, which is appropriate to the language in use. If a pair of programs contains similarities, then it is likely that it will be shown by one or more of these comparisons [JL99].

SHERLOCK compares two programs by traversing them, while looking for sequences of lines of maximum length common to both, where the sequence might not be contiguous (taking extra or deleted lines interrupting the sequence into account). A record file is stored containing information about the location of similar lines and their size as a percentage of the length of each program.

A neural-network program is then run which reads all the record files and generates an image which illustrates the similarities between the programs listed in the record file [JL99].

The image generated is a graph, whose nodes are the programs suspected of fraud and the length of the edge determines the level of similarity between two programs. By analysing this image, it is possible to quickly arrive at a preliminary judgement as to which programs are worth a detailed examination.

This system was subject to two tests, being the first an attempt to manually deceive SHERLOCK. No postgraduate student was able to fool the system without making substantial changes. The second test was a comparison with the Plague system, developed by Whale [Wha90]. The results proved that both systems are capable of achieving a similar level of performance [JL99].

Thanks to this system, the plagiarism at the University of Warwick has been decreased from 6 to 1 percent.

RoboProf

The plagiarism detection system works by adding a watermark to a student's program at submission time. If another student gets an electronic copy of this program (complete with the watermark), and then submits it, RoboProf will recognize it as a copy [DH05].

This watermark is a binary number formed by information on the exercise, the student and a checksum, which is imprinted onto the source code by the means of a local Java applet used for the submission process. To avoid detection of this strategy, the watermark is written using the tab and space characters as binary digits at the end of the main method signature. White space is undetectable by most standard editors and the watermark is unlikely to be changed by normal editing of the file.

The detection system used in RoboProf differs from most systems since it does not compare pairs of programs with each other in order to find potential similarities.

When comparing the watermark technique over pair-wise comparison techniques, it is possible to outline a few advantages [DH05]:

- It can detect plagiarism in very short programs that are typical of introductory programming exercises. With short programs, it is likely that students may come up with similar solutions by chance. Pair-wise detection systems cannot distinguish these chance similarities from cases of plagiarism;
- It distinguishes between the supplier of the code and the recipient of the code;
- No manual intervention is required;
- Plagiarism is detected as soon as it occurs. This feature is specially useful when exercise solutions may be submitted at any time.

Nevertheless, there are a few drawbacks to this strategy [DH05]:

- It will only detect plagiarism if the copying student gets an electronic copy after the original program has been submitted;
- A student may disturb the watermark inadvertently. Also, the watermark could be disturbed if the program is submitted while the editor is open, thus allowing saving the version of the program without the watermark. This would defeat the detection system;
- It is easy to bypass the system if the students discover how it works.

When evaluated for detection performance, this system was compared against the MOSS online plagiarism detector, based on pair-wise detection. RoboProf identified seven dishonest students which MOSS could not. Likewise, MOSS highlighted other seven students whom RoboProf failed to detect. This system, although presenting a different solution from other plagiarism detection systems, proves that is as good at detecting plagiarism, while requiring no manual processing of its results [DH05].

GAME

GAME is based on a preliminary system [GVN02] for computer-program marking assistance and plagiarism detection (C-Marker).

The GAME system's plagiarism detection module was imported from C-Marker, is tightly integrated with the marker and is based on the examination and comparison of each program's structure using a physical and logical profile [BGNM04].

3.2.2 Algorithms

The only tools viable to implement and study are the ones that are freely distributed or available. Both Chudá's and Mozgovoy's papers [CK10, Moz06] present some of the free tools available, online or offline.

The studied plagiarism detection systems (SIM, MOSS, JPlag and YAP3) can be classified as hermetic, and subdivided according to the figure 3.2. Hermetic plagiarism systems are either universal, i.e. can process text documents of any nature, or are specially fine-tuned to detect plagiarism in source code files [Moz08].

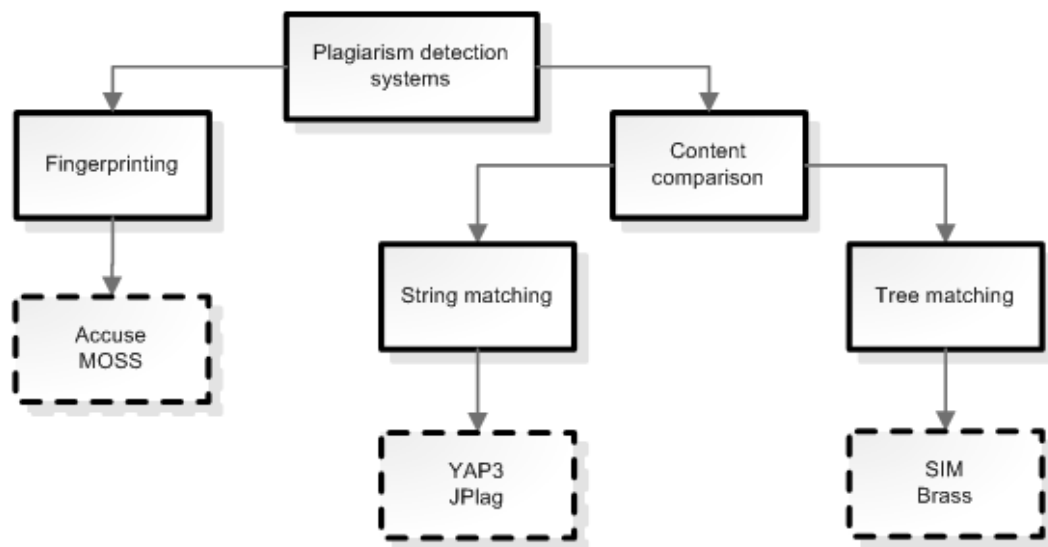


Figure 3.2: Simplified classification of plagiarism detection systems (Source: [Moz08] p.17)

Fingerprinting

The only tool studied which implements an algorithm of this kind is MOSS.

MOSS (Measure Of Software Similarity) is primarily used for detecting plagiarism in programming assignments in computer science and other engineering courses, though several text formats are supported as well [SWA03].

MOSS is being provided as an Internet service and presents the results in the form of a HTML page, although it requires a subscription [Aik10].

The algorithm used to implement the plagiarism detection is based on k -gram fingerprinting. A k -gram is a contiguous substring of length k . Divide a document into k -grams, where k is a parameter chosen by the user. A fingerprint is a subset of all hashed k -grams, containing information describing the document and its origin. If the hash function is chosen so that the probability of collisions is very small, then whenever two documents share one or more fingerprints, it is extremely likely that they share a k -gram as well [SWA03].

This algorithm is called *Winnowing* and differs from other k -gram algorithms, such as the Karp-Rabin string matching and All-to-all matching algorithms, and it is claimed to be more efficient and scalable than other fingerprinting algorithms. Moreover, since years of its implementation, a false positive has almost never been reported, providing a trusting and accurate service [SWA03].

String Matching

Early string matching based plagiarism detection systems like YAP used a *structure-metric* system for detecting suspected plagiarism in computer program texts [Wis96].

String matching systems have evolved, and currently one of the most popular file comparison methods is Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) algorithm, implemented in a variety of systems, including YAP3 and JPlag, the analysed systems of this kind [Moz06]. Its applicability is not limited to strings, it ranges as far as to comparing biological sequences [DW95].

The first tool implementing the RKR-GST algorithm was YAP3, a free offline tool [Wis05].

The basic aim of the RKR-GST algorithm is to find a "best tiling" for two given input files, i.e., the joint coverage of non-overlapping strings that includes as many tokens from both files as possible. The existence of a polynomial algorithm that provides an exact solution is still an open problem so it is necessary to make several heuristic assumptions to develop a practically applicable procedure [Moz06].

JPlag develops further the idea of the RKR-GST algorithm used in the YAP3 tool and currently it is one of the most advanced and reliable content comparison methods [MKK07].

JPlag is available as a web application in Java and requires registration. Its algorithm first converts the programs into token strings, bearing in mind that the tokens should be chosen such that they characterize the essence of a program (which is difficult to change

by a plagiarist) rather than surface aspects [PMP00]. It proceeds then to applying the RKR-GST algorithm to the token streams. The RKR-GST used is similar to YAP3 although it uses different optimization algorithms [CK10].

Overall, for clearly plagiarized programs, i.e. programs taken completely and then modified to hide the origin, JPlag's results are almost perfect, often even if the programs are less than 100 lines long [PMP00]. Also, it presents a powerful GUI and a wide array of ways of presenting results and a good support for comparing programs manually.

Tree Matching

The parse trees describe the source code content better than any fingerprinting algorithm. However, this technique requires the most processing and presents itself as the slowest.

The first project implementing an algorithm of this kind was the SIM utility [Moz08].

SIM tests lexical similarity in texts in C, Java, Pascal, Modula-2, Lisp, Miranda, and natural language. It is used to detect both potentially duplicated code fragments in large software projects, in program text, in shell scripts and in documentation as well as plagiarism in software projects, educational and otherwise [Gru09].

SIM uses a string alignment algorithm to compare two programs. The optimal alignment score between two strings is the maximum score among all alignments. This value can be computed using dynamic programming. For two strings s and t , the running time of evaluating the best alignments is $O(|s||t|)$, whereas its space requirement is only $O(\max(|s|, |t|))$, since only two rows are needed at any time [GT99].

The programs to compare are first passed through a lexical analyser to reduce the program to tokens referring to either an arithmetic or logical operation, a numeric or string constant, a comment, or an identifier, for example. The string alignment algorithm is then applied to these tokens in order to detect plagiarism.

Concluding, SIM is a hybrid approach that lies somewhere between ordinary string matching and tree comparison [Moz06].

The SIM platform is available as an offline and open-source tool and provides a Tk/Tcl graphical user interface to allow comparison of a reference file against a collection of files and display and printing of the results in the form of a bar graph. Although its execution is slow (see tree matching algorithms in figure 3.3) it has proven often enough to find the highest-scoring pairs in a group of programs, taking into account that the number of cheating incidents is usually small [GT99].

3.2.3 Summary

This chapter approaches the solutions for plagiarism detection implemented in four different CBA systems: CourseMarker, BOSS, RoboProf and GAME; in addition to three plagiarism detection algorithms: fingerprinting, string matching and tree matching.

CourseMarker does not feature an integrated tool for detecting plagiarism. An external tool is sporadically used to generate public similarity lists instead. Conversely, BOSS implements its own tool, SHERLOCK, based on the structure comparison method. RoboProf implements the watermark technique, a less orthodox and totally automated, albeit unreliable, solution. GAME integrates its own tool in a similar fashion to BOSS, although its documentation is very poor.

Four external tools were analysed which implement the aforementioned algorithms. MOSS takes the quicker, less reliable, fingerprinting approach. On the other hand, SIM requires the highest computational power using its hybrid string matching and tree matching algorithm. Both YAP3 and JPlag implement the same string matching algorithm, RKR-GST (figure 3.2).

The different plagiarism detection algorithms, comparing speed versus reliability, are classified as shown in figure 3.3 [Moz08].

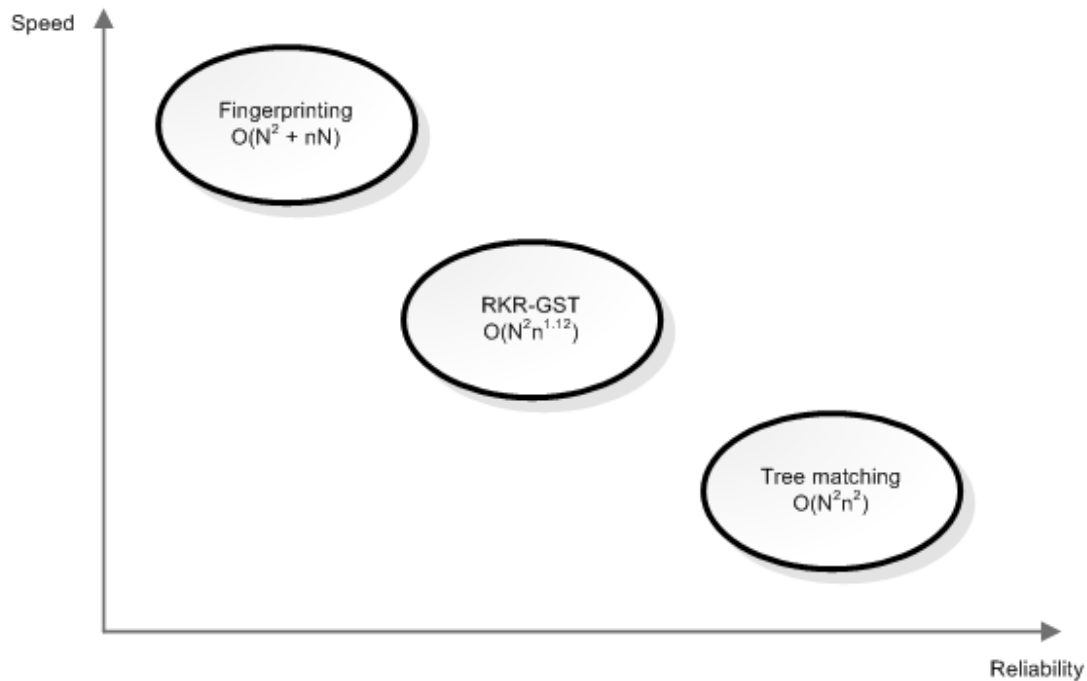


Figure 3.3: Speed and reliability of different plagiarism detection schemes (Source: [Moz08] p.38)

The performance of the different algorithms analysed convey that it is hard to combine both speed and effectiveness in a single algorithm. Nevertheless, the RKR-GST algorithm proves itself as the most successful and balanced solution for this problem. There have been some attempts at improving the speed of this algorithm through the use of suffix arrays and by applying several *ad hoc* heuristics, such as FPDS [MFW⁺05].

Moreover, Mozgovoy has identified in his thesis the most common plagiarism hiding techniques and how to overcome them by preprocessing the code [Moz08], as well as specifying the guidelines to a fast, reliable plagiarism detection system, based on the RKR-GST algorithm [MKK07].

The RKR-GST algorithm seems to be one of the main choices of implementation for a plagiarism detection system based on string matching. Several modifications to the heuristics and variations have been made by different authors, conducting experiences with satisfactory results.

RoboProf features a different and totally automated plagiarism detection scheme from the other systems studied. The watermark technique might fare better in smaller exercises than a more complex, string matching algorithm, providing it remains undetected. The authors suggest some improvements to this system [DH05], which are worth being considered, as well as a possible combination of this technique with one or multiple hermetic plagiarism systems studied.

In the end, in most cases, it is still the tutor's duty to manually check the potentially similar offending programs, since plagiarism detection tools cannot entirely replace a human decision.

State of the Art

Chapter 4

Integration of Advanced Functionalities in CBA Systems

Chapters 2 and 3 allowed to, respectively, synthesize the most important functionalities of some state-of-the-art CBA systems and further study good practices and techniques used in the implementation of two advanced modules which were considered essential, static analysis of code quality and plagiarism detection. An additional three advanced functionalities were identified: feedback level regulation, support for assessment creation and automatic correction of projects with multiple source files.

These particular modules are of great importance to any CBA system since they provide the teacher with invaluable resources for student grading, in particular by regulating the feedback and configuring the desired metrics. Implementing a plagiarism detection aims to refrain students from copying, by thinking twice whether is it worth trying to reuse other students' code. The support for assessment creation means ensuring the teacher easily understands how to create and update a programming assessment in order to further add value to the CBA platform. Lastly, these types of systems benefit from being able to grade projects with any number of source files, since as the level of programming improves the projects tend to have more files.

This chapter contains the methodology used in the course of this dissertation, the requirements, the new proposed architecture and aims to cover the different implementation possibilities of the advanced modules, based on the conclusions taken by the study of the aforementioned state-of-the-art CBA systems, external tools and algorithms. Additionally, it aims to confront some of the difficulties in the integration with Moodle 2.0 and how to take advantage of its new functionalities.

4.1 Applicability

The proposed advanced functionalities can greatly improve the current CBA system. This section details the requirements of the outlined modules, tackling possible integration issues and the system extensibility. Moreover, it also presents a discussion on how the CBA system would benefit from extending the Assignment activity and a small study of the Moodle plagiarism API, comprising two plugins abiding by this framework.

4.1.1 Requirements

Regarding the non-functional requirements of the system, it is desired that the system presents evolution qualities, namely maintainability, extensibility and scalability. The importance of this fact is due to Moodle being a constant changing open-source project with a heavy number of contributions from the community. The developed modules and the system itself should adapt to internal changes and must allow the development of gradual improvements. This can be translated by additional metrics or programming languages in the static analysis module example. Usability is both a requirement and a concern of one specific functionality itself: support for assessment creation. Performance and the response time might be an aggravating factor depending on the number of people using Moodle, so the system hardware might need to be reviewed when imbuing the new modules.

Additional requirements of the advanced modules for the CBA system include:

- For each selected metric in a submission, evaluating its result;
- For each pair of student submitted solutions, computing its similarity value;
- Upon submitting any number of files, grade the program in an identical way as if it were a single file.

Not directly concerning the advanced functionalities but nonetheless requirements for this platform in particular, are the integration with the newest version of Moodle and the capability of adding new programming languages, more specifically ones which do not fit in the simple standard of compiling and comparing the input against the expected output.

Integration with Moodle 2.0

One of the technical requirements imposed by FEUP was the use of the current major version of Moodle, 2.0. Despite being developed with this version in mind, at the development date of the previous implementation it was still unreleased, having been officially published only in the 24th of November of 2010, according to the Moodle documentation

[Dou11]. The development will begin with the latest version of Moodle 2, which is 2.0.2 (as of March, 2011).

It is reasonable to expect integration problems since a few months have passed between the Beta version and the current version, specially considering the heavy relying on the new Moodle file Application Programming Interface (API).

Adding New Programming Languages

DOMjudge, in theory, supports virtually any programming language, assuming the specific compiler is integrated in the system. Pedro Pacheco validated this fact by using Scheme, which is not supported by default, as a case study in his dissertation [Pac10], from where he concluded that it requires as little effort as installing a compiler and adding a simple compiling script to each Judgehost.

In order to further push the DOMjudge and, hence, the CBA system capabilities, implementing the SQL language for the Oracle Database Management System (DBMS) was proposed as a requirement by a few teachers responsible for the Databases course unit. Incorporating SQL in DOMjudge is a challenge since all processes - compiling, running and comparing - need to be redefined. In the same way that it is possible to use custom scripts to compile and run the code, it should also be achievable to exploit this possibility to specify the whole evaluation process for SQL.

The concept of input and output files also needed a redefinition, since there is no actual input for SQL queries. Instead, the input could specify the credentials and the database to which to connect. Moreover, since it would be difficult to use a query result as output, the solution query should be specified in the output file. The grading would be subsequently done by connecting to the database using the input file, running both the student and the teacher queries in order to compare the results.

While the notion of compiler errors and wrong answers still applies to SQL, the student should be able to read more insightful messages. Therefore, the compiler output should be passed to the student in case a submitted query could not be compiled and, in case of a wrong answer, notifying the student that the number of columns or rows of his submission did not match the solution is helpful and is too considered a requirement.

4.1.2 Assignment versus Module

Pedro Pacheco [Pac10] discussed the advantages of developing a new Module over extending the current Assignment in Moodle.

Despite the Module solution presenting solid advantages such as being less coupled to Moodle and giving more development freedom, it has many drawbacks and the current solution does not provide an answer to all of them.

The advantages outlined for Assignment were:

- Reduced implementation effort;
- Number of configuration parameters;
- Existence of an upload mechanism.

The Assignment architecture was created bearing in mind what would be essential for a teacher to grade their students. With the advent of Moodle 2, a new API for plagiarism detection was released and is tightly integrated with the Assignment activity. This API will be further discussed in section 4.1.3.

The upload mechanism is a very important factor at this point as well. In order to support the submission of multiple source files, there are two available solutions which are, either submitting a compressed file and dealing with the decompression in the server-side, which may not be safe, or using the advanced upload mechanism (figure 4.1), already implemented as an extension to the Assignment module.

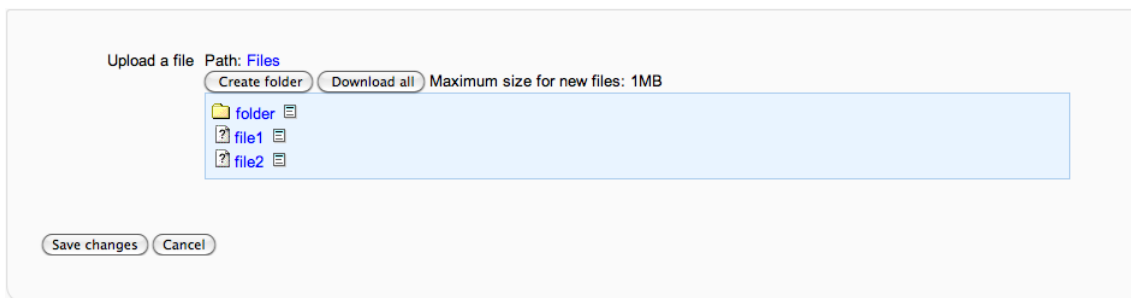


Figure 4.1: Advanced uploading of files

Concerning the assessment creation, Assignment does not present a real advantage. Nevertheless, it provides a grading screen which shows detailed information about each student's submission, as it is seen in figure 4.2. Since it is already possible to manually comment on each submission it would probably not be hard to extend this logic to support automatic feedback. Moreover, if the teacher desires, he can download all assignments at once.

Providing these functionalities are the Assignment main advantages, it would be important that the Programming Assessment module adopted these characteristics in order to turn it into a more integrated evaluation tool.

