DevEval
Porting e-learning technologies to lab examinations

Nuno Miguel Roque Gomes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: João Nuno de Oliveira e Silva

Examination Committee
Chairperson: Professor Nuno Cavaco Gomes Horta
Supervisor: Professor João Nuno de Oliveira e Silva
Member of the Committee: Professor David Manuel Martins de Matos

November 2015
Abstract

With the growing of e-learning technologies, more tools are being built in order to help assessing students’ performances. But one area which is not being properly focused is laboratory programming examinations (lab examinations).

The existing tools that focus on programming development are more oriented towards projects than assessments leading to a lack of assistance and support for lab examinations.

This work focus on allowing the electronic resolution of lab examinations. DevEval was built including the core modules of a Learning Management System platform, but integrates those capabilities with Integrate Development Environments in order to facilitate the development of lab examinations, by students. In contrast with existing tools, DevEval allows the storage of multiple answers per question. Since the tool has an open client-server architecture, it allows the expansion to other evaluation scenarios.

Keywords: E-learning, LMS, development assessment, development history, lab examinations.
Resumo

Com a evolução das tecnologias de e-learning, cada vez mais ferramentas são criadas com o objetivo de auxiliar na avaliação dos alunos. Mas as ferramentas existentes não têm como principal objetivo as avaliações laboratoriais (de programação). E as ferramentas que se focam na programação estão mais especializadas em projectos, fazendo com que as avaliações laboratoriais não tenham ferramentas que facilitem a sua avaliação.

Este trabalho procura criar uma solução eletrónica para colmatar essa falha. A ferramenta DevEval foi construída com base nos sistemas fundamentais que qualquer ferramenta LMS tem, mas permite a integração com IDEs, de modo a facilitar a resolução das avaliações laboratoriais por parte dos alunos. Outra diferença que o DevEval exibe, é o facto de permitir guardar mais do que uma resposta por pergunta. Dada a sua arquitetura de cliente-servidor, é permitida a expansão para outros cenários de avaliação.

**Palavras chave:** E-learning, LMS, avaliações de programação, histórico do desenvolvimento, avaliações laboratoriais de programação.
This thesis is dedicated to my family, specially my parents, brother and girlfriend, for all the support throughout the years I took to finish my studies and also during the development of this thesis. In a different but still meaningful way, it is also dedicated to my friends that helped me through those years, being with words of wisdom, helping me studying or by grouping with me for projects and laboratory classes.
Acknowledgements

I would like to express my gratitude to my thesis supervisor: Prof. João Nuno Silva.

This thesis couldn’t have been written, and the related work done, without his patient towards the copious amount of questions and doubts I had during the process. All the time he disposed to help me better understand the circumstances of the topic, and guidance towards its investigation, were fundamental to convert months of work into this final product.

I would also like thank the rest of my thesis committee: Professor Nuno Horta and Professor David Matos for their comments and hard questions, made in the intermediary delivery, which led me to use a better approach to develop this work and to write this thesis.
# Contents

1 Introduction  
   1.1 Problem statement ................................................. 2  
   1.2 Objectives .......................................................... 3  
   1.3 Results .............................................................. 3  
   1.4 Document Structure ............................................... 4  

2 State of the Art  
   2.1 Question and Test Interoperability ................................. 6  
   2.2 Cognitive process in programming ................................ 7  
   2.3 Student’s Behaviour ................................................ 8  
   2.4 Related Systems .................................................... 9  
      2.4.1 Version Control ............................................... 9  
      2.4.2 LMS .......................................................... 11  
      2.4.3 Automated Programming Assessing systems ................. 17  
   2.5 Discussion ......................................................... 24  

3 DevEval  
   3.1 Architecture ........................................................ 28  
   3.2 Data Model .......................................................... 29  
   3.3 Implementation ..................................................... 30  
      3.3.1 Server ........................................................ 31  
      3.3.2 Students’ Client ............................................ 34  
      3.3.3 Instructor’s Client ......................................... 38  
   3.4 External API .......................................................... 46  
   3.5 Fenix Integration ..................................................... 48  
   3.6 Evaluation ........................................................... 49  
   3.7 Reuse of existing LMS .............................................. 51  

4 Conclusion  
   4.1 Future Work .......................................................... 54  

Bibliography  ........................................................... 55
List of Figures

1.1 Example of a laboratory examination ............................................. 2

2.1 Generic architecture of a LMS [1] .................................................. 12
2.2 Obtained from Moodle's website [2] ................................................ 14
2.3 Assessment mechanism of AutoLEP [3] .......................................... 23