4.1.3 Plagiarism API

The new Plagiarism API is a core set of functions that all Moodle code can use to send submitted user content to plagiarism prevention systems and is currently natively supported by two different types of Assignment: *Upload a single file* and *Advanced uploading of files* [Mar11a].

Integration of Advanced Functionalities in CBA Systems

[See all course grades](#)
[Download all assignments as a zip](#)

First name : All [A](#)[B](#)[C](#)[D](#)[E](#)[F](#)[G](#)[H](#)[I](#)[J](#)[K](#)[L](#)[M](#)[N](#)[O](#)[P](#)[Q](#)[R](#)[S](#)[T](#)[U](#)[V](#)[W](#)[X](#)[Y](#)[Z](#)
 Surname : All [A](#)[B](#)[C](#)[D](#)[E](#)[F](#)[G](#)[H](#)[I](#)[J](#)[K](#)[L](#)[M](#)[N](#)[O](#)[P](#)[Q](#)[R](#)[S](#)[T](#)[U](#)[V](#)[W](#)[X](#)[Y](#)[Z](#)

	First name / Surname ↓	Grade	Comment	Last modified (Submission)	Last modified (Grade)	Status	Final grade
	student 1	76 / 100	Good	<ul style="list-style-type: none"> folder <ul style="list-style-type: none"> file3 file1 file2 Sunday, 29 May 2011, 04:03 PM	Sunday, 29 May 2011, 04:04 PM	Update	76.00
	student 10	62 / 100		<ul style="list-style-type: none"> file2 Sunday, 29 May 2011, 03:59 PM	Sunday, 29 May 2011, 04:04 PM	Update	62.00
	student 2	-		<ul style="list-style-type: none"> file1 file2 Sunday, 29 May 2011, 03:56 PM		Grade	-
	student 3	-		<ul style="list-style-type: none"> file1 Sunday, 29 May 2011, 03:56 PM		Grade	-
	student 4	99 / 100	Very good	<ul style="list-style-type: none"> file2 file3 Sunday, 29 May 2011, 03:57 PM	Sunday, 29 May 2011, 04:05 PM	Update	99.00
	student 5	-		<ul style="list-style-type: none"> folder <ul style="list-style-type: none"> file2 file3 file1 Sunday, 29 May 2011, 03:58 PM		Grade	-

Figure 4.2: Assignment grading screen

The workflow of a module taking advantage from this API can be summarized as follows [Mar11a]:

1. When Plagiarism tools are enabled, every module that allows it will have a group of settings added to allow management of sending the user content to a plagiarism service;
2. A user submits a file inside a module that a teacher / administrator has configured the tool to be used;
3. A Moodle Event is triggered which contains details about the user, module and submission they have made;
4. Event handlers in the Plagiarism plugin are triggered and process anything required;
5. Hooks for displaying information returned from the Plagiarism tools to both the user and teacher (controlled by the plugin).

This API allowed developers to create new implementations for plagiarism detection systems in order to be tightly integrated with the supported Assignment modules.

4.1.3.1 Moodle 2.0 Plugins

With the release of Moodle 2.0 a few plagiarism blocks developed for the old version were adapted to the new plagiarism API. There are currently only two up-to-date plagiarism plugins officially supported by Moodle, which are covered in this section.

Crot

Crot is available as a desktop tool and a Moodle plugin, being recommended as one of the plagiarism detection modules to support Computer Science education by Rößling *et al.* [RMC⁺10].

The algorithm for plagiarism detection was conceived bearing in mind two main ideas [BS09]:

- The algorithm should work on plain text;
- The algorithm should have good performance on large amounts of data while performing one-vs.-all checks.

To achieve this, the authors used the fingerprinting "Winnowing" algorithm used in MOSS, as described in section 3.2.2. Since this tool is meant to be used in any type of text, it allows for both local and global search. The latter is performed using the Microsoft Bing Search API.

Although the authors obtained good results when measuring similarity in essays, the fact a fingerprinting algorithm is being used also means that the system will have a harder time identifying plagiarism in short source code files. Moreover, the implemented algorithm is not resilient to source code structure change nor to noise introduced by comments in code.

Currently this plugin only works with the *Advanced uploading of files* Assignment. Notwithstanding, it supplies the teacher an array of parameters to fine-tune the algorithm and the global search.

Turnitin

The Turnitin system is one of the most tried and trusted system in use around the world concerning electronic plagiarism detection [Jon08].

Each document submitted to this system is compared against billions of Internet documents and a local base of submitted student papers and an originality report is generated to estimate the percentage of matches between that document and the Turnitin database [Jon08]. The drawback of this system is the concern about intellectual property violation, since the submitted document may contain personal details depending on how Turnitin is used.

Although the algorithm is not publicly available, it is important to note that there have been many reported attempts of deceiving Turnitin and some were well succeeded. Jones and Moore [JM10] briefly highlight some issues in trusting the similarity score produced by Turnitin.

The plagiarism detection plugin for Moodle using the Turnitin API works for both the supported Assignment types, which is a plus. Conversely, it does require a Turnitin account and an API key, which may incur in an extra cost [Mar11b]. Additionally, the global configuration process is not as out-of-the-box as it should.

4.2 Proposed Architecture

With base on the identified advanced modules, a new architecture for the CBA system was proposed and is illustrated by figure 4.3, as opposed to the old prototype architecture depicted in section 2.3. While the two are structurally the same in terms of high-level components, they differ in how the communication is set between modules.

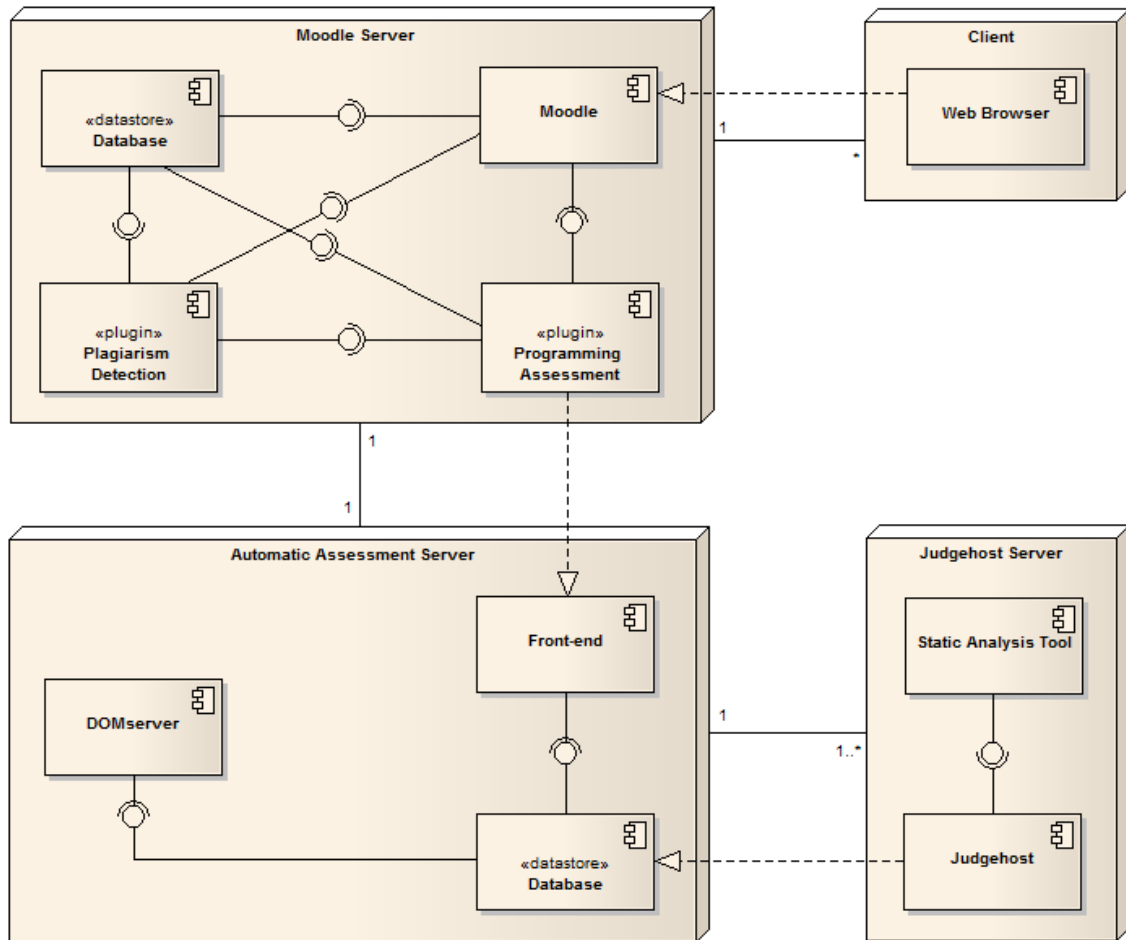


Figure 4.3: New CBA system architecture

The "Moodle plugin" instance in the previous diagram was remapped to "Programming Assessment" to reflect the actuality of the architecture. Moreover, the "Unit tests component" was removed since it does not fit on the current requirements of this work.

The plagiarism detection tool will be implemented using the Moodle plagiarism API as a plugin, thus interacting with Moodle in a similar way as the plugin Programming Assessment. When submitted, a file will be stored in the database and scheduled to be analysed by the plagiarism plugin, which in turn will fetch the needed files from the database to compute their similarity score.

Both the Moodle Server and the Automatic Assessment Server maintain their previous interaction scheme, possibly with added functionalities, depending on the functions needed to extend the CBA system. The internal logic of the latter component was simplified, keeping only the DOMserver, database and front-end.

Lastly, the static code analysis tool was transported to the Judgehost Server, since it makes more sense to be run right after (or before) the compilation step. Also, the number of these servers should be at least one, as without Judgehosts the CBA system would not be able to grade any submission. This detail was incorporated in the new system architecture proposal as well.

4.3 Advanced Module Specification

The advanced modules to be developed were clearly identified as follows:

- Static analysis of code quality;
- Plagiarism detection;
- Feedback level configuration;
- Support for assessment creation;
- Automatic correction of projects with multiple source files.

The study comprised in chapter 3 provides some important guidelines for the implementation of two of the modules enumerated, namely the static analysis and plagiarism detection. The latter modules are technology dependent and a more detailed specification needs to be conceived at implementation time.

Static Analysis of Code Quality

Although all metrics covered could be implemented, as Wigner states in his thesis, the general rule of thumb is that as few metrics as possible should be used to simplify the

interpretation of the results [Wig02]. The key to this module is gathering an appropriate number of metrics through the balance of quantity and quality.

The tool should be designed bearing in mind the possibility of using it as a standalone application (students who want to rate the quality of their code without submitting) as well as tightly integrated with the dynamic marking tool. When analysing projects, it should output a report per source file reviewed.

If used in conjunction with the marker, it is possible to compare the metrics of a model solution against submitted solutions in order to grade the code quality component of a student's submission. Also, the quality of the comments can be graded using the aforementioned fuzzy logic.

The metrics that play the most important role on such a tool are the McCabe cyclo-matic complexity and the Halstead complexity measures, as they constitute the backbone of the calculation of the complexity of a solution.

The tool should also provide a basic profiling of the program and some simple averages, such as the lines of code per function.

Finally, the style should be accounted for as well by type checking some general programming good practices, whereas providing support for language-specific features such as the *CamelCase* naming convention for the Java language.

Ideally, the teacher should be presented a list of metrics from which to choose and apply to an exercise, by individually selecting acceptable values and the weight of each metric in the final grade. The student would be able to check his grade in the static analysis component, learning about which he had done wrong. He could then use this information to his advantage to improve his programming style.

Plagiarism Detection

As covered in the previous chapter, the available methods for the implementation of a plagiarism detection tool are the integration of an external tool and the manual implementation of an algorithm, without discarding the possibility of having more than one solution.

The main requirement of this tool is to generate a list of potential plagiarists given a submission list. When it comes to implementation, the best algorithm to use would be the RKR-GST algorithm, since it provides the best results overall and there is a large amount of documentation on variations and heuristics. The watermark technique is a simple solution that can be implemented to complement the RKR-GST algorithm.

Taking integration of an external tool into consideration, YAP3 and SIM would be the better solutions providing there is no need for a subscription. Nevertheless, it is worth mentioning that JPlag is the keeper of the most reliable results, making it a possible solution as well. The main drawback of such an online tool is the need to manually upload

the students' submissions, defeating the purpose of being an automatic and independent tool. It still might be a valuable option when evaluating assessments of higher importance, namely exams.

Finally, concerning the plagiarism plugins covered in section 4.1.3, Crot presents a solid choice as well.

Feedback Level Configuration

As reviewed in the chapter 2, the base CBA system presented the most advanced feedback system when comparing to the other CBA systems studied.

However, the feedback system is currently limited, and can be improved by offering the possibility to manually configure on a per exercise basis the output to provide to the students. With the addition of the static code analysis tool, the feedback can be even further configured to display the desired metrics.

Support for Assessment Creation

Although the base CBA system already provides support for assessment creation featuring a skeleton mechanism, it lacks a proper interface for doing so. Consequently, it is an aspect that needs to be addressed when it comes to usability.

A tutor must to be able to easily set up an assignment. The current system is based in text files and could be improved at a technology level by implementing an appropriate Graphical User Interface (GUI) based on dynamic programming languages such as JavaScript and AJAX to aid in this process.

Automatic Correction of Projects with Multiple Source Files

This feature is unique to this type of systems and so far no CBA system implements anything similar. Its purpose is to accept submissions with multiple source files in a first phase and later, to automatically solve potential linking errors due to a poorly generated *makefile* or malformed or misplaced includes of other source files. In section 4.1.2 was discussed how this feature could be achieved by:

- Changing structurally the current module to extend the Assignment activity;
- Implementing a similar upload process as the *Advanced uploading of files* Assignment;
- Providing the student the option to submit compressed files.

On the understanding that automatically solving linking errors is highly technology dependent, its requirements will be further specified at implementation time.

Chapter 5

Implementation

Using the specification of the previous chapter as a guideline, a subset of the advanced modules was implemented in the existing CBA system. The vast majority of the code was written in PHP, the core language of both Moodle and DOMjudge architectures, and in Bash scripts, also used in DOMjudge and in the developed static analysis module in particular.

This chapter aims to cover the most relevant implementation details and all setbacks encountered in the course of the work produced in this dissertation. It includes a description of:

1. The architecture implemented: covering the hardware specification, software used, installation process and Moodle 2.0 integration;
2. A case study on the extensibility of the CBA system, more specifically the incorporation of the SQL language;
3. The main steps to create a contest using the DOMjudge interface as an assisting tool;
4. Every step of the development of the static analysis of code quality module and a brief description on how it can be extended;
5. The underlying changes to the communication protocol in order to adapt to the system changes;
6. The choices behind the implementation of the plagiarism detection plugin and its features.

5.1 Full System Architecture

The global system architecture was implemented exactly as proposed in section 4.2.

The complete configuration and development of the CBA system involved a few steps, being the first restoring the previous functionality of the prototype implemented in the last year. Although three distinct virtual machines were used then, it was proposed to isolate in different machines the Automatic Assessment Server and the Moodle Server to prevent hardware issues and excess of traffic. Additionally, the following steps involved specifying the hardware details and installing the software needed for establishing the CBA system from scratch. Finally, the complications brought by the Moodle version disparity were addressed.

5.1.1 Hardware Specifications

A total of four virtual machines were provided and set up by CICA (Centro de Informática Correia de Araújo), the FEUP computer centre: *domserver*, *domjudge1*, *domjudge2* and *dommoodle*, running the operative system Debian GNU/Linux 6.0 (squeeze).

The main components of the architecture were deployed in these virtual machines in the following manner:

- **Moodle Server** - *dommoodle*;
- **Automatic Assessment Server** - *domserver*;
- **Judgehost Servers** - *domjudge1* and *domjudge2*.

These machines run with an Intel®Xeon®Processor E7330 (6M Cache, 2.40 GHz, 1066 MHz FSB) CPU and with 1 GB RAM, with the exception of *domserver*, for which was reserved 2 GB. Although, as learned in the case study discussed later on, the VM running Moodle would have greatly benefited from extra RAM.

5.1.2 Software Used

In every virtual machine, the following tools were installed:

- Apache/2.2.16 (Debian);
- PHP 5.3.3-7+squeeze1 with Suhosin-Patch (cli);
- git 1.7.2.3.

More specifically in *domserver* and *dommoodle*, both MySQL and phpMyAdmin were installed, being the latter to provide an interface with the database. The respective versions were used:

- MySQL Ver 14.14 Distrib 5.1.49, for debian-linux-gnu (x86_64) using readline 6.1;
- phpMyAdmin 3.3.7deb5.

Concerning the local development for this thesis, the full list of the software used is as follows:

- Mac OS X Snow Leopard (version 10.6.7);
- TextMate Version 1.5.10 (1623);
- Oracle SQL Developer Version 3.0.04 Build MAIN-04.34;
- Transmit Version 4.1.5 (MAS);
- Google Chrome 12.0.742.100;
- git version 1.7.3.4;
- Apple clang version 2.0 (tags/Apples/clang-137) (based on LLVM 2.9svn);
- PHP 5.3.4 (cli) (built: Jan 17 2011 21:54:20);
- Dropbox v1.1.3.5;
- TeXworks Version 0.2.3 (r.466).

DOMjudge 3.1.3 was installed in *domserver*, *domjudge1* and *domjudge2*, and Moodle 2.0.2 (Build: 20110221) in *dommoodle*. The special software requirements for the static analysis are covered in appendix [C](#).

5.1.3 Installation Process

The full installation process for DOMjudge and Moodle can be found in appendix [A](#), whereas the installation of the advanced modules static analysis of code quality and plagiarism detection, are covered respectively in appendixes [C](#) and [D](#).

5.1.4 Integration with Moodle 2.0

In the course of the development of the CBA prototype, only a few Moodle 2.0 Preview release notes had been announced, and it went through a very long changing process until the installation used in this dissertation, Moodle 2.0.2.

As expected, this brought some major setbacks while configuring the Programming Assessment module in Moodle which caused the PHP code to halt related to file handling. This was probably due to possible changes in the file API from the early versions to the current version.

After identifying the source of the problem, the following changes needed to be made so that the system could regain its previous functionality:

- Adding the parameter *component* to every file access, with the value *mod_prog-assessment*;
- Adding the parameter *mod_progassessment* to every method call related with file access;
- Including the parameter *component* in how links were being build to the files;
- A fix to a SQL query in a web service in `frontend.php` which was not transmitting all submissions to Moodle;
- Adding a condition to the `cron` method of `lib.php` to ensure all retrieved files are parsed.

5.2 Integrating New Programming Languages in the CBA System

As discussed in the previous chapter, adding another language to the CBA system has already been validated by Pacheco [Pac10] in his case study using the programming language Scheme.

In this dissertation, it has been specified as a requirement to further prove the extensibility of the CBA system, by adopting support for SQL assessments in Oracle databases. Since evaluating SQL queries does not fit in the standard scheme of input-output supported natively by DOMjudge (as seen in chapter 2), the workflow had to be changed. This section details every step of the implementation of this case study in the CBA system.

5.2.1 Input and Output

While an example for an input and output file for a hypothetical C++ assessment with two test cases could be:

Listing 5.1: C++ assessment input file

```
#testcase
#weight 50
#name gcd_test
gcd
12 18

#testcase
#weight 50
#name prime_test
prime
```

Implementation

```
2
3
4
```

Listing 5.2: C++ assessment output file

```
#testcase
6

#testcase
1
1
0
```

A SQL assessment needs to know which database to connect to and which credentials to use. Therefore, the input files use the same system keywords (`#testcase` and `#weight`) and takes the *username*, *password*, *hostname*, *port* and *SID* parameters to specify an Oracle database connection. On the other hand, the output file should be the query that will generate the correct solution. The following input and output files are an example of two test cases in different Oracle accounts in the same server:

Listing 5.3: SQL assessment input file

```
#testcase
#weight 50
#name sql_test1
username = domjudge
password = judgedoom
hostname = oraalu.fe.up.pt
port = 1521
SID = ALU

#testcase
#weight 50
#name sql_test2
username = Concurso1
password = concurso
hostname = oraalu.fe.up.pt
port = 1521
SID = ALU
```

Listing 5.4: SQL assessment output file

```
#testcase
select nif_pessoa, nome, area, grau
from Pessoa p
inner join Competencia c on c.cod = p.cod_competencia
order by nome

#testcase
select nif_pessoa, nome, area, grau
from Pessoa p
inner join Competencia c on c.cod = p.cod_competencia
order by nome
```

By using connection strings as input test case files, the system can virtually be extended to support every Database Management System (DBMS), as long as a database type parameter is added to select the connection driver to be used. The DOMjudge architecture needs every supported compiler installed in the Judgehosts [EKW10]. In a similar fashion, a PHP database connection driver needs to be installed in every machine. The recommended driver for Oracle [Jon11] and the one used in the current implementation is *php-oci8*. A more detailed guide on how to install the connection driver can be found in appendix B.

5.2.2 Connection Library

A set of functions in PHP were written in order to establish a connection and interact with the Database server. Although these are Oracle specific, any DBMS will use similar methods, which calls for the need of an abstract high-level class containing the following set of functions to be implemented in each extending class:

- `openConnection` - establishes a connection with the database server;
- `closeConnection` - closes a connection with the database server;
- `execQuery` - executes and returns the query result;
- `getColumnArray` - retrieves the column names of a query;
- `getRowMatrix` - returns a matrix containing all rows and columns, including the column names, or an error if it does not succeed;
- `compareMatrices` - compares two query matrices and outputs the differences to an array;

Implementation

- `printMatrixToHTML` - prints the query matrix to a HTML table (used for testing purposes);
- `printPretty` - prints the errors in a human-readable format, if any.

The method `compareMatrices` is currently able to compute the number of different rows and tell the user whether the student submission has a different total number of columns, rows or different column names from the solution. The comparison method is resilient to different column ordering i.e., the submission will be graded full marks even though the order of the columns differ from the solution. Listing 5.5 shows an example of a wrong SQL submission:

Listing 5.5: Wrong SQL query result

```
Wrong answer ("The total number of columns between the
submission and solution did not match.
The total number of rows between the submission and
solution did not match.
5 row(s) were different from the solution.") .
```

A error compiling a SQL query in the database server would be communicated to the user with a system message:

Listing 5.6: Malformed SQL query result

```
Unknown result: Wrong answer ("ORA-01861: literal does not
match format string").
```

Lastly, a correct submission shows the running time:

Listing 5.7: Correct SQL query result

```
Correct ("Accepted")! Runtime is 0.009 seconds.
```

5.2.3 Running the Program

In order to override the standard input-output scheme, special run scripts and compare scripts have to be specified upon the assessment creation. Providing the functions are used as defined in the connection library, the developed shell scripts are generic enough to work for every DBMS. These scripts were developed following the DOMjudge and the ICPC validator interface standard [CSU08].

The overall logic is to bypass every step as silently as possible via piping until the compare step, where the run script will be called for both the submission and the solution to generate a matrix for each query result. Any semantic error will be filed as a compiler

error and the comparison result will be either blank (in case the submission is correct) or describing the differences in the query results.

The task of each script can be briefly summarized as follows:

- **compile_sql.sh**: do not compile - bypass step;
- **run_oracle.sh**: do not run now - bypass step;
- **compare_oracle.sh**: run the check script and handle its output by writing the results in the appropriate files;
- **check_oracle.sh**: call the runjury script for both queries and write the comparison result;
- **runjury_oracle.sh**: write a matrix generated from a SQL query.

5.2.4 Moodle Integration

Concerning the integration with the Moodle system, there were some minor tweaks which had to be done in order to support SQL as a language. The web services which communicate with the DOMjudge main server have to specify the special run and compare scripts if a SQL assessment is detected and the cron task handles the DOMjudge database information differently. Namely, since the ICPC validator interface standard only supports three types of answers - correct, wrong answer and unknown result - unknown results have to be parsed as compiler errors and updated in the appropriate table column. Although, this is only relevant for the Compilation Playground feature.

An important factor when creating SQL assessments is considering the use of skeleton code in every query in which the solution does not contain an *ORDER BY* clause. Different queries can originate the same solutions with different row orders and the system will mark it as a false negative. The most practical workaround to this issue is using the skeleton code to wrap the student query with an *ORDER BY* clause, as it is shown in listing 5.8.

Listing 5.8: Skeleton code for a SQL assessment

```
select *
from (
  -- studentcode
)
order by 1 desc
```

The teacher has to make sure that the solution abides by the skeleton code standard. An example of a solution query following this scheme would be:

Listing 5.9: Wrapped output file for a SQL assessment

```
#testcase
select nif_empresa, designacao
from Empresa
order by 1 desc
```

5.3 Throwing a Contest with DOMjudge

For one to throw a contest with the CBA system while taking advantage from the features of DOMjudge, the system has to be set properly in order to ensure nothing fails while the contest is occurring.

5.3.1 Before Starting

DOMjudge

Firstly, it is strongly recommended to have a grasp of the jury interface of DOMjudge by reading the manual [EKW11b], since it is a more complete way of checking all the results in real time.

Secondly, it might be a good idea to isolate all submissions in a separate contest. To achieve this, the best way is toggling the current *progassessment* contest as inactive and create a new contest with the appropriate date and time.

Should a new contest be activated, it is mandatory that the name is changed in the file `frontend.php` under the deployed *frontend* folder. This step can be in the future skipped with the implementation of a function to get the current active contest, since DOMjudge forces that there can be only one contest running at a given time.

If the submission times are not intended to make a difference on the classification, the activation date should be the same as the end date.

By default, DOMjudge penalizes a user for 10 minutes for each incorrect submission. In this context, the user will be penalized for every single test case which is not correct. It is up to the teacher to calculate the penalty time or to set it as 0 if he wants to. This property can be found and changed on the main DOMjudge architecture server under the `/domjudge/etc/domserver-config.php` file.

Lastly, for testing purposes, there should be at least one team flagged as Organization, since its results will not appear in the public scoreboard. This property can be set via jury interface to a team which has made at least one submission. It is highly recommended that this team is owned by a teacher or someone with testing responsibilities.

Moodle

Concerning Moodle use, the teacher should use a separate topic for creating every exercise so that all the other topics can be hidden. All assessment parameters should be equal and the beginning and end dates in conformation with the specified in DOMjudge.

It is important to warn students of exercises with skeleton code to avoid needlessly wrong submissions.

Every other concern is related to security within Moodle and blocking external access. These issues are generally handled by the system administrator.

5.3.2 Mid-contest

While the contest is in course there will probably not be any need for interfering. Nevertheless, if the students keep submitting a solution wrong, it might be worthwhile to check if there is any problem with the exercise or even give a hint. In case any system problem occurs and a submission is not properly judged or not judged at all, the teacher can command the system a rejudge order.

The best approach is to keep one eye on the scoreboard and other on the jury interface to quickly fix any problem in case one appears.

5.3.3 Gathering Results

When the contest reaches its end is when DOMjudge can shine the most. A detailed scoreboard (e.g. figure 5.1) can help a teacher to find the number of attempts on each exercise, exercises solved per student and a ranking based on correctness and time. This information, when combined with the exporting power of Moodle, provides the teacher diverse grading options and more evaluation parameters when comparing to the use of the Moodle programming assessment module by itself.

In the end, a link to the public scoreboard can be communicated to the students so that they can compare their performance with their colleagues’.

5.4 Static Analysis of Code Quality

The static analysis module required the most effort, since it was developed to support any extent of metrics and programming languages and is comprised of the greatest number of implementation steps:

1. Choosing the source files parsing tools and preparing the auxiliary files to be used by the metrics;
2. Choosing the metrics to develop and implement;

Implementation

#	AFFIL.	TEAM	SCORE	131	131_190	131_191	133	133_194	133_195	134	134_196	134_197	135	135_198	135_199	137	137_202	137_203	138	138_21	
1		ci06008	18	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	0
2		ci08022	12	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	0
3		ci08001	10	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1
		ci08020	10	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	0
5		ci08031	8	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	1
		ci08053	8	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	0
		ci08054	8	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	1
		ci08057	8	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1
		ext10466	8	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	1
		mie10010	8	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	1
11		ci06039	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	0
		ci08003	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	0
		ci08004	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	0	0	0	0
		ci08015	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	1
		ci08039	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	0
		ci08043	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	1
		ci08047	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	0	0	0	0
		ci08056	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	0	0	0	1 (0 + 0)	1 (0 + 0)	0	0
		ci08059	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	0
		ci08061	6	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	0	0	0	1 (0 + 0)	1 (0 + 0)	0	0
21		ci05031	4	0	0	1 (0 + 0)	1 (0 + 0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		ci06023	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	0
		ci06042	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	0
		ci07015	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	0	0	0	0	0	0	0
		ci08024	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	0	0	0	1
		ci08041	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	1
		ci08045	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	0	0	0	0	0	0	0
		ci08050	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	0	0	0	0	0	0	0
		ext10467	4	0	0	1 (0 + 0)	1 (0 + 0)	0	1	1	0	1	1	0	1	1	0	0	0	0	0

Figure 5.1: Scoreboard of a finished SQL contest

3. Providing an effortless way to compute the metric results;
4. Integrating the module in the DOMjudge architecture;
5. Integrating the module in the Moodle plugin.

The metrics developed only apply to the C and C++ programming languages, since integrating more languages would mean repeating the first two steps of the whole procedure, which would be unrealistic due to the time at hand.

5.4.1 Parsing Techniques

In order to analyse the source code in a static fashion a simple text parsing would not suffice, since it is hard to account for every programming language feature. Consequently, the chosen technique was an approach using Abstract Syntax Trees (AST).

The first option was using the GCC compiler for dumping an AST representation using the `-ftree-dump` command. The result was very verbose and difficult to parse. The second option was using the LLVM (Low Level Virtual Machine) collection of compilers, an open-source solution sponsored by Apple, which provides Clang, the front-end for C, C++ and Objective-C compilation.

Maturity reports on Clang claim that its performance is superior in most cases to GCC [Fan10] and, most importantly, it provides a compilation flag which dumps a XML representation of an AST (`-ast-print-xml`). The downside to this strategy is the code

needing to be successfully compiled; nevertheless, it is pointless to give a student a mark different from zero if his code does not pass the compilation step.

In order to parse the XML file generated, it was chosen a grep-like utility directed for XML files - *XGrep* - freely available on <http://wohlberg.net/public/software/xml/xgrep/>. The dump is parsed using *XGrep* after identifying the important information to extract using grep.

To calculate most metrics it was needed a source file with the least noise possible. A sed script is used to generate a new source file with no comments.

Finally, a shell script was developed to be used in conjunction with the aforementioned tools to automate the whole process. The simplified workflow of the script is as follows:

1. Check the file extension to determine if the file is a C or C++ source;
2. Generate the AST dump with Clang;
3. Determine the important sections to be extracted using grep;
4. Output the result of *XGrep* to a new file specified as a script parameter;
5. Run the sed script and output the source file without comments to the file with the format `_nc_<source file>`.

The resulting XML file and the source file without comments are the most important input files for metric calculation, as covered in the next section. Details on how to configure *XGrep* and Clang can be found on appendix C.

5.4.2 Metrics Development

As proposed on chapter 4, both the Halstead complexity measures and the McCabe's cyclomatic complexity were the primary focus and were implemented successfully. A set of style metrics, namely the ones used in Submit! (section 3.1.1), were implemented as well.

This spanned a total of 20 metrics which were subdivided in three distinct groups:

- **Halstead:** fitting the 12 metrics covered by software science;
- **Style:** all metrics concerning code style;
- **Misc:** every metric which does not fit in the other groups.

The metrics were developed as bash scripts, relying heavily on the use of Unix utilities such as grep, awk, sed and bc for more complex arithmetic operations. Every metric was unit tested against a set of sample C and C++ sources.

These scripts include an opening snippet which should be included in every script to inform the user of the expected parameters. An example of this opening snippet for a script which takes as input the source file and the XML file is shown on listing 5.10.

Listing 5.10: Opening snippet for a metric script

```
ME="$(basename "$(test -L "$0" && readlink "$0" ||
    echo "$0")")"

if [ $# -ne 2 ]; then
    echo "usage: \"$ME\" <source code file> <xml dump file>"
    exit
fi
```

5.4.2.1 Implementation Details

Halstead

The Halstead software science metrics were implemented according to the Verifysoft guidelines [Gmb10a], as covered in chapter 3.1.

All of these metrics take the generated XML file and the source file stripped out of comments as input parameters and output a numerical value.

The two crucial steps of the calculation are identifying the number of operators and operands. The following list explain how these cases were handled:

- **Number of unique operators:**

1. Search for an occurrence of a storage class specifier or type qualifier using `grep -w` in the source code (*auto*, *extern*, *inline*, *register*, *static*, *typedef*, *virtual*, *mutable*, *const*, *friend*, *volatile*) and increment the counter for each one that is found;
2. Search for an occurrence of a reserved word using `grep -w` in the source code (*asm*, *break*, *case*, *class*, *continue*, *default*, *delete*, *do*, *else*, *enum*, *for*, *goto*, *if*, *new*, *operator*, *private*, *protected*, *public*, *return*, *sizeof*, *struct*, *switch*, *this*, *union*, *while*, *namespace*, *using*, *try*, *catch*, *throw*, *const_cast*, *static_cast*, *dynamic_cast*, *reinterpret_cast*, *typeid*, *template*, *explicit*, *true*, *false*, *typename*) and increment the counter for each one that is found;
3. Extract all binary and unary operators from the XML file and insert them in a map and add the number of keys to the counter;
4. Find if there are conditional and compound expressions present and add one to the counter for each distinct occurrence of these entries in the XML file;

5. Print the counter value.

- **Total number of operators:**

1. Count all occurrences of storage class specifiers and type qualifiers and add the result to the counter;
2. Count all occurrences of reserved words and add the result to the counter;
3. Count all binary and unary operators as well as conditional and compound expressions present in the XML file and add the result to the counter;
4. Print the counter value.

- **Number of unique operands:**

1. Count all variable declarations and class / struct field declarations in the XML file and add the result to the counter;
2. Search for an occurrence of a type specifier using `grep -w` in the source code (*bool, char, double, float, int, long, short, signed, unsigned, void*) and increment the counter for each one that is found;
3. Extract all integer, character and string literals from the XML file and insert them in a map and add the number of keys to the counter;
4. Run a sed script to extract all the floating point literals from the source code to a temporary file, sort the literals removing the duplicates and count them using `grep`, adding the result to the counter;
5. Print the counter value.

- **Total number of operands:**

1. Count all variable declarations and class / struct field declarations in the XML file and add the result to the counter;
2. Count all occurrences of type specifiers and add the result to the counter;
3. Count all literal occurrences present in the XML file and add the result to the counter;
4. Print the counter value.

The remaining metrics can be calculated with simple mathematical operations using these values. The only exceptions are the metrics which use floating point values, since the operation must be piped into `bc` to retrieve the correct result.

Style

The style metrics were developed in two different flavours, a simple and a verbose version. The simple version prints 1 if the style violation occurred and 0 if the source code successfully passed the verification, whereas the verbose version outputs the lines and the respective line number which caused the verification to fail.

All metrics take as the sole parameter the source code without comments, with the exception of the "Program line size" metric, which expects an additional parameter, the line size.

The implemented metrics were the same as the ones in the CBA system Submit! [PRS⁺03]:

- **Block brackets in a single line:** detects a violation if " { " and " } " are present in the same line in the source code;
- **Check if tabs exist:** checks for tab characters using awk and counts the occurrences. If there is any, print 1;
- **Program line size:** measures each line size using awk and outputs 1 if the size surpasses the limit specified in the input parameter;
- **Space between brackets and code:** detects a violation if there is any character after an opening bracket (" { ");
- **Space between comments and code:** checks if there is any character that is not a space following " / / " or " / * ", printing 1 if such a line exists;
- **Space between parentheses and brackets:** detects a violation if the string ") { " occurs in the source code.

Misc

The only two metrics implemented which do not fit in the other groups are the "Lines of code" and "McCabe's cyclomatic complexity" metrics.

The former is a simple count of all lines which are not empty and takes as parameter the source code file, while the latter takes as input the generated XML file and was implemented and tested following the Watson and McCabe's algorithm [WMW96], which can be briefly explained as follows:

1. Initialize the $v(G)$ counter as 1;
2. Increase the $v(G)$ counter by 1 for each conditional expression in the code: *if* statements, *case* in switch statements and ternary operators;

3. Increase the $v(G)$ counter by 1 for each loop expression in the code: *for*, *while* or *do* statements;
4. Increase the $v(G)$ counter by 1 for each logical branch in the code: "&&" (logical and) or "||" (logical or);
5. Print the $v(G)$ counter.

5.4.3 Metrics Library

Two PHP files, one with a set of functions and another with constants, were developed to handle the file generation and metric calling.

The constants file contains a few strings and mappings, being the most relevant:

- Programming language extensions to programming language names;
- Programming language names to the respective dump script locations;
- Metrics to their script locations;
- Definition of which metrics are available for a given programming language.

The library file benefits from the aforementioned constants file and is meant to be used to automatically compute the results of all metrics for any source file. The workflow can be outlined as:

1. Accept the source file as a parameter, and optionally the name of the XML file to be generated;
2. Exit with an error if the source file does not exist;
3. Get the programming language by the source file extension;
4. Exit with an error if the programming language is not supported;
5. If the source file without comments or the XML file does not exist, generate them using the dump script mentioned on section [5.4.1](#);
6. Exit with an error if there was any error running the dump script;
7. Get all available metrics for the source file programming language;
8. Execute all the metrics and store the results;
9. Export the results;
10. Remove all generated files.

The results are exported in the form of a PHP array to be imported at a later time by Moodle.

5.4.4 Integration with DOMjudge

This integration needed to be done in two steps, one for all the Judgehosts and another for the DOMserver.

Judgehost

Firstly, it was required to install all the software dependencies: LLVM, Clang and *XGrep*. The complete and more thorough instructions for this installation can be found in appendix C.

Then, the metrics package needed to be copied to the library folder in order to be used by the compiling scripts. By running the PHP library at the Judgehost compiling step, it is ensured that all output is appended to the compiler output and, consequently, transmitted to Moodle. An example of a modified compile script for the C language is as follows:

Listing 5.11: Modified C compile script

```
#!/bin/sh

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

gcc -Wall -O2 -static -pipe -o $DEST $SOURCE -lm
EXITCODE=$?

# static analysis
cp -r /home/domjudge-run/domjudge/judgehost/lib/static/* .
php metricslib.php $SOURCE

exit $EXITCODE
```

DOMserver

The last step was letting the Moodle users know which metrics exist for a given programming language.

The function `getMetrics` was added to the web services and returns an array abiding to the format:

`METRICS[<programming language>][<group key>][<metric key>] = <metric name>`, where each parameter corresponds to the rules:

- `<programming language>` is the programming language name, for instance, C++ or Scheme;

Implementation

- `<group key>` is one of the metric subgroups: *halstead*, *style* or *misc*;
- `<metric key>` is the key name for each metric, such as *spaceCommentsText*;
- `<metric name>` is the proper name for each metric to be shown in the Moodle interface, like "Space between a comment tag and text".

5.4.5 Integration with Moodle

The integration at the Moodle plugin level consisted of the following steps:

- Changing the database schema to include the static analysis information;
- Building an interface for selecting the metrics to be computed in the static analysis grade;
- Calculating the final grade and showing the result to the student.

For each instance of *progassessment*, it is kept track whether the static analysis is enabled and the percentage of the total grade reserved for this evaluation. When it is enabled, the metrics in use are recorded in a separate table specifying a minimum and maximum accepted values for each metric and the weight in the static analysis grade.

The function `getMetrics` is called when the interface to create or update a programming assessment is loaded. The form section "Static analysis" is replicated for every programming language, being shown only the one corresponding to the selected programming language, dynamically updating upon change with JavaScript.

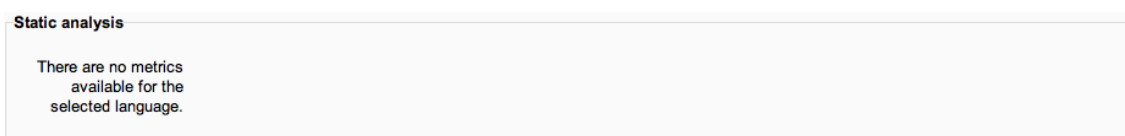


Figure 5.2: Static analysis interface when no metrics are available in the Programming Assessment form

If there are no metrics available for the selected language, the teacher is informed, as shown in figure 5.2, being unable to interact with any control.

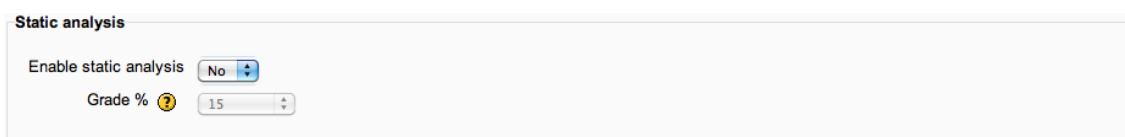
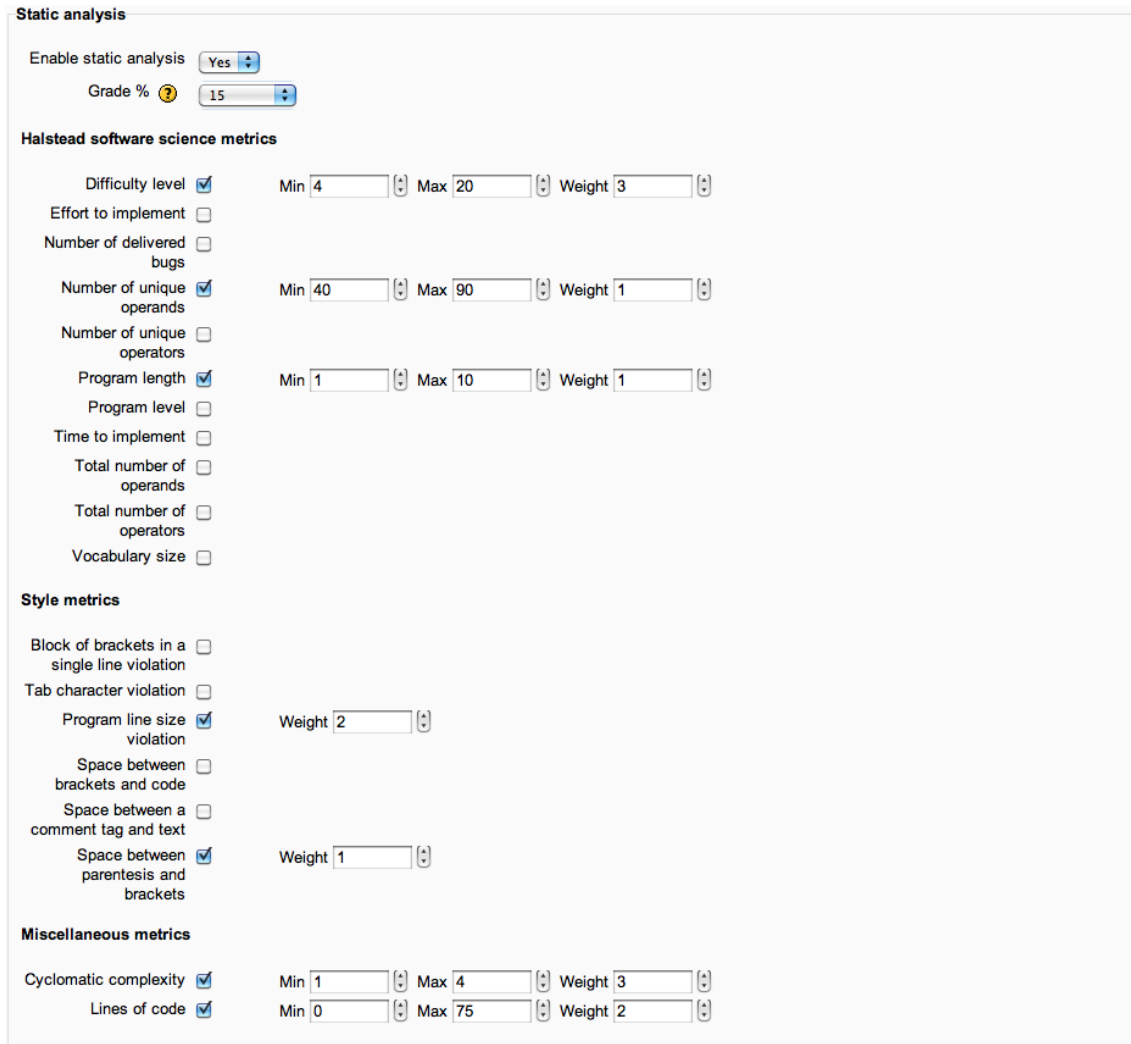


Figure 5.3: Collapsed static analysis interface in the Programming Assessment form

Implementation

Conversely, if any metrics exist for the selected programming language, a simple interface containing a control to toggle the use of the static analysis module and the grade percentage (figure 5.3) is shown. If the user enables the module, the interface is automatically expanded with all the fetched metrics divided in groups in the form of checkboxes.



The figure shows a web interface titled "Static analysis". At the top, there is a toggle for "Enable static analysis" set to "Yes" and a "Grade %" input set to "15". Below this, the interface is divided into three sections: "Halstead software science metrics", "Style metrics", and "Miscellaneous metrics". Each section contains a list of metrics with checkboxes and, for some, input fields for Min, Max, and Weight.

Section	Metric	Checked	Min	Max	Weight
Halstead software science metrics	Difficulty level	<input checked="" type="checkbox"/>	4	20	3
	Effort to implement	<input type="checkbox"/>			
	Number of delivered bugs	<input type="checkbox"/>			
	Number of unique operands	<input checked="" type="checkbox"/>	40	90	1
	Number of unique operators	<input type="checkbox"/>			
	Program length	<input checked="" type="checkbox"/>	1	10	1
	Program level	<input type="checkbox"/>			
	Time to implement	<input type="checkbox"/>			
	Total number of operands	<input type="checkbox"/>			
	Total number of operators	<input type="checkbox"/>			
Vocabulary size	<input type="checkbox"/>				
Style metrics	Block of brackets in a single line violation	<input type="checkbox"/>			
	Tab character violation	<input type="checkbox"/>			
	Program line size violation	<input checked="" type="checkbox"/>			2
	Space between brackets and code	<input type="checkbox"/>			
	Space between a comment tag and text	<input type="checkbox"/>			
Miscellaneous metrics	Space between parenthesis and brackets	<input checked="" type="checkbox"/>			1
	Cyclomatic complexity	<input checked="" type="checkbox"/>	1	4	3
	Lines of code	<input checked="" type="checkbox"/>	0	75	2

Figure 5.4: Expanded static analysis interface in the Programming Assessment form

Upon checking a metric checkbox, it dynamically shows the *Min*, *Max* and *Weight* fields, which are number input controls that automatically validate the correctness of the inserted information via HTML5. An example of a possible configuration of this module for a given programming assessment is illustrated by figure 5.4.

The only exception is the "Style" group, where the expected minimum and maximum values are always 0 and the controls remain hidden.

Concerning the grade calculation, it takes place before the final mark is inserted or updated in the database in the `cron` method. The static analysis maximum grade is always

100 and is calculated as a weighted mean, according to the formula 5.1, where n is the total number of metrics, $weight_i$ is the weight of a given metric and the value of x_i follows the conditional expression described below:

$$x_i = \begin{cases} 1 & \text{if } (min_i \leq metric\ result \leq max_i), \\ 0 & \text{otherwise.} \end{cases}$$

$$SG = \frac{\sum_{i=1}^n weight_i \cdot x_i}{\sum_{i=1}^n weight_i} * 100. \quad (5.1)$$

The final grade (FG) is consequently computed using the equation 5.2, where G is the grade obtained after applying the penalty time calculations, SP is the percentage of the grade reserved to the static analysis and MG is the maximum grade that can be attained for the current programming assessment.

$$FG = G * \frac{100 - SP}{100} + SG * \frac{MG}{100} * \frac{SP}{100}. \quad (5.2)$$

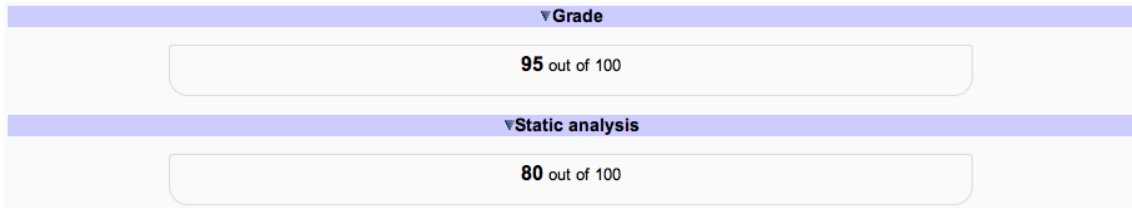


Figure 5.5: Static analysis grade interface

The student view was updated with an additional component, the "Static analysis" section, which reports the SG value. In the example of figure 5.5, a student who scored 100 out of 100 in the grade and 80 out of 100 in the static analysis component when it weighted 25% of the final grade, had a final score of $100 * \frac{100-25}{100} + 80 * \frac{100}{100} * \frac{25}{100} = 95$ out of 100.

5.4.6 Extensibility

This advanced module was developed with extensibility in mind. In order to integrate more metrics in the system for the currently supported programming languages, it is advised to program a bash script which follows the example given by one of the 20 metrics implemented.

Adding a new programming language may require more work, since it might be necessary to find tools with similar capabilities of both Clang and *XGrep*, implement a new dump script and update the PHP libraries with the location of the new metrics.

From an administrator point of view, a front-end for uploading metrics for any programming language would be desirable, or a more detailed and powerful metric management interface. Although, in order to take a big step forward, the module would benefit from an interface where the student could learn about the metrics he did not pass and understand how to overcome them. The verbose style metrics developed, which are not currently in use, could play an important informative role in the case such a feature was implemented.

5.5 Plagiarism Detection

Due to a lot of different possibilities for implementing plagiarism detection, this chapter exposes the choice process, details the configuration parameters of the implemented plugin and focuses on the adaptation of the current system to the new plagiarism API released with Moodle 2.0.

5.5.1 Choice

The study comprised in the previous chapters allowed to highlight the following possibilities for implementing plagiarism detection:

- Implementation of an algorithm;
- Development of a plagiarism detection plugin integrating an external tool;
- Using an existing plugin:
 - Crot;
 - Turnitin.

For this deliberation, the RKR-GST was considered the algorithm of choice to implement, alongside the possible need of applying heuristics to improve its performance.

A few important factors were taken into consideration when choosing the implementation method:

- Difficulty of integration in the existing module;
- Time to implement the algorithm;
- Dependence of external services.

Table 5.1: Comparison of different plagiarism detection implementations.

Factor	Implementation			
	RKR-GST	External Tool	Crot	Turnitin
Difficulty of integration	*****	*****	***	***
Time to implement the algorithm	****	*	*	*
Requires external services	NO	YES	NO	YES

The table 5.1, measuring each parameter, was used to support the final decision.

Providing the difficulty of integration was rated in a five-star scale from easy to hard, Crot and Turnitin were both awarded a 3 since it is needed to adapt the current module to support the plagiarism API hooks. Both the algorithm and the external tool integration scored the maximum due to the need of developing a plagiarism detection plugin followed by the need to extend the module because of the motive previously specified.

The time to implement the algorithm was rated in a similar scale and is only relevant for RKR-GST, since the other solutions do not require algorithm implementation.

Concerning the dependence of external services, only the local implementations of the algorithm are not dependent, although Crot may benefit from the use of a Bing API key for global search.

The main choice, considering the time available, was using an existing plagiarism detection plugin. Crot was selected since it is a local, free and open-source solution.

5.5.2 Crot Analysis

As reviewed in section 4.1.3, Crot uses a fingerprinting algorithm - "Winnowing" - loosely based on MOSS. Figure 5.6 presents a simple diagram illustrating the main steps of the algorithm for both local and global search [BS09]. It is possible to select if Crot is activated on a per assignment basis, as well as configure if which types of search are desired: local, global or both.

All fingerprints are stored in the Moodle database and can be used to compare files independent of the course and of the year.

Upon comparison, similarities are recorded in the database as well and a teacher can click on a submitted assignment to check the similarity sources (figure 5.7) and compare them to decide whether it is an instance of plagiarism or not.

Implementation

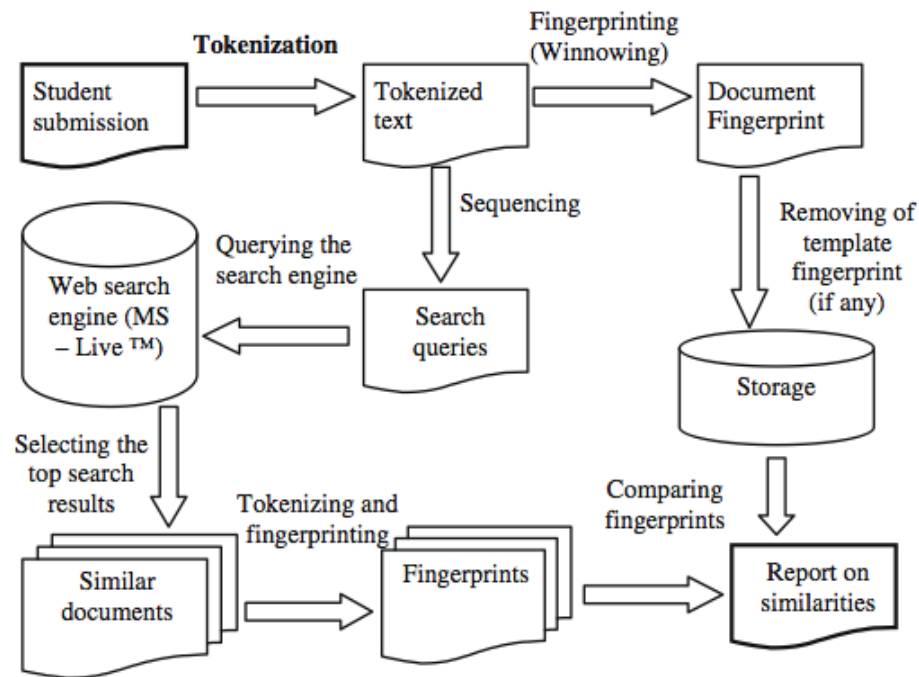


Figure 5.6: Crot main steps for local and global search (Source: [BS09] p.783)

Similar assignments		
Name	Course	Similarity score
http://dic.academic.ru/dic.nsf/ruwiki/64	Web document	100 %
http://ru.wikipedia.org/wiki/%D0%98%D1%8	Web document	100 %
http://dic.academic.ru/dic.nsf/ruwiki/11	Web document	100 %
http://www.world-wide-gifts.ru/index.php	Web document	100 %
http://www.otpusk.com/ref/kr/info-histor	Web document	93.94 %
http://ru.pandapedia.com/wiki/%D0%AE%D0%	Web document	90.91 %
http://www.carstock.ru/Dictionary/%D0%A0	Web document	90.91 %
http://ru.science.wikia.com/wiki/%D0%98%	Web document	87.88 %
http://tutit.ru/countrylist/korea-sout	Web document	87.88 %
http://www.grinweb.ru/	Web document	48.48 %

Figure 5.7: Crot assignment global similarity (Source: abridged from [But11])

In order to customize how Crot compares the files, it is possible to fine-tune some algorithm parameters. Here follows a list of the available options and the effect on the algorithm:

- **Grammar size** - size of text used to calculate one hash in the document fingerprint;
- **Window size** - represents how granular text search will be;

- **Maximum distance between hashes in the text cluster** - used for allocation of hashes into text clusters in the colouring of similar text fragments;
- **Minimum cluster size** - the minimum number of hashes in the text cluster used for colouring of similar text fragments.

Concerning the global search, there are specific parameters:

- **Global search query size** - the number of words in the query for global search;
- **Percentage of search queries for web search** - the percentage of randomly selected queries from all search queries for web search;
- **Number of web documents to be downloaded** - how many web documents will be downloaded to the Moodle server from the list of possible sources on the web;
- **Culture info for global search** - used in queries in the Bing search engine;
- **Maximum file size** - files with the size greater than this value are not downloaded from the Internet.

Finally, the administrator can specify the student disclosure, which is the message that will appear in every assignment to warn the user his files will be submitted to a plagiarism detection system, as well as the default threshold, the minimum score for which assignments are displayed as plagiarism instances.

More details on how to install and configure this plagiarism detection plugin can be found on [Appendix D](#).

5.5.3 Adapting the Programming Assessment Module

This section explains how the module was adapted to support any type of plagiarism detection plugin using the plagiarism API and how its interface was improved by adopting traits from the Assignment module.

The first step was analysing the *Advanced upload of files* Assignment code and study its integration with Crot. Then, it was possible to clearly outline what was missing in the Programming Assessment module which was needed for the integration with a plagiarism prevention tool:

1. Adding the student disclosure to the submission form - [figure 5.8](#);
2. Adding the number of submitted assessments with a link to the grading view - [figure 5.9](#);
3. Adding the option to download all submissions and a link to the course grades - [figure 5.10](#);

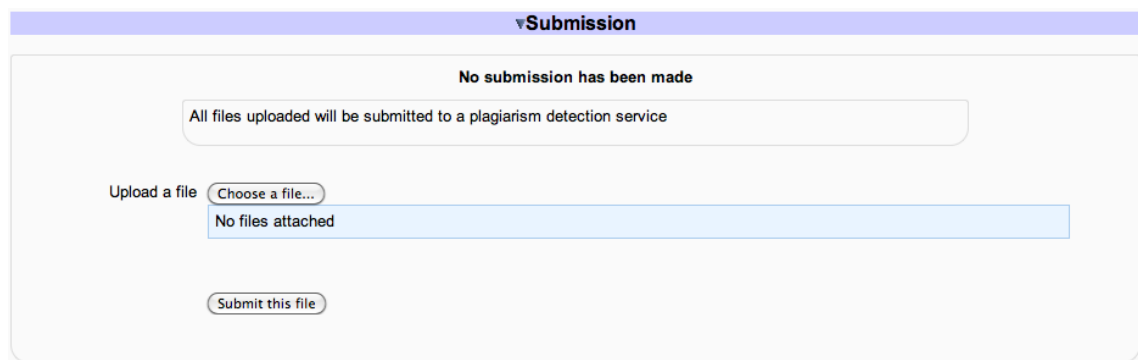
Implementation

4. Adding optional settings to customize the grading view - figure 5.11;
5. Setting up the full grading view table - figure 5.12.

The plagiarism API allows to add the student disclosure by including a call to `plagiarism_get_form_elements_module`, as shown in listing 5.12.

Listing 5.12: Student disclosure code snippet

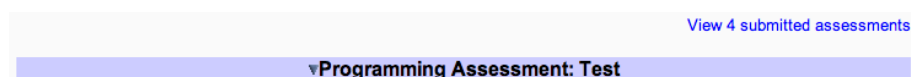
```
$course_context = get_context_instance(CONTEXT_COURSE,  
    $COURSE->id);  
plagiarism_get_form_elements_module($mform,  
    $course_context);
```



The screenshot shows a web interface for a submission. At the top, there is a purple header bar with the text "Submission". Below this, a message states "No submission has been made". A text box below the message says "All files uploaded will be submitted to a plagiarism detection service". Underneath, there is a section labeled "Upload a file" with a button "Choose a file...". Below the button, a blue box indicates "No files attached". At the bottom of the section, there is a button "Submit this file".

Figure 5.8: Student disclosure in Programming Assessment

Concerning the link to the grading view, the method developed was loosely based on the function `submittedlink`, available only for someone with the *grade* capability. Since the submitted assessments are not replaced, special care has been taken to count multiple submissions by one student as a single submission.



The screenshot shows a header bar for a programming assessment. It has a purple header bar with the text "Programming Assessment: Test". To the right of the header bar, there is a link "View 4 submitted assessments".

Figure 5.9: Number of submitted assessments in Programming Assessment

The downloading of all submissions was tweaked comparing to the original version from the *Advanced uploading of files* Assignment. When the link is clicked, the browser automatically downloads a compressed *zip* file with all the submissions of every student. This is intended and is due to the possibility of the grade being based either on the best submission or the last submission. Moreover, a teacher can check how a student has improved his submissions over time.

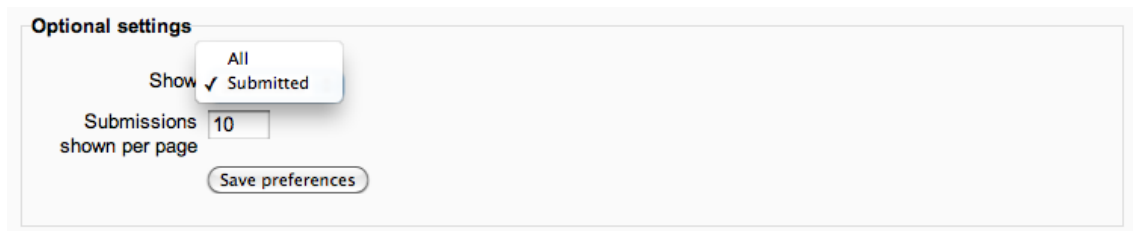
Based on the optional settings screen from Assignment, the options were cut down to showing all students or the students who submitted a solution. The quick grading

Implementation

[See all course grades](#)
[Download all submissions as a zip](#)

Figure 5.10: Options in the grading view of Programming Assessment

feature was not included since the grading is automatic and there is no need for manual intervention in the final score.



Optional settings

Show

Submissions shown per page

Figure 5.11: Optional settings in the grading view of Programming Assessment

When computing the grading view table, a few columns were left out, namely the *Comment*, *Last modified (Grade)*, *Status* and *Final grade*, as they are not needed and clutter the view. A new column, *Graded*, was added instead to notify the teacher whether Moodle has already fetched the submission outcome from DOMjudge.

The overall style of the table is heavily inspired on the Assignment one, using the same JavaScript library for expanding / collapsing folders when viewing the submitted files. Furthermore, every submission is contained in the folder named after its submission number (starting with 0).

Upon setting all plagiarism hooks and preparing the grading view interface, the system is now ready to adopt any plagiarism detection plugin.

5.5.4 Adapting Crot

After setting up the Programming Assessment Module with the plagiarism API hooks, the last task was adapting the Cron plugin to fetch the Programming Assessment submissions besides the *Advanced uploading of files* Assignment submissions.

The code snippet from the `crot_event_files_done` method in `lib.php`, triggered by the `assessable_files_done` event, which accesses the assignments, is presented in listing 5.13.

Listing 5.13: Fetching assignment submissions in Crot

```
$files = $fs->get_area_files($modulecontext->id,  
    'mod_assignment','submission', $eventdata->itemid);
```

Implementation

[See all course grades](#)
[Download all submissions as a zip](#)

First name : [All](#) [A](#)[B](#)[C](#)[D](#)[E](#)[F](#)[G](#)[H](#)[I](#)[J](#)[K](#)[L](#)[M](#)[N](#)[O](#)[P](#)[Q](#)[R](#)[S](#)[T](#)[U](#)[V](#)[W](#)[X](#)[Y](#)[Z](#)
Surname : [All](#) [A](#)[B](#)[C](#)[D](#)[E](#)[F](#)[G](#)[H](#)[I](#)[J](#)[K](#)[L](#)[M](#)[N](#)[O](#)[P](#)[Q](#)[R](#)[S](#)[T](#)[U](#)[V](#)[W](#)[X](#)[Y](#)[Z](#)

First name / Surname	Grade	Last modified (Submission)	Graded
student 1	100 / 100	0 student_solution.cpp 100% Monday, 23 May 2011, 06:50 PM	Yes
student 2	100 / 100	0 student_solution.cpp 100% Monday, 23 May 2011, 06:50 PM	Yes
student 3	100 / 100	0 source.cpp 100% Tuesday, 24 May 2011, 01:09 AM	Yes
student 4	100 / 100	0 source.cpp 100% 1 source.cpp 86.67% Tuesday, 24 May 2011, 01:18 AM	Yes

Optional settings

Show

Submissions shown per page

Figure 5.12: Full grading view of Programming Assessment

To this statement must be added the files from Programming Assessment, which need to be retrieved from their specific file area, *mod_progassessment*, as follows:

Listing 5.14: Fetching all submissions in Crot

```
$files = $fs->get_area_files($modulecontext->id,
    'mod_assignment','submission', $eventdata->itemid);
$progassessment_files = $fs->get_area_files(
    $modulecontext->id, 'mod_progassessment',
    'progassessment_submission', $eventdata->itemid);

if ($progassessment_files)
    $files = $files + $progassessment_files;
```

Implementation

The modifications that follow invalidate the possibility of using Crot in a module other than Programming Assessment, since every instance of Assignment is replaced:

- Every instance of the table *assignment* was replaced by *progassessment*;
- Every instance of the table *assignment_submissions* was replaced by *progassessment_submissions*;
- Every `require_once` statement including `mod/assignment/lib.php` was re-designed to include `mod/progassessment/lib.php`;
- Every Moodle capability referencing *assignment* was replaced by its *progassessment* equivalent;
- Every link referring *assignment* was switched to refer to *progassessment*;
- Each direct reference to the assignment ID (`$submission->id`) was replaced by `$submission->progassessment`.

Both `compare.php` and `index.php` were changed according to the specified rules. Shall the current module extend Assignment in the future, there is no need for any fix, since the file areas and the tables used will be the same.

The whole procedure permitted the link building in the grading view, allowing to navigate to the similarity report view, as shown in figure 5.13.

Student name	Similar assignments		
student 4	Name	Course	Similarity score
	student 3	FPRO	100 %
	student 4	FPRO	92.86 %


Global plagiarism detection is supported by Bing search engine 

Figure 5.13: Crot similarity report view in Programming Assessment

Upon clicking in a similarity score link, the user is redirected to the compare screen (figure 5.14 shows an example of two files with a 100% similarity score) where he can check side-to-side the two offending sources.

One last step which needs to be taken is enabling additional file extensions. In order to achieve this, the function `tokenize` in `locallib.php` must be updated with the intended programming languages. The code snippet 5.15 provides an example on how to add support to the C, C++, C#, Java, Python, Scheme and SQL programming languages.

Implementation

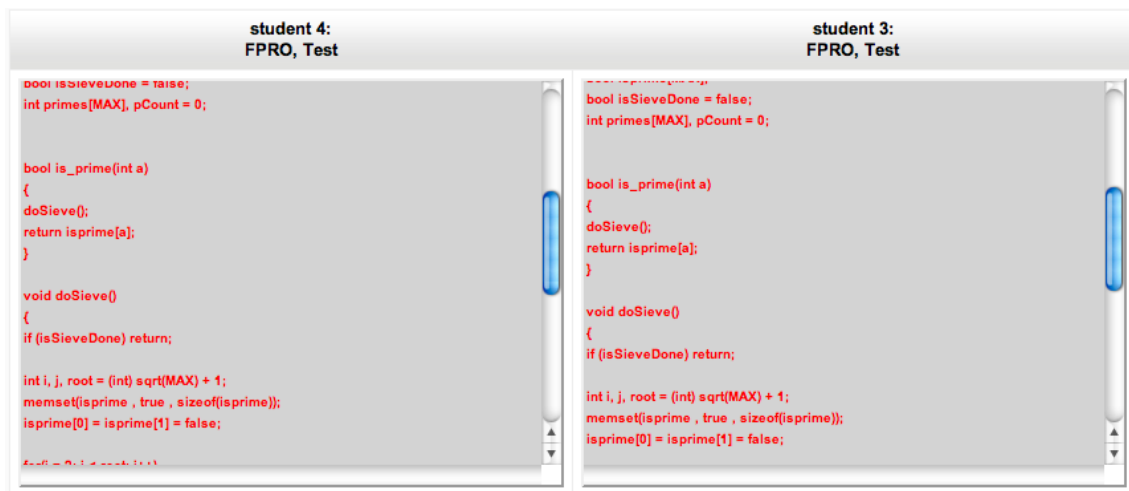


Figure 5.14: Crot compare view in Programming Assessment

Listing 5.15: Adding multiple file extension support to Crot

```
case "c":
case "cpp":
case "cs":
case "java":
case "py":
case "scm":
case "sql":
    return file_get_contents($path);
```

Implementation

Chapter 6

Results

This chapter presents an evaluation of the results achieved by the advanced module specification and by the CBA system, which was installed and configured in a test server running on four virtual machines provided by CICA.

It begins by confronting the initial specification of the modules to their final implementation state and detailing the hindrances which occurred during the development process. Besides, it covers every testing component conducted to the system, from a case study in the Databases course unit of LCI (Bachelor in Information Science) to unit tests and a report on the effectiveness of one module.

Finally, the chapter ends with a short synopsis and by drawing conclusions from the results obtained in the course of this thesis.

6.1 Implemented Features

This section comprises an overview of the implementation completeness of the modules specified in chapter 4, as well as a description of the setbacks which occurred during the development process.

The two most important modules, static analysis of code quality and plagiarism detection, were fully implemented and covered in chapter 5.

The static analysis module was developed and tested locally, in a machine running Mac OS X Snow Leopard. The Clang version running in this operating system is: *Apple clang version 2.0 (tags/Apple/clang-137) (based on LLVM 2.9svn)* as opposed to *clang version 1.1 (branches/release_27)* running on the Judgehosts' Debian GNU/Linux 6.0 (squeeze). Unfortunately, this caused a problem when running the flag `-ast-print-xml`

in Clang to compile C++ sources which caused the generated XML to be malformed. Although the metrics developed work for both the C and C++ programming languages, the system currently does not accept C++ sources for static analysis due to this software issue.

Regarding the feedback customization, it was taken a second important step forward with the implementation of the static analysis module. The first step had been taken by Pedro Pacheco, providing three default levels of feedback and allowing to personalize a message to show the student in case the submitted solution is right or wrong.

Although the current interface for creating programming assessments was not improved *per se*, the static analysis module is dynamic and uses HTML and JavaScript injection into Moodle forms, providing a set of important methods and a guideline to improve the usability to support the assessment creation.

The automatic correction of projects with multiple source files was not implemented, although two different ways of developing this module were outlined in chapter [4.1.2](#).

Finally, and despite not being a new feature, SQL was introduced to the CBA system as a new programming language with support for the Oracle DBMS, laying the ground work for the implementation of additional DBMS.

6.2 Tests

In order to test the modules implemented and the overall CBA system functionality, a variety of methodologies were used. This section briefly explains how a case study with the SQL programming language was conducted and how the two main advanced modules developed were tested, alongside with the conclusions retrieved from the obtained results.

6.2.1 SQL Contest

In order to test the CBA system in the context of a course unit, it was suggested by the teachers of the Databases course of LCI the use of the CBA platform in a SQL contest.

A new Moodle course was created in the test system with a total of 49 students; moreover, proper security measures were taken to ensure no interaction between Moodle users was possible. In this course, 10 programming assessments were set up with unlimited submissions and with the grading method "Best submission". Plagiarism detection was not enabled since it had not been totally implemented by the contest time.

All guidelines covered in section [5.3](#) for throwing a contest with DOMjudge were followed. However, in the contest day, possibly due to the Moodle virtual machine hardware limitations, the system crashed with the number of simultaneous connections. According to the Moodle installation guide [[Laf11](#)], a system with 1 GB of RAM theoretically could have supported a maximum of 50 connections.

Results

The students tested their queries in Oracle's SQL Developer and submitted their solution to the FEUP exam management system, SIGEX. Upon contest termination, the submissions were subsequently uploaded *en masse* to the CBA system to be automatically graded. A total of 219 submissions by 34 students were judged and a chart illustrating the number of correct and wrong submissions was generated and is depicted in figure 6.1.

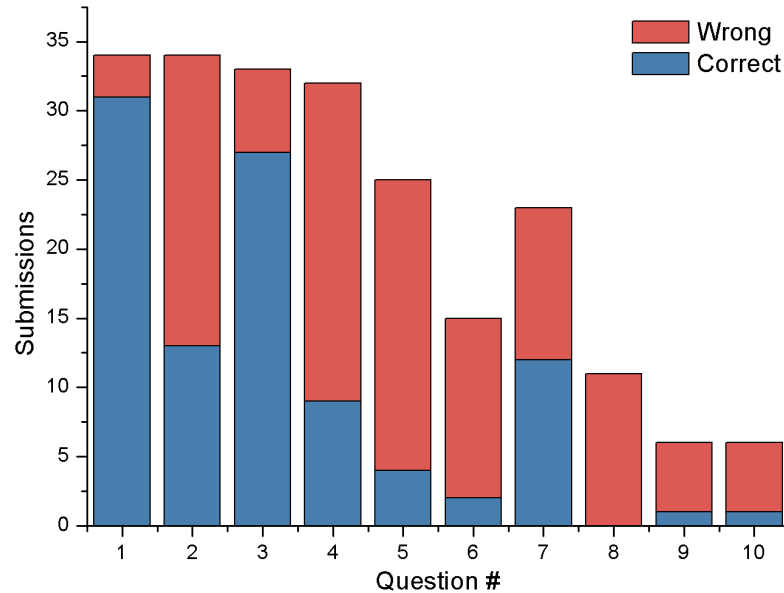


Figure 6.1: Contest submission outcome chart

Some submissions marked as incorrect were rejudged, since they presented situations which might have generated a correct solution in SQL Developer. These cases were files with multiple queries or plain text, due to SQL Developer allowing the users to compile the instruction they select in a file with multiple queries.

In a fully working test environment, the students would have been warned by the plugin feedback that their query had not compiled and the error message from the Oracle *php-oci8* driver.

6.2.2 Metric Unit Testing

Since metrics were developed as independent bash scripts, it was possible to apply unit testing for validating the correctness of the results obtained. This section exhibits a few sample C programs, alongside with the generated metric values for each one of them and a brief explanation on why they match the expected outcomes.

Test Case #1

The program #1 (figure 6.2) is short, being only a simple function call. Since there is no branching or loops involved, the cyclomatic complexity value should be 1. Also, it has 6

Results

```
1 short func(int a, int b, int c) {  
2     return a + b + c;  
3 }  
4  
5 int main() {  
6     return func(0, 0, 0);  
7 }
```

Figure 6.2: C program example #1

actual lines of code and does not incur in any style violation. These values were the same as the ones obtained in the tests, as can be seen in listing 6.1.

Listing 6.1: Test results for style and misc metrics in example #1

```
'blockBracketsSingle' => 0,  
'checkTabs' => 0,  
'programLineSize' => 0,  
'spaceBracketsCode' => 0,  
'spaceCommentsText' => 0,  
'spaceParensisBrackets' => 0,  
  
'cyclomaticComplexity' => 1,  
'linesOfCode' => 6.
```

Concerning the Halstead software science, the source contains the operands: {short, func, int, a, int, b, int, c, a, b, c, int, main, func, 0, 0, 0} for a grand total of 17 operands and 8 distinct operands.

Regarding operators, the following are present: {function, return, +, +, function, return}, totalling 6 operators, being 3 of them unique.

In order to obtain the remaining Halstead metric values, a few calculations need to be done, according to the formulas specified in chapter 3.1:

$$N_1 = 6, N_2 = 17, \eta_1 = 3, \eta_2 = 8. \quad (6.1)$$

$$N = 6 + 17 = 23. \quad (6.2)$$

$$\eta = 3 + 8 = 11. \quad (6.3)$$

$$V = 23 * \log_2(11) \approx 79. \quad (6.4)$$

Results

$$D = \frac{3}{2} * \frac{17}{8} \approx 3. \quad (6.5)$$

$$L = \frac{1}{3} \approx 0.333333. \quad (6.6)$$

$$E = 79 * 3 = 237. \quad (6.7)$$

$$T = \frac{237}{18} \approx 13. \quad (6.8)$$

$$B = \frac{237^{\frac{2}{3}}}{3000} \approx 0.012765. \quad (6.9)$$

The results from equations 6.4, 6.5, 6.8 were truncated and the results from equations 6.6, 6.9 were truncated to 6 decimal places to show the values in a more user-friendly way.

The following listing shows the obtained values for the Halstead metrics in the order they were calculated and settles this case with full marks:

Listing 6.2: Test results for Halstead metrics in example #1

```
'totalNumberOperators' => 6,  
'totalNumberOperands' => 17,  
'numberUniqueOperators' => 3,  
'numberUniqueOperands' => 8,  
'programLength' => 23,  
'vocabularySize' => 11,  
'programVolume' => 79,  
'difficultyLevel' => 3,  
'programLevel' => 0.333333,  
'effortToImplement' => 237,  
'timeToImplement' => 13,  
'numberDeliveredBugs' => 0.012765.
```

Test Case #2

The program #2, illustrated by figure 6.3, adds some complexity without violating any style rules once more. 12 lines of code and a cyclomatic number of 3 caused by the `if` and `while` instructions are the expected results for these metrics.

Since most Halstead metrics are obtained by operations with the four most important metrics and they have already been proven correct, only these will be focused.

Results

```
1 int main() {  
2  
3     static unsigned int minimal_complexity = 1;  
4  
5     if (minimal_complexity == 1)  
6         minimal_complexity--;  
7  
8     while (minimal_complexity > 1)  
9         minimal_complexity--;  
10  
11     minimal_complexity;  
12     minimal_complexity;  
13     minimal_complexity;  
14     minimal_complexity;  
15  
16     return minimal_complexity;  
17 }
```

Figure 6.3: C program example #2

Breaking the source code into operands generates the list: {int, main, unsigned, int, minimal_complexity, 1, minimal_complexity, 1, minimal_complexity, minimal_complexity, 1, minimal_complexity, minimal_complexity, minimal_complexity, minimal_complexity, minimal_complexity, minimal_complexity}, containing a total of 17 elements, being only 5 of them unique.

The operator list is shorter and can be detailed as: {function, static, =, if, ==, --, while, >, --, return}, resulting in 9 distinct operators out of a total 10.

The program output is in conformation with the predicted values, as shown in the listing 6.3.

Listing 6.3: Test results for example #2

```
'totalNumberOperators' => 10,  
'totalNumberOperands' => 17,  
'numberUniqueOperators' => 9,  
'numberUniqueOperands' => 5,  
  
'blockBracketsSingle' => 0,  
'checkTabs' => 0,  
'programLineSize' => 0,  
'spaceBracketsCode' => 0,  
'spaceCommentsText' => 0,  
'spacePentesisBrackets' => 0,  
  
'cyclomaticComplexity' => 3,  
'linesOfCode' => 12.
```

Test Case #3

```

1  int main(int argc, char* argv[]) {
2
3      // line size violation
4      double identifier_with_a_very_long_name_intended_to_cause_a_style_violation = 1.0;
5
6      //no space between comment and text
7
8      unsigned char a = 'b';
9      int b = 5;
10     float c = 1.0;
11
12     while (0){ /* brackets in a single line */ }
13
14     int i;
15     for (i = 0; i != 5; i++){ // no space between parenthesis and brackets
16
17         if (i <= b) {
18
19             c = (b + b - b) * (b / b) +
20                 (i == 5 ? 5 : 0); // tab violation
21         }
22     } i = 0; // space between brackets and code
23
24     return 0;
25 }

```

Figure 6.4: C program example #3

Finally, the last presented test case provides a more complex structure and aims to fail every style metric.

The program shown in figure 6.4 is commented to illustrate which style violations are occurring in a given line. Line 4 is 87 characters long, which surpasses the default limit of 78 characters for a single line. In line 6 there is no space between the comment declaration ("//") and the text, whereas in line 12 a closing bracket ("}") is present in the same line as an opening bracket ("{"). In line 15 there is no space between the closing parenthesis in the `for` instruction and the opening bracket. Although it is not visible, there is a tab character present in line 20, as well as no space between the closing bracket and the next instruction in line 22.

The results for the test case #3 are shown in the form they are printed by the PHP metrics library in listing A.1 and assert all covered style violations.

Listing 6.4: Full test results for example #3

```

## static analysis ##
array (
    'halstead' =>
        array (
            'difficultyLevel' => 19,

```

Results

```
'effortToImplement' => 6517,  
'numberDeliveredBugs' => 0.116299,  
'numberUniqueOperands' => 17,  
'numberUniqueOperators' => 16,  
'programLength' => 68,  
'programLevel' => 0.052631,  
'programVolume' => 343,  
'timeToImplement' => 362,  
'totalNumberOperands' => 42,  
'totalNumberOperators' => 26,  
'vocabularySize' => 33,  
,  
'style' =>  
  array (  
    'blockBracketsSingle' => 1,  
    'checkTabs' => 1,  
    'programLineSize' => 1,  
    'spaceBracketsCode' => 1,  
    'spaceCommentsText' => 1,  
    'spaceParensBrackets' => 1,  
  ),  
'misc' =>  
  array (  
    'cyclomaticComplexity' => 5,  
    'linesOfCode' => 17,  
  ),  
)
```

Although only three test cases were presented, each metric was tested in more detail. Nonetheless, it is possible to conclude from the unit tests conducted to this specific module that the values computed are accurate for simple and slightly more complex test cases.

6.2.3 Plagiarism Detection Effectiveness

In order to test the plagiarism detection algorithm implemented by Crot, a few different modifications were made to a sample C program with 80 lines of code (<http://www.cis.temple.edu/~ingargio/cis71/code/binary.c>) which aimed to test the algorithm resilience to these changes.

Results

Seven different files were submitted to the CBA system for testing (refer to appendix E for the modified files):

- **M0**: the original file with no modifications;
- **M1**: removing all comments and empty lines;
- **M2**: adding noise (comments, empty lines and statements with no effect);
- **M3**: changing variable names;
- **M4**: changing function order;
- **M5**: inversing condition logic and swapping `while` statements with `for` statements;
- **M6**: **M1 + M2 + M3 + M4 + M5**.

The results obtained spanned the double-entry table 6.1, with the most striking similarity values highlighted for each submitted file.

Table 6.1: Similarities between the modifications of the source file `binary.c`.

%	M0	M1	M2	M3	M4	M5	M6
M0	-	57.14	80.95	28.57	88.1	85.71	4.76
M1	85.47	-	67.86	10.71	82.14	67.86	7.14
M2	80.95	45.24	-	23.81	71.43	73.81	4.76
M3	24.49	6.12	20.41	-	16.33	24.49	42.86
M4	84.09	52.27	68.18	18.18	-	70.45	4.55
M5	83.72	44.19	72.09	27.91	72.09	-	4.65
M6	4.55	4.55	4.55	47.73	4.55	4.55	-

Although at a first glance the table might seem as if it should have been symmetrical, the reason because it is not is explained by how Crot calculates the similarity score percentage, which is given by the formula 6.10. Files with different size will have a different number of total hashes, causing the disparity in the values.

$$similarity_score_{(x,y)} = \frac{common_hashes_{(x,y)}}{total_hashes_{(x)}}. \quad (6.10)$$

The algorithm proved to be the least resilient against a mass variable refactoring, scoring lower marks in the similarity comparisons which involved **M3**. One of the possible reasons for **M3** and **M6** averaging as much as 45% in their resemblance is the fact of the variable naming being identical in both cases.

Removing all comments and empty lines proved to be the second most effective way to deceive the system, specially in cases with a short number of lines of code, although it is likely that better results can be achieved if the grammar size value used to calculate the fingerprints is reduced.

Finally, the last case provided the least similarities to the original file as expected, albeit a 4.55% similarity score might not be enough to induce teachers in suspecting occurrences of plagiarism. Nevertheless, for an algorithm not directly focused in source code structure comparison, the results are reasonably satisfactory. Moreover, the number of changes that needs to be done for causing the system to score very low implies a mild degree of understanding of how the code works.

6.3 Summary

The focus of this chapter resided in the accomplishment level and validation of the goals of this thesis, and how it extended the functionalities of the original system.

The main achievements were the implementation and integration of the two advanced modules covered in the state of the art study in chapter 3: static analysis of code quality and plagiarism detection. Both of these modules were independently tested and details on how to improve and extend them were provided. The development of additional metrics and the integration of more programming languages were the focus of the static analysis module. On the other hand, the plagiarism detection module blesses the system with new grading interfaces, easing the implementation of new plagiarism plugins or the improvement of the current algorithm, Crot.

Additionally, two of the advanced modules specified in the requirements were partially developed, namely the feedback level configuration and support for assessment creation.

The only module which does not reflect any actual implementation progress is the "automatic correction of project with multiple source files", although two different architecture proposals were discussed in section 4.1.2, *Assignment versus Module*.

A case study using the programming language SQL allowed to partially test the system under a real working environment, although hardware issues prevented more detailed and accurate information to be retrieved. The positive aspect of this setback was the discovery of some of the system limitations and learning a few parameters which will help to conceive a safer system specification.

In terms of innovation, besides the functionalities already covered in Pacheco's report [Pac10], the CBA system now supports an additional programming language, SQL, that had been currently unheard of or almost nonexistent in similar systems. Moreover, it integrates advanced modules such as static analysis and plagiarism detection, which may not be new in other CBA systems, but contributes in the number of features and to the growing into a more mature marking tool relying solely in open-source software.

Chapter 7

Conclusions and Future Work

CBA systems have been integrated in an increasingly number of universities. This phenomenon is due to the effectiveness of this systems in greatly easing the teachers' job and allowing them to devote more time to each student individually. Some CBA systems were developed with the intent of supporting a wide array of functionalities, proving to be an invaluable tool both for helping students in learning programming and aiding the instructors in grading, making assessments and ultimately detecting frauds.

Facing the rising success and accuracy of these platforms, it a CBA system was implemented in order to provide the basic functionalities to be adopted in the programming courses of the Master in Informatics and Computing Engineering.

7.1 Work Summary

The goals of this dissertation were to compile a study on advanced functionalities for CBA systems and implement a set of modules in an existing platform of this type, in order to be ready to play an active role on the courses of DEI ¹. The system, with the contribution of the advanced modules, is able to help reducing the amount of work needed for marking and grading programming assignments, by automating and adding additional parameters to the assessment process. The work developed in this thesis should be flexible enough to be extended or integrated in any other CBA platform. In order to achieve the proposed goals, the following steps were taken:

- A comparison between the features of eight state-of-the-art CBA systems and the base CBA system was performed, with the goal of understanding the state of development of the prototype implemented by Pacheco [Pac10]. Moreover, the architecture of the system was detailed and the limitations of the current implementation

¹Departamento de Engenharia Informática (Informatics Engineering Department)

were identified as the main objectives to be fulfilled in this thesis;

- A detailed analysis of the state of the art of two advanced modules: static analysis of code quality and plagiarism detection. This study covered the approaches taken by five different CBA systems, focusing on the most used metrics in these platforms as well as a comparison between three distinct types of plagiarism detection algorithms;
- Specification of the advanced modules which can be integrated in CBA systems, covering the possible implementation options and how to add them to the previously developed CBA system, in order to be used in programming course units at FEUP. The system requirements were exposed and a new architecture bearing in mind the new modules was proposed;
- Upon setting up the prototype in four virtual machines running Debian GNU/Linux, the system was extended to support the programming language SQL and a case study with the CBA platform was done in the course unit Databases of LCI. The static analysis module was implemented, unit-tested and integrated, as well as the plagiarism detection module;
- Finally, the results achieved by the project were assessed. Out of the 5 advanced modules initially proposed, 2 were fully implemented and 2 were partially implemented. It was possible to successfully define the extensibility degree of each module alongside with the guidelines to improve them. Moreover, a report was compiled on the case study conducted, providing important information on the system hardware limitations.

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 A** - a guide with some tips for installing the implemented prototype;
- **Appendix B** - a step-by-step guide for configuring a new programming language in the CBA system, with special detail regarding the SQL language;
- **Appendix C** - provides instructions on how to configure the static analysis module developed in the CBA system;
- **Appendix D** - provides instructions on how to configure the plagiarism detection module developed in the CBA system;
- **Appendix E** - a list of the source code files used for testing purposes;
- **Appendix F** - a guide on how to create and configure a new programming assessment using the advanced functionalities development;

- **Appendix G** - a description of the front-end web services.

7.2 Future Work

Despite having a working CBA platform which implements a significant amount of features and a few advanced modules, the system, although in a functional state, could benefit from more testing. Furthermore, the following improvements would be an added value to enhance the whole educational process:

- Implement the only module which was remaining, the automatic correction of projects with multiple source files. As the problem complexity grows, students start using more than one file and it would be interesting to support the uploading of multiple files, which would be easier if the Programming Assessment module extended the Assignment activity;
- The static analysis module would benefit from an additional number of metrics and supported programming languages. Since this development process is continuous, it might mean that at least one person should be allocated to the CBA system maintenance.
- Fine-tuning the plagiarism detection algorithm, integrating a different or additional plagiarism plugin in Moodle or developing a new one from scratch. The CBA system now fully supports the Moodle Plagiarism API, meaning it will support the plugins developed by the community in the future;
- Improving the usability and the way that the feedback is shown to students. The user experience would be better if students did not have to refresh the page to check if they have already been graded and if they could check exactly the results obtained in each metric. An alternative to this measure would be implementing a notification system which would alert the student as soon as his submission is graded.

Conclusions and Future Work

References

- [Aik10] Alex Aiken. Plagiarism detection. <http://theory.stanford.edu/~aiken/moss/>, July 2010.
- [BGNM04] Michael Blumenstein, Steve Green, Ann Nguyen, and Vallipuram Muthukkumarasamy. GAME: A generic automated marking environment for programming assessment. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2 - Volume 2*, ITCC '04, pages 212–, Washington, DC, USA, 2004. IEEE Computer Society.
- [BS09] Sergey Butakov and Vladislav Scherbinin. The toolbox for local and global plagiarism detection. *Computers & Education*, 52(4):781 – 788, 2009.
- [But11] Sergei Butakov. Moodle.org: Modules and plugins. <http://moodle.org/mod/data/view.php?id=13&rid=4655>, March 2011.
- [CK10] Daniela Chudá and Bianka Kováčová. Checking plagiarism in e-learning. In *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*, CompSysTech '10, pages 419–424, New York, NY, USA, 2010. ACM.
- [CSU08] Sacramento California State University. International collegiate programming contest validator interface standard. <http://www.ecs.csus.edu/pc2/doc/valistandard.html>, August 2008.
- [Dal99] Charlie Daly. RoboProf and an introductory computer programming course. *SIGCSE Bull.*, 31:155–158, June 1999.
- [DH05] Charlie Daly and Jane Horgan. Patterns of plagiarism. *SIGCSE Bull.*, 37:383–387, February 2005.
- [Dou11] Martin Dougiamas. Moodle version history - moodledocs. http://docs.moodle.org/en/Moodle_version_history, May 2011.
- [DW95] Michael Wise Department and Michael J. Wise. Neweyes: A system for comparing biological sequences using the running karp-rabin greedy string-tiling algorithm. In *Third International Conference on Intelligent Systems for Molecular Biology*, pages 393–401. AAAI Press, 1995.
- [EKW10] Jaap Eldering, Thijs Kinkhorst, and Peter van de Warken. *DOMjudge Administrator's Manual*, 2010.

REFERENCES

- [EKW11a] Jaap Eldering, Thijs Kinkhorst, and Peter van de Warken. DOMjudge - programming contest jury system. <http://domjudge.sourceforge.net/>, January 2011.
- [EKW11b] Jaap Eldering, Thijs Kinkhorst, and Peter van de Warken. *DOMjudge Jury Manual*, 2011.
- [Fan10] Dominic Fandrey. Clang/llvm maturity report. Moltkestr. 30, 76133 Karlsruhe - Germany, June 2010. See <http://www.iwi.hs-karlsruhe.de>.
- [Gmb10a] Verifysoft Technology GmbH. Verifysoft -> Halstead metrics. http://www.verifysoft.com/en_halstead_metrics.html, June 2010.
- [Gmb10b] Verifysoft Technology GmbH. Verifysoft -> McCabe metrics. http://www.verifysoft.com/en_mccabe_metrics.html, June 2010.
- [Gru09] Dick Grune. The software and text similarity tester SIM. <http://www.cs.vu.nl/~dick/sim.html>, January 2009.
- [GT99] David Gitchell and Nicholas Tran. SIM: a utility for detecting similarity in computer programs. *SIGCSE Bull.*, 31:266–270, March 1999.
- [GVN02] Moumita Ghosh, Brijesh Kumar Verma, and Anne T. Nguyen. An automatic assessment marking and plagiarism detection system. 2002.
- [HGST05] Colin A. Higgins, Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintsifas. Automated assessment and experiences of teaching programming. *J. Educ. Resour. Comput.*, 5, September 2005.
- [HLVW02] Bodo Husemann, Jens Lechtenbörger, Gottfried Vossen, and Peter Westerkamp. XLX - a platform for graduate-level exercises. In *Proceedings of the International Conference on Computers in Education, ICCE '02*, pages 1262–, Washington, DC, USA, 2002. IEEE Computer Society.
- [JGB05] Mike Joy, Nathan Griffiths, and Russell Boyatt. The BOSS online submission and assessment system. *J. Educ. Resour. Comput.*, 5, September 2005.
- [JL99] M. Joy and M. Luck. Plagiarism in programming assignments. *Education, IEEE Transactions on*, 42(2):129–133, may 1999.
- [JM10] Karl O. Jones and T. Anthony Moore. Turnitin is not the primary weapon in the campaign against plagiarism. In *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*, CompSysTech '10, pages 425–429, New York, NY, USA, 2010. ACM.
- [Jon08] Karl O. Jones. Practical issues for academics using the turnitin plagiarism detection software. In *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, CompSysTech '08, pages 50:IV.1–50:1, New York, NY, USA, 2008. ACM.

REFERENCES

- [Jon11] Christopher Jones. PHP Oracle FAQ - Oracle wiki. <http://wiki.oracle.com/page/PHP+Oracle+FAQ>, March 2011.
- [KST⁺86] Joseph P. Kearney, Robert L. Sedlmeyer, William B. Thompson, Michael A. Gray, and Michael A. Adler. Software complexity measurement. *Commun. ACM*, 29:1044–1050, November 1986.
- [Laf11] Eloy Lafuente. Installing moodle - moodledocs. http://docs.moodle.org/20/en/Installing_Moodle, March 2011.
- [LC03] Thomas Lancaster and Fintan Culwin. Classifications of plagiarism detection engines. <http://www.ics.heacademy.ac.uk/italics/Vol4-2/Plagiarism%20-%20revised%20paper.htm>, 2003.
- [LGG07] Romans Lukashenko, Vita Graudina, and Janis Grundspenkis. Computer-based plagiarism detection methods and tools: an overview. In *Proceedings of the 2007 international conference on Computer systems and technologies*, CompSysTech '07, pages 40:1–40:6, New York, NY, USA, 2007. ACM.
- [LLL08] Rüdiger Lincke, Jonas Lundberg, and Welf Löwe. Comparing software metrics tools. In *Proceedings of the 2008 international symposium on Software testing and analysis*, ISSTA '08, pages 131–142, New York, NY, USA, 2008. ACM.
- [LS03] José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Softw. Pract. Exper.*, 33:567–581, May 2003.
- [Mar11a] Dan Marsden. Development:plagiarism API - moodledocs. http://docs.moodle.org/en/Development:Plagiarism_API, May 2011.
- [Mar11b] Dan Marsden. Plagiarism prevention turnitin - moodledocs. http://docs.moodle.org/en/Plagiarism_Prevention_Turnitin, May 2011.
- [MBG07] Roozbeh Matloobi, Michael Myer Blumenstein, and Steven Green. An enhanced generic automated marking environment: Game-2. 2007.
- [MBG09] Roozbeh Matloobi, Michael Myer Blumenstein, and Steven Green. Extensions to generic automated marking environment: Game-2+. 2009.
- [McC76] Thomas J. McCabe. A complexity measure. In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.
- [MFW⁺05] Maxim Mozgovoy, Kimmo Fredriksson, Daniel White, Mike Joy, and Erkki Sutinen. Fast plagiarism detection system. In *Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE2005)*, Buenos Aires, Argentina, November 2005 (Lecture Notes in Computer Science, pages 267–270. Springer, 2005.

REFERENCES

- [MKK07] M. Mozgovoy, S. Karakovskiy, and V. Klyuev. Fast and reliable plagiarism detection system. In *Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07. 37th Annual*, pages S4H-11 –S4H-14, 2007.
- [Moz06] Maxim Mozgovoy. Desktop tools for offline plagiarism detection in computer programs. *Informatics in education*, 5:97–112, January 2006.
- [Moz08] Maxim Mozgovoy. *Enhancing Computer-Aided Plagiarism Detection*. VDM Verlag, Saarbrücken, Germany, Germany, 2008.
- [MY99] Susan A. Mengel and Vinay Yerramilli. A case study of the static analysis of the quality of novice student programs. *SIGCSE Bull.*, 31:78–82, March 1999.
- [Pac10] Pedro Pacheco. Computer-based assessment system for e-learning applied to programming education. Master’s thesis, 2010.
- [PMP00] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. JPlag: Finding plagiarisms among a set of programs. Technical report, 2000.
- [PRS⁺03] Yusuf Pisan, Debbie Richards, Anthony Sloane, Helena Koncek, and Simon Mitchell. Submit! a web-based system for automatic program critiquing. In *Proceedings of the fifth Australasian conference on Computing education - Volume 20, ACE '03*, pages 59–68, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.
- [RMC⁺10] Guido Rößling, Myles McNally, Pierluigi Crescenzi, Atanas Radenski, Petri Ihantola, and M. Gloria Sánchez-Torrubia. Adapting moodle to better support cs education. In *Proceedings of the 2010 ITiCSE working group reports on Working group reports*, ITiCSE-WGR '10, pages 15–27, New York, NY, USA, 2010. ACM.
- [RS86] K. A. Redish and W. F. Smyth. Program style analysis: a natural by-product of program compilation. *Commun. ACM*, 29:126–133, February 1986.
- [SWA03] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 76–85, New York, NY, USA, 2003. ACM.
- [Tsi02] Athanasios Tsintsifas. *A Framework for the Computer Based Assessment of Diagram-Based Coursework*. PhD thesis, March 2002.
- [Wha90] Geoff Whale. Software metrics and plagiarism detection. *J. Syst. Softw.*, 13:131–138, October 1990.
- [Wig02] Joseph A. Wigner. Evaluating student programs by metric analysis. Master’s thesis, 2002.
- [Wik11] Wikipedia. Cyclomatic complexity - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Cyclomatic_complexity, June 2011.

REFERENCES

- [Wis96] Michael J. Wise. YAP3: improved detection of similarities in computer program and other texts. *SIGCSE Bull.*, 28:130–134, March 1996.
- [Wis05] Michael Wise. Plagiarism detection. <http://paml.bcs.uwa.edu.au/~michaelw/YAP.html>, February 2005.
- [WMW96] Arthur H. Watson, Thomas J. McCabe, and Dolores R. Wallace. Special publication 500-235, structured testing: A software testing methodology using the cyclomatic complexity metric. In *U.S. Department of Commerce/National Institute of Standards and Technology*, 1996.
- [WSWM07] Tiantian Wang, Xiaohong Su, Yuying Wang, and Peijun Ma. Semantic similarity-based grading of student programs. *Inf. Softw. Technol.*, 49:99–107, February 2007.

REFERENCES

Appendix A

Installation Guide

This appendix contains a guide with some useful tips for installing the base CBA. These tips are based on the experience of installing the system on a test server running on Debian GNU/Linux 6.0 (squeeze) and it is recommended to have some experience in the configuration and installation of Unix applications as well as some expertise in MySQL before proceeding.

Furthermore, all of the developed code is stored in different git repositories, so installing this software in every server is advised (`sudo apt-get install git`).

The appendix is divided in three sections, one for each physical node identified in the system 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.

Further details on how to install the advanced modules can be found on appendixes [C](#) and [D](#).

A.1 Configuring the Main Automatic Assessment Server

In order to configure the main Automatic Assessment Server, it is first needed to install the main server of DOMjudge architecture, DOMserver. To do so, it is recommended to follow attentively the instructions of the administrator's manual [\[EKW10\]](#), which are very thorough. In the first place, it is necessary to have all the listed software requirements installed. Since the DOMserver runs on Apache and MySQL, XAMPP can be used, as suggested by Pacheco in his similar guide [\[Pac10\]](#).

After the DOMserver is installed it is highly recommended to install phpMyAdmin as the database interface. Then, all the default information in the *contest* table should be removed and a single entry added, as shown in figure [A.1](#).

	cid	contestname	activatetime	starttime	freezetime	endtime	unfreezetime	enabled
Contest ID	Descriptive name	Time contest becomes visible in team/public views	Time contest starts, submissions accepted	Time scoreboard is frozen	Time after which no more submissions are accepted	Unfreeze a frozen scoreboard at this time	Whether this contest can be active	
	1	progassessment	2000-01-01 00:00:00	2000-01-01 00:00:00	NULL	2100-01-01 00:00:00	NULL	1

Figure A.1: Adding an entry to the table *contest*

Finally, the front-end component of the server can be retrieved from the repository [git://github.com/jcxavier/domserver.git](https://github.com/jcxavier/domserver.git) under the folder `domjudge/-domserver/www/frontend`. This folder should be added to the DOMserver following a similar directory path. If someone desires to change the contest name, it has to be updated every time in the line 3 of `frontend.php`:

Listing A.1: Changing the contest name

```
3 define('CONTEST', ' "progassessment"');
```

A.2 Installing a Judgehost

It is also strongly recommended to follow the instructions of DOMjudge's administrator manual [EKW10] 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.

  			hostname Resolvable hostname of judgehost	active Should this host take on judgings?	polltime Time of last poll by autojudge
<input type="checkbox"/>			domjudge1	1	2011-06-18 18:58:42
<input type="checkbox"/>			domjudge2	1	2011-06-18 18:58:42

Figure A.2: Adding an entry to the table *judgehost*

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 A.2 presents the example of two entries for Judgehosts running on machines named *domjudge1* and *domjudge2*.

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.

A.3 Installing the Moodle Server

Since the CBA system was developed for Moodle 2.0.2, this or a later version should be downloaded. The main requirements are Apache and MySQL, being phpMyAdmin an important plus for development. Instructions for installing Moodle and Moodle plugins can be found in its official site, <http://moodle.org/>.

The source code of the core Programming Assessment activity should be cloned to the `mod` folder under the name `progassessment` from the git repository [git@github.com:jcxavier/moodle-mod_progassessment.git](https://github.com/jcxavier/moodle-mod_progassessment.git).

Appendix B

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 [EKW10], 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.

The present appendix contains a step-by-step guide for configuring a new programming language, PLT Scheme. This involves not only configuring the language in DOMjudge but also configuring it in the Moodle plugin. Additionally, it explains how to further configure the system to support SQL as a programming language.

B.1 DOMjudge Configuration

1. Install an appropriate compiler, by running the command `sudo apt-get install plt-scheme` (in a Debian GNU/Linux environment). This will install MzScheme, which can be used to generate executables from Scheme source code;
2. Create a new entry in the database in the table *language*, as illustrated in figure B.1;

			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
<input type="checkbox"/>			c	C	c	1	1	1
<input type="checkbox"/>			cpp	C++	cpp	1	1	1
<input type="checkbox"/>			cs	C#	cs	1	1	1
<input type="checkbox"/>			java	Java	java	1	1	1.5
<input type="checkbox"/>			python	Python	py	1	1	1
<input type="checkbox"/>			scheme	Scheme	scm	1	1	5
<input type="checkbox"/>			sql	SQL	sql	1	1	1

Figure B.1: Adding an entry to the table *language*

3. Edit 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, as shown in listing B.1;

4. Create the compilation script (listing B.2).

Listing B.1: domserver-config.php

```
47 define('LANG_EXTs', 'C,c C++,cpp,cc,c++ Java,java Pascal,pas,p
    Haskell,hs,lhs Perl,pl Bash,sh C#,cs AWK,awk Python,py
    Scheme,scm SQL,sql');
```

Listing B.2: compile_scheme.sh

```
1 #!/bin/sh
2
3 # Scheme compile wrapper-script for 'compile.sh'.
4 # See that script for syntax and more info.
5
6 SOURCE="$1"
7 DEST="$2"
8
9 mzc --exe $DEST $SOURCE
10 exit $?
```

B.2 Moodle Configuration

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 B.3: languages_config.php

```
1 <?php
2
3 $progassessment_languages_comments = array(
4     "cpp" => "//",
5     "c" => "//",
6     "cs" => "//",
7     "java" => "//",
8     "scheme" => ";",
9     "sql" => "--"
10 );
11
12 define('STUDENT_CODE_IDENTIFIER', "studentcode");
13
14 ?>
```

B.3 Adding Support to SQL

Allowing the CBA system to grade SQL programming exercises is entirely optional. If this language is to be added to the system, the aforementioned steps have to be followed in a similar fashion except for the compiler installation, which is replaced by the Oracle Call Interface driver installation, explained in this section.

B.3.1 Installation of the Oracle Call Interface Driver

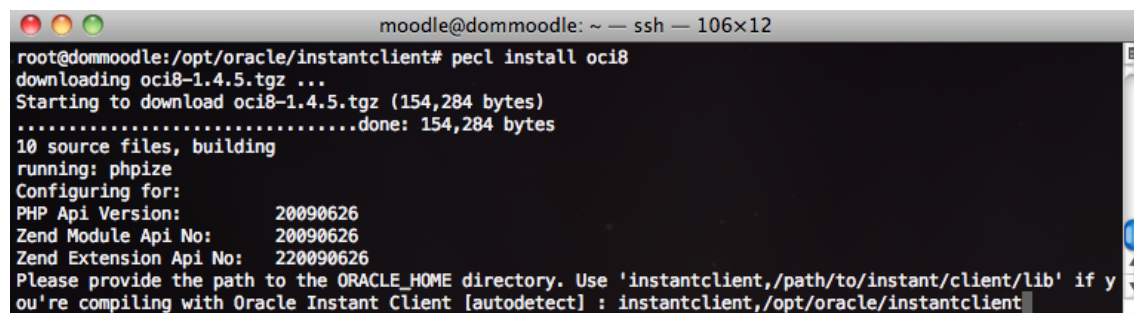
Firstly, it is needed to install the Oracle Call Interface (OCI) driver for PHP in every Judgehost.

To achieve this, it is necessary an Oracle account (which is free) to download the Instant Client Package - Basic and the Instant Client Package - SDK for Linux x86-64 at <http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html>. Then, assuming the version of the downloaded files is 11.2.0.2.0 and they are in the current folder, the full installation guide, step-by-step, is given by the listing below:

Listing B.4: Instructions for configuring the *oci8-php* extension

```
sudo su
apt-get install php5-dev php-pear libaio-dev unzip
unzip instantclient-basic-linux-x86-64-11.2.0.2.0.zip
unzip instantclient-sdk-linux-x86-64-11.2.0.2.0.zip
mkdir -p /opt/oracle/instantclient
mv instantclient_11_2/* /opt/oracle/instantclient
cd /opt/oracle/instantclient
echo /opt/oracle/instantclient >> /etc/ld.so.conf
ldconfig
ln -s libclntsh.so.11.1 libclntsh.so
ln -s libocci.so.11.1 libocci.so
pecl install oci8
```

When prompted for the Oracle Instant Client installation path, insert, following the example of figure B.2: "instantclient,/opt/oracle/instantclient".



```
moodle@dommoodle: ~ — ssh — 106x12
root@dommoodle:/opt/oracle/instantclient# pecl install oci8
downloading oci8-1.4.5.tgz ...
Starting to download oci8-1.4.5.tgz (154,284 bytes)
.....done: 154,284 bytes
10 source files, building
running: phpize
Configuring for:
PHP Api Version:      20090626
Zend Module Api No:   20090626
Zend Extension Api No: 220090626
Please provide the path to the ORACLE_HOME directory. Use 'instantclient,/path/to/instant/client/lib' if y
ou're compiling with Oracle Instant Client [autodetect] : instantclient,/opt/oracle/instantclient
```

Figure B.2: Installing *oci8* using *pecl*

To successfully finish the installation, insert the line "extension=oci8.so" near Dynamic Extensions (figure B.3) in the files:

- `/etc/php5/cli/php.ini;`
- `/etc/php5/apache2/php.ini.`

Finally, the command `/etc/init.d/apache2 restart` should be executed if Apache is installed in the system.

```

917 ;;;;;;;;;;;;;;;;;;
918 ; Dynamic Extensions ;
919 ;;;;;;;;;;;;;;;;;;
920
921 ; If you wish to have an extension loaded automatically, use the following
922 ; syntax:
923 ;
924 ;   extension=modulename.extension
925 ;
926 ; For example, on Windows:
927 ;
928 ;   extension=msql.dll
929 ;
930 ; ... or under UNIX:
931 ;
932 ;   extension=msql.so
933 ;
934 ; ... or with a path:
935 ;
936 ;   extension=/path/to/extension/msql.so
937 ;
938 ; If you only provide the name of the extension, PHP will look for it in its
939 ; default extension directory.
940 ;
941
942 extension=oci8.so
943

```

Figure B.3: Adding the *oci8* extension in the configuration files

B.3.2 Adding the Oracle Scripts to the Judgehosts

The last step is copying the judging scripts to the `lib/judge` folder of each Judgehost. The files which need to be copied can be found in the git repository [git@github.com: jcxavier/judgehost.git](https://github.com/jcxavier/judgehost) and are listed as follows:

- `check_oracle.sh;`
- `compare_oracle.sh;`
- `compile_sql.sh;`
- `oci_lib.php;`
- `run_oracle.sh;`
- `runjury_oracle.sh.`

Appendix C

Configuration of the Static Analysis Module

In order to configure the static analysis module using the C programming language in the system, one needs to install the following additional software in all Judgehost machines: Clang 2.7, *XGrep*. Step-by-step instructions can be found below on how to successfully configure the module. The guides assume the user is in the *sudoers* list and has permission to install software.

Listing C.1: Instructions for configuring Clang

```
cd ~
wget http://llvm.org/releases/2.7/llvm-2.7.tgz
tar xvfz llvm-2.7.tgz
cd llvm-2.7
wget http://llvm.org/releases/2.7/clang-2.7.tgz
tar xvfz clang-2.7.tgz
mv clang-2.7 clang
./configure && make
sudo make install
```

Listing C.2: Instructions for configuring *XGrep*

```
sudo apt-get install bzip2 xorg xorg-dev libxml2-dev
cd ~
wget http://xorg.freedesktop.org/releases/individual/util/
    makedepend-1.0.3.tar.bz2
wget http://wohlberg.net/public/software/xml/xgrep/xgrep-0.07.
    tar.gz
tar jxf makedepend-1.0.3.tar.bz2
tar xvfz xgrep-0.07.tar.gz
cd ~/makedepend-1.0.3.tar.bz2
./configure && make
sudo make install
cd ~/xgrep-0.07
```

Configuration of the Static Analysis Module

```
./configure && make
sudo make install
```

Once the software is configured, the static analysis repository (<https://github.com/jcxavier/sa>) needs to be added as `static` to the `lib` folder. The command to do this would be `git clone git@github.com:jcxavier/sa.git static`, although other option would be adding this git repository as a submodule, assuming the Judgehost is under git version control.

After that, to add support to any programming language, the compiling scripts under `lib/judge` need to be changed according to the following example:

Listing C.3: `compile_c.sh`

```
1  #!/bin/sh
2
3  # C compile wrapper-script for 'compile.sh'.
4  # See that script for syntax and more info.
5
6  SOURCE="$1"
7  DEST="$2"
8
9  # -Wall:          Report all warnings
10 # -O2:            Level 2 optimizations (default for speed)
11 # -static:        Static link with all libraries
12 # -pipe:          Use pipes for communication between stages of
   compilation
13 # -lm:            Link with math-library (has to be last
   argument!)
14 gcc -Wall -O2 -static -pipe -o $DEST $SOURCE -lm
15 EXITCODE=$?
16
17 # static analysis
18 cp -r /home/domjudge-run/domjudge/judgehost/lib/static/* .
19 php metricslib.php $SOURCE
20
21 exit $EXITCODE
```

Appendix D

Configuration of the Plagiarism Detection Module

Installing the Crot plugin is simple, requiring only the cloning of the altered version of Crot repository to the plagiarism plugins folder of Moodle:

```
git clone git@github.com:jcxavier/moodle-plagiarism_crot.git
moodle/plagiarism/crot.
```

If the global search feature of Crot is to be used, it is needed to obtain the MS-Application ID key from the Microsoft. This can be achieved by navigating to <http://www.bing.com/developers> and following the instructions to get the Application ID.

The user needs then to follow to remaining steps to install this plugin (abridged from the original instructions):

1. Log in to Moodle as an admin. Plugins check page will be opened;
2. Click on the Upgrade button at the bottom of the page. Moodle will setup the required tables;
3. Proceed to Advanced features in the admin's menu and check "Enable plagiarism plugins" option. Save changes;
4. Open Plugins / Plagiarism prevention / Crot link from the admin's menu;
5. Check "Enable Crot" and put the MS-Application ID key in the appropriate text box. Save changes;
6. Other settings can be changed later based on the experience with the plugin. The complete description of these settings is as follows:
 - Student disclosure: This text will appear on the assignment submission page when student work will be uploaded to the system;
 - Grammar size is the size of text used to calculate one hash in document fingerprint. Recommended value is 30;

Configuration of the Plagiarism Detection Module

- Window size represents how granular the text search will be. Recommended value is 60;
 - Colours are used for highlighting of similar text fragments in the side by side comparison of documents. But at the moment the colouring function works only with the one colour (#FF0000);
 - Maximum distance between hashes in the text cluster: This parameter is used for allocation of hashes into text clusters in the colouring of similar text fragments. Recommended value is 100;
 - Minimum cluster size is a minimum number of hashes in the text cluster used for colouring of similar text fragments. Recommended value is 2;
 - Default threshold: Assignments with similarity score less than threshold value are not displayed on Anti-Plagiarism - Assignments page;
 - Global search threshold reserved for future development. Recommended value is 90;
 - Global search query size is the number of words in the query for global search. Recommended value is 7;
 - Percentage of search queries for Web search is randomly selected percentage of queries from all search queries for Web search. Recommended value is 40;
 - Number of web documents to be downloaded: How many web documents will be downloaded to the Moodle server from the list of possible sources on the web. Recommended value is 10;
 - Culture info for global search: Culture information is used in queries for Bing search engine;
 - Maximum file size: Files with the size more than this value are not downloaded from the internet. Recommended value is 1000000.
7. Select "Test global search" to run a quick test of global search. If it works the results of test search query will be seen on the next step;
 8. Click the Save Changes button.

Appendix E

Tests

This appendix contains the files used for the testing of the advanced modules mentioned in chapter 6. A total of 3 files were listed for static analysis and 7 for plagiarism detection.

E.1 Files Used for Metric Unit Testing

Listing E.1: test1.c

```
1 short func(int a, int b, int c) {  
2     return a + b + c;  
3 }  
4  
5 int main() {  
6     return func(0, 0, 0);  
7 }
```

Listing E.2: test2.c

```
1 int main() {  
2  
3     static unsigned int minimal_complexity = 1;  
4  
5     if (minimal_complexity == 1)  
6         minimal_complexity--;  
7  
8     while (minimal_complexity > 1)  
9         minimal_complexity--;  
10  
11     minimal_complexity;  
12     minimal_complexity;  
13     minimal_complexity;  
14     minimal_complexity;  
15 }
```

```

16     return minimal_complexity;
17 }

```

Listing E.3: test3.c

```

1 int main(int argc, char* argv[]) {
2
3     // line size violation
4     double identifier_with_a_very_long_name_intended_to_
        cause_a_style_violation = 1.0;
5
6     //no space between comment and text
7
8     unsigned char a = 'b';
9     int b = 5;
10    float c = 1.0;
11
12    while (0){ /* brackets in a single line */ }
13
14    int i;
15    for (i = 0; i != 5; i++){ // no space between parenthesis
        and brackets
16
17        if (i <= b) {
18
19            c = (b + b - b) * (b / b) +
20                (i == 5 ? 5 : 0); // tab violation
21        }
22    }i = 0; // space between brackets and code
23
24    return 0;
25 }

```

E.2 Submitted Files for Plagiarism Detection

Listing E.4: M0.c

```

1 /* binary.c - Binary search using two methods. The first is
    more intuitive, but it is
2     slower.
3 */
4
5 #include <stdio.h>
6 #include <sys/time.h>
7
8 /* Given a sorted array with n elements, we search for who
    using binary search.

```

```

9      We return a position where found, or -1 if not there
10 */
11 int binary1(int n, int a[n], int who) {
12     int left = 0;
13     int right = n-1;
14     while (left <= right) {
15         int mid = left + (right-left)/2;
16         if (who < a[mid])
17             right = mid - 1;
18         else if (who > a[mid])
19             left = mid + 1;
20         else
21             return mid;
22     }
23     return -1;
24 }
25
26 /* Given a sorted array with n elements, we search for who
27 using binary search.
28 We return a position where found, or -1 if not there
29 */
30 int binary2(int n, int a[n], int who) {
31     int p = n/2;
32     while (n > 0) {
33         n = n/2;
34         if (who < a[p]) {
35             p -= n;
36         } else if (who > a[p]) {
37             p += n;
38         } else
39             return p;
40     }
41     return -1;
42 }
43
44 /* Returns the difference in microseconds between before and
45 after */
46 long timediff(struct timeval before, struct timeval after) {
47     long sec = after.tv_sec - before.tv_sec;
48     long microsec = after.tv_usec - before.tv_usec;
49     return 1000000*sec + microsec;
50 }
51
52 int main() {
53     int a[] = {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
54     int n = sizeof(a)/sizeof(int);
55     int where;

```

Tests

```

54     struct timeval before;
55     struct timeval after;
56     int k;
57     int j;
58     gettimeofday(&before, NULL);
59     for (j = 0; j < 1000000; j++)
60     for (k = 0; k < 2*n+1; k++) {
61         where = binary1(n, a, k);
62 //     printf("who = %d, \tvalue = %d\n", k, where);
63     }
64     gettimeofday(&after, NULL);
65     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
        tv_sec, before.tv_usec,
66             after.tv_sec, after.tv_usec);
67     printf("The difference is %ld\n", timediff(before, after))
        ;
68     printf("-----\n");
69     gettimeofday(&before, NULL);
70     for (j = 0; j < 1000000; j++)
71     for (k = 0; k < 2*n+1; k++) {
72         where = binary2(n, a, k);
73 //     printf("who = %d, \tvalue = %d\n", k, where);
74     }
75     gettimeofday(&after, NULL);
76     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
        tv_sec, before.tv_usec,
77             after.tv_sec, after.tv_usec);
78     printf("The difference is %ld\n", timediff(before, after))
        ;
79     return 0;
80 }

```

Listing E.5: M1.c

```

1  #include <stdio.h>
2  #include <sys/time.h>
3  int binary1(int n, int a[n], int who) {
4      int left = 0;
5      int right = n-1;
6      while (left <= right) {
7          int mid = left + (right-left)/2;
8          if (who < a[mid])
9              right = mid - 1;
10         else if (who > a[mid])
11             left = mid + 1;
12         else
13             return mid;
14     }

```



```

15     return -1;
16 }
17 int binary2(int n, int a[n], int who) {
18     int p = n/2;
19     while (n > 0) {
20         n = n/2;
21         if (who < a[p]) {
22             p -= n;
23         } else if (who > a[p]) {
24             p += n;
25         } else
26             return p;
27     }
28     return -1;
29 }
30 long timediff(struct timeval before, struct timeval after) {
31     long sec = after.tv_sec - before.tv_sec;
32     long microsec = after.tv_usec - before.tv_usec;
33     return 1000000*sec + microsec;
34 }
35 int main() {
36     int a[] = {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
37     int n = sizeof(a)/sizeof(int);
38     int where;
39     struct timeval before;
40     struct timeval after;
41     int k;
42     int j;
43     gettimeofday(&before, NULL);
44     for (j = 0; j < 1000000; j++)
45         for (k = 0; k < 2*n+1; k++) {
46             where = binary1(n, a, k);
47         }
48     gettimeofday(&after, NULL);
49     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
        tv_sec, before.tv_usec,
50         after.tv_sec, after.tv_usec);
51     printf("The difference is %ld\n", timediff(before, after))
        ;
52     printf("-----\n");
53     gettimeofday(&before, NULL);
54     for (j = 0; j < 1000000; j++)
55         for (k = 0; k < 2*n+1; k++) {
56             where = binary2(n, a, k);
57         }
58     gettimeofday(&after, NULL);

```

Tests

```
59     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.  
        tv_sec, before.tv_usec,  
60             after.tv_sec, after.tv_usec);  
61     printf("The difference is %ld\n", timediff(before, after))  
        ;  
62     return 0;  
63 }
```

Listing E.6: M2.c

```
1  /* binary.c - Binary search using two methods. The first is  
   more intuitive, but it is  
2     slower.  
3  */  
4  
5  
6  
7  #include <stdio.h>  
8  
9  // comment  
10 #include <sys/time.h>  
11  
12 /* Given a sorted array with n elements, we search for who  
   using binary search.  
13   We return a position where found, or -1 if not there  
14  */  
15 int binary1(int n, int a[n], int who) {  
16     int left = 0;  
17     int right = n-1;  
18     while (left <= right) {  
19  
20         // random comment  
21  
22         int mid = left + (right-left)/2;  
23         if (who < a[mid])  
24             right = mid - 1;  
25  
26  
27  
28  
29         else if (who > a[mid])  
30             left = mid + 1;  
31         else  
32             return mid;  
33     }  
34     return -1;  
35 }  
36
```

Tests

```
37  /* Given a sorted array with n elements, we search for who
    using binary search.
38  We return a position where found, or -1 if not there
39  */
40  int binary2(int n, int a[n], int who) {
41      int p = n/2;
42      while (n > 0) {
43          n = n/2;
44          if (who < a[p]) {
45              p -= n;
46
47              /* this one is a bigger comment spanning
48              * multiple
49              * lines */
50
51
52          } else if (who > a[p]) {
53              p += n;
54          } else
55
56              // add
57
58              return p;
59      }
60      return -1;
61  }
62
63  /* Returns the difference in microseconds between before and
    after */
64  long timediff(struct timeval before, struct timeval after) {
65      long sec = after.tv_sec - before.tv_sec;
66
67      sec = sec; // noise
68
69      long microsec = after.tv_usec - before.tv_usec;
70      return 1000000*sec + microsec;
71  }
72
73  int main() {
74      int a[] = {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
75      int n = sizeof(a)/sizeof(int);
76      int where;
77
78      // comment
79      struct timeval before;
80
81      struct timeval after;
```

Tests

```

82     int k;
83     int j;
84
85
86     gettimeofday(&before, NULL);
87     for (j = 0; j < 1000000; j++)
88     for (k = 0; k < 2*n+1; k++) {
89         where = binary1(n, a, k);
90 //         printf("who = %d, \tvalue = %d\n", k, where);
91
92 // debug
93     }
94     gettimeofday(&after, NULL);
95     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
          tv_sec, before.tv_usec,
96           after.tv_sec, after.tv_usec);
97     printf("The difference is %ld\n", timediff(before, after))
98     ;
99     printf("-----\n");
100    gettimeofday(&before, NULL);
101    for (j = 0; j < 1000000; j++)
102    for (k = 0; k < 2*n+1; k++) {
103        where = binary2(n, a, k);
104 //        printf("who = %d, \tvalue = %d\n", k, where);
105    }
106    gettimeofday(&after, NULL);
107    printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
          tv_sec, before.tv_usec,
108          after.tv_sec, after.tv_usec);
109    printf("The difference is %ld\n", timediff(before, after))
110    ;
111
112    n = n;
113
114    return 0;
115 }
116 // end

```

Listing E.7: M3.c

```

1  /* binary.c - Binary search using two methods. The first is
2     more intuitive, but it is
3     slower.
4  */
5  #include <stdio.h>
6  #include <sys/time.h>

```

```

7
8  /* Given a sorted array with n elements, we search for who
   using binary search.
9    We return a position where found, or -1 if not there
10 */
11 int function1(int index, int a[index], int parameter) {
12     int var1 = 0;
13     int var2 = index-1;
14     while (var1 <= var2) {
15         int mid = var1 + (var2-var1)/2;
16         if (parameter < a[mid])
17             var2 = mid - 1;
18         else if (parameter > a[mid])
19             var1 = mid + 1;
20         else
21             return mid;
22     }
23     return -1;
24 }
25
26 /* Given a sorted array with n elements, we search for who
   using binary search.
27    We return a position where found, or -1 if not there
28 */
29 int randomfunc(int index, int a[index], int parameter) {
30     int varvar = index/2;
31     while (index > 0) {
32         index = index/2;
33         if (parameter < a[varvar]) {
34             varvar -= index;
35         } else if (parameter > a[varvar]) {
36             varvar += index;
37         } else
38             return varvar;
39     }
40     return -1;
41 }
42
43 /* Returns the difference in microseconds between before and
   after */
44 long hourcalcul(struct timeval previously, struct timeval
   later) {
45     long sec = later.tv_sec - previously.tv_sec;
46     long microsec = later.tv_usec - previously.tv_usec;
47     return 1000000*sec + microsec;
48 }
49

```

Tests

```

50 int main() {
51     int arraynameda[] =
        {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
52     int n = sizeof(arraynameda)/sizeof(int);
53     int somevar;
54     struct timeval previously;
55     struct timeval later;
56     int count2;
57     int count;
58     gettimeofday(&previously, NULL);
59     for (count = 0; count < 1000000; count++)
60     for (count2 = 0; count2 < 2*n+1; count2++) {
61         somevar = function1(n, arraynameda, count2);
62     //     printf("parameter = %d, \tvalue = %d\n", count2,
        somevar);
63     }
64     gettimeofday(&later, NULL);
65     printf("before=[%ld,%ld], after=[%ld,%ld]\n", previously.
        tv_sec, previously.tv_usec,
66             later.tv_sec, later.tv_usec);
67     printf("The difference is %ld\n", hourcalcul(previously,
        later));
68     printf("-----\n");
69     gettimeofday(&previously, NULL);
70     for (count = 0; count < 1000000; count++)
71     for (count2 = 0; count2 < 2*n+1; count2++) {
72         somevar = randomfunc(n, arraynameda, count2);
73     //     printf("parameter = %d, \tvalue = %d\n", count2,
        somevar);
74     }
75     gettimeofday(&later, NULL);
76     printf("before=[%ld,%ld], after=[%ld,%ld]\n", previously.
        tv_sec, previously.tv_usec,
77             later.tv_sec, later.tv_usec);
78     printf("The difference is %ld\n", hourcalcul(previously,
        later));
79     return 0;
80 }

```

Listing E.8: M4.c

```

1  /* binary.c - Binary search using two methods. The first is
   more intuitive, but it is
2     slower.
3  */
4
5  #include <stdio.h>
6  #include <sys/time.h>

```

```

7
8 int binary1(int n, int a[n], int who);
9 int binary2(int n, int a[n], int who);
10 long timediff(struct timeval before, struct timeval after);
11
12 int main() {
13     int a[] = {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
14     int n = sizeof(a)/sizeof(int);
15     int where;
16     struct timeval before;
17     struct timeval after;
18     int k;
19     int j;
20     gettimeofday(&before, NULL);
21     for (j = 0; j < 1000000; j++)
22     for (k = 0; k < 2*n+1; k++) {
23         where = binary1(n, a, k);
24         //      printf("who = %d, \tvalue = %d\n", k, where);
25     }
26     gettimeofday(&after, NULL);
27     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
           tv_sec, before.tv_usec,
           after.tv_sec, after.tv_usec);
28     printf("The difference is %ld\n", timediff(before, after))
           ;
29     printf("-----\n");
30     gettimeofday(&before, NULL);
31     for (j = 0; j < 1000000; j++)
32     for (k = 0; k < 2*n+1; k++) {
33         where = binary2(n, a, k);
34         //      printf("who = %d, \tvalue = %d\n", k, where);
35     }
36     gettimeofday(&after, NULL);
37     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
           tv_sec, before.tv_usec,
           after.tv_sec, after.tv_usec);
38     printf("The difference is %ld\n", timediff(before, after))
           ;
39     return 0;
40 }
41
42
43
44 /* Given a sorted array with n elements, we search for who
45    using binary search.
46    We return a position where found, or -1 if not there
47    */
47 int binary2(int n, int a[n], int who) {
48     int p = n/2;

```

```

49     while (n > 0) {
50         n = n/2;
51         if (who < a[p]) {
52             p -= n;
53         } else if (who > a[p]) {
54             p += n;
55         } else
56             return p;
57     }
58     return -1;
59 }
60
61 /* Returns the difference in microseconds between before and
62    after */
63 long timediff(struct timeval before, struct timeval after) {
64     long sec = after.tv_sec - before.tv_sec;
65     long microsec = after.tv_usec - before.tv_usec;
66     return 1000000*sec + microsec;
67 }
68 /* Given a sorted array with n elements, we search for who
69    using binary search.
70    We return a position where found, or -1 if not there
71    */
72 int binary1(int n, int a[n], int who) {
73     int left = 0;
74     int right = n-1;
75     while (left <= right) {
76         int mid = left + (right-left)/2;
77         if (who < a[mid])
78             right = mid - 1;
79         else if (who > a[mid])
80             left = mid + 1;
81         else
82             return mid;
83     }
84     return -1;
85 }

```

Listing E.9: M5.c

```

1  /* binary.c - Binary search using two methods. The first is
2     more intuitive, but it is
3     slower.
4     */
5  #include <stdio.h>
6  #include <sys/time.h>

```



```

7
8  /* Given a sorted array with n elements, we search for who
   using binary search.
9     We return a position where found, or -1 if not there
10 */
11 int binary1(int n, int a[n], int who) {
12     int left = 0;
13     int right = n-1;
14     for (; right >= left; ) {
15         int mid = left + (right-left)/2;
16         if (a[mid] < who)
17             left = mid + 1;
18         else if (a[mid] > who)
19             right = mid - 1;
20         else
21             return mid;
22     }
23     return -1;
24 }
25
26 /* Given a sorted array with n elements, we search for who
   using binary search.
27     We return a position where found, or -1 if not there
28 */
29 int binary2(int n, int a[n], int who) {
30     int p = n/2;
31     for (; 0 < n ; ) {
32         n = n/2;
33
34         if (a[p] < who) {
35             p += n;
36         } else if (a[p] > who) {
37             p -= n;
38         }
39         else
40             return p;
41     }
42     return -1;
43 }
44
45 /* Returns the difference in microseconds between before and
   after */
46 long timediff(struct timeval before, struct timeval after) {
47     long sec = after.tv_sec - before.tv_sec;
48     long microsec = after.tv_usec - before.tv_usec;
49     return 1000000*sec + microsec;
50 }

```

Tests

```

51
52 int main() {
53     int a[] = {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
54     int n = sizeof(a)/sizeof(int);
55     int where;
56     struct timeval before;
57     struct timeval after;
58     int k;
59     int j;
60     gettimeofday(&before, NULL);
61     for (j = 0; j < 1000000; j++)
62     for (k = 0; k < 2*n+1; k++) {
63         where = binary1(n, a, k);
64         //      printf("who = %d, \tvalue = %d\n", k, where);
65     }
66     gettimeofday(&after, NULL);
67     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
        tv_sec, before.tv_usec,
68             after.tv_sec, after.tv_usec);
69     printf("The difference is %ld\n", timediff(before, after))
70     ;
71     printf("-----\n");
72     gettimeofday(&before, NULL);
73     for (j = 0; j < 1000000; j++)
74     for (k = 0; k < 2*n+1; k++) {
75         where = binary2(n, a, k);
76         //      printf("who = %d, \tvalue = %d\n", k, where);
77     }
78     gettimeofday(&after, NULL);
79     printf("before=[%ld,%ld], after=[%ld,%ld]\n", before.
        tv_sec, before.tv_usec,
80             after.tv_sec, after.tv_usec);
81     printf("The difference is %ld\n", timediff(before, after))
82     ;
83     return 0;
84 }

```

Listing E.10: M6.c

```

1  #include <stdio.h>
2  #include <sys/time.h>
3
4  long hourcalcul(struct timeval previously, struct timeval
    later) {
5      long sec = later.tv_sec - previously.tv_sec;
6      long microsec = later.tv_usec - previously.tv_usec;
7      return 1000000*sec + microsec;
8  }

```

```

9
10 int randomfunc(int index, int a[index], int parameter) {
11
12     // noise
13     index = index;
14
15     {
16         (1 == 1);
17     }
18
19     int varvar = index/2 + /* noise */ 1 - 1;
20
21
22
23     for ( ; index > 0; ) {
24         index = index/2;
25         if (a[varvar] > parameter) {
26             varvar -= index;
27         } else if (a[varvar] < parameter) {
28             varvar = index + varvar;
29         } else
30             return varvar;
31     }
32     return -1;
33 }
34
35 int function1(int index, int a[index], int parameter) {
36     int var1 = 0;
37     int var2 = index-1;
38
39     // random comment
40
41     for ( ; var1 <= var2 ; ) {
42         int xyz = var1 + (var2-var1)/2;
43         if (a[xyz] > parameter)
44             var2 = xyz - 1;
45
46
47         // comment
48         else if (a[xyz] < parameter)
49             var1 = xyz + 1;
50         else
51             return xyz;
52     }
53     return -1;
54 }
55

```

Tests

```

56 int main() {
57     int arraynameda[] =
        {1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33};
58     int n = sizeof(arraynameda)/sizeof(int);
59
60     n = n;
61
62
63     int somevar;
64     struct timeval previously;
65
66     /* long
67      * span
68      * comment
69      * to
70      * produce
71      * noise */
72     {
73         n = n;
74     }
75
76     struct timeval later;
77
78     // obs
79     int count2;
80
81     // opt
82     int count;
83
84
85
86     gettimeofday(&previously, NULL);
87     for (count = 0; count < 1000000; count++)
88
89         // more noise
90
91     for (count2 = 0; count2 < 2*n+1; count2++) {
92         somevar = function1(n, arraynameda, count2);
93 //         printf("parameter = %d, \tvalue = %d\n", count2,
somevar);
94     }
95     gettimeofday(&later, NULL);
96     printf("before=[%ld,%ld], after=[%ld,%ld]\n", previously.
        tv_sec, previously.tv_usec,
97             later.tv_sec, later.tv_usec);
98     printf("The difference is %ld\n", hourcalcul(previously,
        later));

```

Tests

```
99     printf("-----\n");
100
101
102     gettimeofday(&previously, NULL);
103     for (count = 0; count < 1000000; count++)
104     for (count2 = 0; count2 < 2*n+1; count2++) {
105         somevar = randomfunc(n, arraynameda, count2);
106
107         // debug
108     //     printf("parameter = %d, \tvalue = %d\n", count2,
109         somevar);
110     }
111     gettimeofday(&later, NULL);
112     printf("before=[%ld,%ld], after=[%ld,%ld]\n", previously.
113         tv_sec, previously.tv_usec,
114         later.tv_sec, later.tv_usec);
115     printf("The difference is %ld\n", hourcalcul(previously,
116         later));
117     return 0;
118 }
119
120
121 // noise at end
```

Tests

Appendix F

Assessment Creation Using the Advanced Functionalities

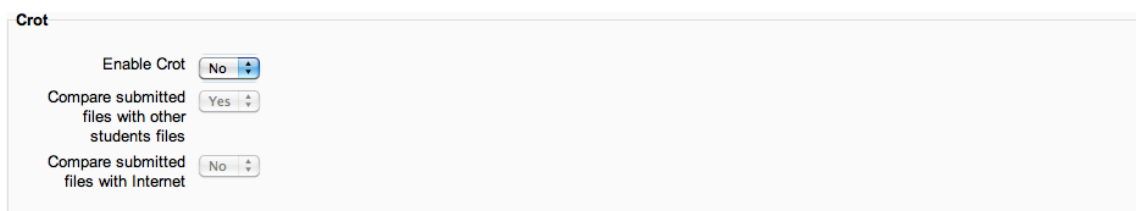
The objective of this appendix is providing a guide on how to configure, in Moodle, the advanced functionalities for a new programming assessment. It does not, however, explain step-by-step the creation of a programming assessment from scratch; Pacheco has covered that subject in the appendix "Assessment Creation Guide" of his thesis [[Pac10](#)].

F.1 Static Analysis of Code Quality

Every detail on how to operate with this module was thoroughly detailed and illustrated in the section [5.4](#) of this thesis.

F.2 Plagiarism Detection

In order to enable Crot upon creating a new programming assessment, a teacher needs to toggle "Enable Crot" in the Crot block, as depicted in figure [F.1](#) and set up the search parameters. Whereas this procedure can be applied to existing programming assessments, only the files submitted after the plagiarism detection plugin was enabled will be scheduled for comparison.



The image shows a Moodle configuration block titled "Crot". It contains three settings, each with a label and a dropdown menu:

- Enable Crot**: The dropdown menu is set to "No".
- Compare submitted files with other students files**: The dropdown menu is set to "Yes".
- Compare submitted files with Internet**: The dropdown menu is set to "No".

Figure F.1: Enabling Crot in a programming assessment

If "Compare submitted files with Internet" is selected, it is worth noting that the search consumes a much higher time, as it requires querying the search engine and downloading similar files from the Internet.

Finally, in order to check the maximum similarity score for submitted assessments, a Moodle user with grading capabilities has to click in the "View submitted assessments" link (figure 5.9) in order to proceed to the grading view (figure 5.12). Opening the link with the similarity score will take the user to the Crot similarity report view (figure 5.13), where he can click in any value to compare the offending documents (figure 5.14).

Appendix G

Communication Protocol

This appendix contains the full list of the front-end service description used for the communication between the main Automatic Assessment Server and the Moodle Server.

Listing G.1: Front-end service description

```
1 <definitions name="ProgAssessment"
2     targetNamespace="http://domserver.fe.up.pt/domjudge/
3     frontend/frontend.wsdl"
4     xmlns:tns="http://domserver.fe.up.pt/domjudge/
5     frontend/frontend.wsdl"
6     xmlns:xsd='http://www.w3.org/2001/XMLSchema'
7     xmlns:xsd1="http://domserver.fe.up.pt/domjudge/
8     frontend/frontend.wsdl"
9     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
10    xmlns="http://schemas.xmlsoap.org/wsdl/">
11
12    <xsd:complexType name="ArrayOfstring">
13        <xsd:complexContent>
14            <xsd:restriction base="soapenc:Array">
15                <xsd:attribute ref="soapenc:arrayType"
16                    wsdl:arrayType="xsd:string[]" />
17            </xsd:restriction>
18        </xsd:complexContent>
19    </xsd:complexType>
20
21    <!-- ##### setupProgassessmentModule
22         #####-->
23
24    <message name="setupProgassessmentModuleResponse">
25        <part name="result" type="xsd:int" />
26    </message>
```

Communication Protocol

```
24
25 <!--##### getLanguages
    #####-->
26
27 <message name="getLanguagesResponse">
28   <part name="result" type="ArrayOfstring"/>
29 </message>
30
31 <!--##### getMetrics
    #####-->
32
33 <message name="getMetricsResponse">
34   <part name="result" type="ArrayOfstring"/>
35 </message>
36
37 <!--##### getAllLanguagesInfo
    #####-->
38
39 <message name="getAllLanguagesInfoResponse">
40   <part name="result" type="ArrayOfstring"/>
41 </message>
42
43 <!--##### addNewAssessment
    #####-->
44
45 <message name="addNewAssessmentRequest">
46   <part name="id" type="xsd:int"/>
47   <part name="name" type="xsd:string"/>
48   <part name="timeLimit" type="xsd:int"/>
49 </message>
50
51 <message name="addNewAssessmentResponse">
52   <part name="Result" type="xsd:int"/>
53 </message>
54
55 <!--##### addNewAssessmentSpecial
    #####-->
56
57 <message name="addNewAssessmentSpecialRequest">
58   <part name="id" type="xsd:int"/>
59   <part name="name" type="xsd:string"/>
60   <part name="timeLimit" type="xsd:int"/>
61   <part name="special" type="xsd:string"/>
62 </message>
63
64 <message name="addNewAssessmentSpecialResponse">
65   <part name="Result" type="xsd:int"/>
```

Communication Protocol

```
66     </message>
67
68     <!--##### removeAssessment
        #####-->
69
70     <message name="removeAssessmentRequest">
71         <part name="id" type="xsd:int"/>
72     </message>
73
74
75     <!--##### updateAssessment
        #####-->
76
77     <message name="updateAssessmentRequest">
78         <part name="id" type="xsd:int"/>
79         <part name="name" type="xsd:string"/>
80         <part name="timeLimit" type="xsd:int"/>
81         <part name="nTestCases" type="xsd:int"/>
82     </message>
83
84         <!--##### updateAssessmentSpecial
            #####-->
85
86     <message name="updateAssessmentSpecialRequest">
87         <part name="id" type="xsd:int"/>
88         <part name="name" type="xsd:string"/>
89         <part name="timeLimit" type="xsd:int"/>
90         <part name="nTestCases" type="xsd:int"/>
91         <part name="special" type="xsd:string"/>
92     </message>
93
94     <!--##### addTestCase
        #####-->
95
96     <message name="addTestCaseRequest">
97         <part name="probid" type="xsd:string"/>
98         <part name="input" type="xsd:string"/>
99         <part name="output" type="xsd:string"/>
100         <part name="id" type="xsd:int"/>
101     </message>
102
103     <message name="addTestCaseResponse">
104         <part name="id" type="xsd:int"/>
105     </message>
106
107     <!--##### removeTestCase
        #####-->
```

Communication Protocol

```
108
109 <message name="removeTestCaseRequest">
110   <part name="id" type="xsd:int"/>
111 </message>
112
113 <!--##### updateTestCase
   #####-->
114
115 <message name="updateTestCaseRequest">
116   <part name="id" type="xsd:int"/>
117   <part name="input" type="xsd:string"/>
118   <part name="output" type="xsd:string"/>
119 </message>
120
121 <!--##### addParticipant
   #####-->
122
123 <message name="addParticipantRequest">
124   <part name="login" type="xsd:string"/>
125   <part name="name" type="xsd:string"/>
126 </message>
127
128 <!--##### participantExists
   #####-->
129
130 <message name="participantExistsRequest">
131   <part name="login" type="xsd:string"/>
132 </message>
133
134 <message name="participantExistsResponse">
135   <part name="result" type="xsd:int"/>
136 </message>
137
138 <!--##### addSubmission
   #####-->
139
140 <message name="addSubmissionRequest">
141   <part name="participantLogin" type="xsd:string"/>
142   <part name="assessmentId" type="xsd:int"/>
143   <part name="testCasesIds" type="xsd:Array"/>
144   <part name="language" type="xsd:string"/>
145   <part name="sourceCode" type="xsd:string"/>
146 </message>
147
148 <message name="addSubmissionResponse">
149   <part name="ids" type="xsd:Array"/>
150 </message>
```

Communication Protocol

```
151
152      <!--#####
      addCompileSubmissionSpecial
      #####-->
153
154      <message name="addCompileSubmissionSpecialRequest">
155          <part name="participantLogin" type="xsd:string"/>
156          <part name="assessmentId" type="xsd:int"/>
157          <part name="testCasesIds" type="xsd:Array"/>
158          <part name="language" type="xsd:string"/>
159          <part name="sourceCode" type="xsd:string"/>
160      </message>
161
162      <message name="addCompileSubmissionSpecialResponse">
163          <part name="id" type="xsd:int"/>
164      </message>
165
166      <!--##### addCompileSubmission
      #####-->
167
168      <message name="addCompileSubmissionRequest">
169          <part name="participantLogin" type="xsd:string"/>
170          <part name="assessmentId" type="xsd:int"/>
171          <part name="language" type="xsd:string"/>
172          <part name="sourceCode" type="xsd:string"/>
173      </message>
174
175      <message name="addCompileSubmissionResponse">
176          <part name="id" type="xsd:int"/>
177      </message>
178
179      <!--##### getSubmissionResult
      #####-->
180
181      <message name="getSubmissionResultRequest">
182          <part name="id" type="xsd:int"/>
183      </message>
184
185      <message name="getSubmissionResultResponse">
186          <part name="result" type="ArrayOfstring"/>
187      </message>
188
189      <!--##### Port
      #####-->
190
191      <portType name="ProgAssessmentPortType">
192
```

Communication Protocol

```
193 <operation name="setupProgassessmentModule">
194   <output message="setupProgassessmentModuleResponse"/>
195 </operation>
196
197 <operation name="getLanguages">
198   <output message="getLanguagesResponse"/>
199 </operation>
200
201   <operation name="getMetrics">
202     <output message="getMetricsResponse"/>
203   </operation>
204
205 <operation name="getAllLanguagesInfo">
206   <output message="getAllLanguagesInfoResponse"/>
207 </operation>
208
209 <operation name="addNewAssessment">
210   <input message="addNewAssessmentRequest"/>
211   <output message="addNewAssessmentResponse"/>
212 </operation>
213
214   <operation name="addNewAssessmentSpecial">
215     <input message="addNewAssessmentSpecialRequest"/>
216     <output message="addNewAssessmentSpecialResponse"/>
217   </operation>
218
219 <operation name="removeAssessment">
220   <input message="removeAssessmentRequest"/>
221 </operation>
222
223 <operation name="updateAssessment">
224   <input message="updateAssessmentRequest"/>
225 </operation>
226
227   <operation name="updateAssessmentSpecial">
228     <input message="updateAssessmentSpecialRequest"/>
229   </operation>
230
231 <operation name="addTestCase">
232   <input message="addTestCaseRequest"/>
233   <output message="addTestCaseResponse"/>
234 </operation>
235
236 <operation name="removeTestCase">
237   <input message="removeTestCaseRequest"/>
238 </operation>
```

Communication Protocol

```
239
240     <operation name="updateTestCase">
241         <input message="updateTestCaseRequest"/>
242     </operation>
243
244     <operation name="addParticipant">
245         <input message="addParticipantRequest"/>
246     </operation>
247
248     <operation name="participantExists">
249         <input message="participantExistsRequest"/>
250         <output message="participantExistsResponse"/>
251     </operation>
252
253     <operation name="addSubmission">
254         <input message="addSubmissionRequest"/>
255         <output message="addSubmissionResponse"/>
256     </operation>
257
258         <operation name="addCompileSubmissionSpecial">
259             <input message="addCompileSubmissionSpecialRequest"/>
260             <output message="addCompileSubmissionSpecialResponse"/>
261         </operation>
262
263     <operation name="addCompileSubmission">
264         <input message="addCompileSubmissionRequest"/>
265         <output message="addCompileSubmissionResponse"/>
266     </operation>
267
268     <operation name="getSubmissionResult">
269         <input message="getSubmissionResultRequest"/>
270         <output message="getSubmissionResultResponse"/>
271     </operation>
272 </portType>
273
274
275
276 <!--##### Bindings
277     #####-->
278
279     <binding name="ProgAssessmentBinding" type="
280         ProgAssessmentPortType">
281         <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
282
283     <operation name="setupProgassessmentModule">
```

Communication Protocol

```
282      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
283      <output> <soap:body use="literal"/> </output>  
284    </operation>  
285  
286    <operation name='getLanguages'>  
287      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
288      <output> <soap:body use="literal"/> </output>  
289    </operation>  
290  
291      <operation name='getMetrics'>  
292      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
293      <output> <soap:body use="literal"/> </output>  
294    </operation>  
295  
296    <operation name='getAllLanguagesInfo'>  
297      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
298      <output> <soap:body use="literal"/> </output>  
299    </operation>  
300  
301    <operation name='addNewAssessment'>  
302      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
303      <input> <soap:body use="literal"/> </input>  
304      <output> <soap:body use="literal"/> </output>  
305    </operation>  
306  
307      <operation name='addNewAssessmentSpecial'>  
308      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
309      <input> <soap:body use="literal"/> </input>  
310      <output> <soap:body use="literal"/> </output>  
311    </operation>  
312  
313    <operation name='removeAssessment'>  
314      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
315      <input> <soap:body use="literal"/> </input>  
316    </operation>  
317  
318    <operation name='updateAssessment'>  
319      <soap:operation soapAction="http://domserver.fe.up.pt/  
      domjudge/frontend"/>  
320      <input> <soap:body use="literal"/> </input>
```


Communication Protocol

```
321     </operation>
322
323         <operation name='updateAssessmentSpecial'>
324             <soap:operation soapAction="http://domserver.fe.up.pt/
325                 domjudge/frontend"/>
326             <input> <soap:body use="literal"/> </input>
327         </operation>
328
329         <operation name='addTestCase'>
330             <soap:operation soapAction="http://domserver.fe.up.pt/
331                 domjudge/frontend"/>
332             <input> <soap:body use="literal"/> </input>
333             <output> <soap:body use="literal"/> </output>
334         </operation>
335
336         <operation name='removeTestCase'>
337             <soap:operation soapAction="http://domserver.fe.up.pt/
338                 domjudge/frontend"/>
339             <input> <soap:body use="literal"/> </input>
340         </operation>
341
342         <operation name='updateTestCase'>
343             <soap:operation soapAction="http://domserver.fe.up.pt/
344                 domjudge/frontend"/>
345             <input> <soap:body use="literal"/> </input>
346         </operation>
347
348         <operation name='addParticipant'>
349             <soap:operation soapAction="http://domserver.fe.up.pt/
350                 domjudge/frontend"/>
351             <input> <soap:body use="literal"/> </input>
352             <output> <soap:body use="literal"/> </output>
353         </operation>
354
355         <operation name='addSubmission'>
356             <soap:operation soapAction="http://domserver.fe.up.pt/
357                 domjudge/frontend"/>
358             <input> <soap:body use="literal"/> </input>
359             <output> <soap:body use="literal"/> </output>
360         </operation>
```

Communication Protocol

```
361         <operation name='addCompileSubmissionSpecial'>
362         <soap:operation soapAction="http://domserver.fe.up.pt/
           domjudge/frontend"/>
363         <input> <soap:body use="literal"/> </input>
364         <output> <soap:body use="literal"/> </output>
365     </operation>
366
367     <operation name='addCompileSubmission'>
368     <soap:operation soapAction="http://domserver.fe.up.pt/
           domjudge/frontend"/>
369     <input> <soap:body use="literal"/> </input>
370     <output> <soap:body use="literal"/> </output>
371 </operation>
372
373     <operation name='getSubmissionResult'>
374     <soap:operation soapAction="http://domserver.fe.up.pt/
           domjudge/frontend"/>
375     <input> <soap:body use="literal"/> </input>
376     <output> <soap:body use="literal"/> </output>
377 </operation>
378 </binding>
379
380 <service name='ProgAssessmentService'>
381     <port name='ProgAssessmentPort' binding='
           ProgAssessmentBinding'>
382         <soap:address location='http://domserver.fe.up.pt/
           domjudge/frontend/frontend.php' />
383     </port>
384 </service>
385
386 </definitions>
```