3.1 Generic architecture of DevEval ..................................................... 29
3.2 Representation of DevEval's data model ........................................ 30
3.3 Example of the server's directories layout ..................................... 31
3.4 DevEval's plugin toolbar shortcut ................................................. 34
3.5 Plugin login menu ................................................................. 35
3.6 Assessments List ................................................................. 35
3.7 Invalid assessment choice ............................................................ 36
3.8 Presentation of first question ....................................................... 36
3.9 Answer to first question ............................................................... 37
3.10 Answer to second question .......................................................... 37
3.11 Warning of assessment’s end ....................................................... 38
3.12 Index page of the HTML response .............................................. 39
3.13 Searching for UCs ................................................................. 39
3.14 Empty new assessment form ....................................................... 40
3.15 Warning shown whenever the initial date is posterior to the final date .. 40
3.16 Filled new assessment form ....................................................... 41
3.17 Semesters’ list for a specific course ............................................ 42
3.18 List of students for the specified semester .................................... 42
3.19 List of students who solved the specified assessment .................... 42
3.20 list of assessments the specified student solved ............................. 43
3.21 Introduction of the visualization webpage .................................... 43
3.22 Presentation of the answer for the first question ......................... 44
3.23 Presentation of the second answer for the second question ........... 45
3.24 Presentation of the first answer for the second question .................. 46
3.25 Example of the web-interface for creating an application to integrate with Fenix ................................. 49
# List of Tables

2.1 Three Generations of Version Control .................................................. 10
2.2 Comparison of VCS tools, relative to the proposed problem .......................... 11
2.3 Comparison of LMS tools, relative to the presented problem .......................... 17
2.4 Comparison of Automated Assessing tools, relative to the proposed problem .... 25
2.5 Comparison of the best tool of each system class .................................... 25

3.1 Available HTTP GET requests that provide HTML answers ............................ 47
3.2 Available HTTP POST requests that provide HTML answers .......................... 47
3.3 Available HTTP GET requests that provide JSON answers ............................. 48
3.4 Available HTTP POST requests that provide HTML answers .......................... 48
3.5 DevEval objectives verification .................................................................. 50
Chapter 1

Introduction

With the increasing developments in hardware and software technologies, it can be considered important to use, those which apply, to support and complement teaching and education, expanding the e-learning world.

The first e-learning tool was the PLATO program (Programmed Logic for Automated Teaching Operations) [4], and from then there have been the creation of standards, protocols and specifications, like QTI [5], and all the new, different and advanced software pieces that exist nowadays.

Teachers and students alike can benefit from e-learning, and even corporate training can. It does not have location restrictions and, compared to traditional learning, it also has much less time restrictions. While textbooks often become obsolete, creating the need for acquiring new editions, paying sometimes high amounts of money, this does not apply to e-learning, in the majority of cases.

It is even easier to have lectures or interactions with field specialists than having to bring them to a specific geographic place to talk in a class or a corporate seminar. [6]

All these advantages push more and more e-learning into the spotlight and, with all the technological advancements that appear every year that can be applied to improve e-learning platforms, it is becoming a solution to more people every day.

Great examples, that are becoming more impactful, are the Coursera, Udacity, and other identical MOOC - Massive Open Online Courses - projects. Some of the courses, in these projects, are lectured by teachers of renowned universities. Students that pass the courses have a certification by that university.

It is undeniable, nowadays, that the impact of e-learning should not be ignored, whatever the form it takes, being it a MOOC, certificated and complex platform, or a simpler Learning Management Systems (LMS) tool that helps teachers and students in assessing knowledge and grades. One type of e-learning platform is not comparable to the other (between MOOCs and LMS platforms) and both need to co-exist since they can be complementary and not competitors.
1.1 Problem statement

When looking for e-learning solutions, the amount and variety of existent tools can be impressive, with each one either trying to specialize in a specific aspect or being as generic as possible.

There are hundreds of solutions to chose from when we are presented with a problem the we want to solve with an e-learning tool. Still there are some areas not focused by any tools: some appear with new necessities that technological evolution brings, others are a minority when looking to the larger picture and are considered not as important or substantial.

Most of the e-learning platforms existing nowadays try to allow the creation of the most generic possible assessments’ form and give the most liberty to that process so teachers can elaborate their assessments how they want, with the complexity level they desire, but there is, at least, one type of assessment that, although can be forcibly adapted to fit these forms, existing platforms do not allow teachers to have the best possible way to assess the students.

This type of assessment is referred as lab examinations [7]. Figure 1.1 presented is from a lab examination of the course PSIS (Programação de Sistemas), from the Masters Degree in Electrical and Computers Engineering, at Instituto Superior Técnico.

A: (0.5 val) O programa principal lança dois processos, P1 e P2. Um semáforo identificado condiciona acesso a um recurso partilhado em exclusão mútua.

B: (0.6 val) Depois de lançar os processos, o programa inicializa um despertador para 20 segundos. Em cada 5 segundos, o programa imprime uma mensagem “\n”. Ao despertar, o programa envia os sinais SIGUSR1 e SIGUSR2, respectivamente, para o primeiro e segundo processos descendentes.

C: (0.5 val) Cíclicamente, os processos descendentes executam as seguintes tarefas: (1) adormece 2 segundos, (2) apropria-se do recurso, (3) escreve na saída a mensagem “Processo P ganhou recurso ‘\n’” (P é o número de processo lançado, P1 ou P2) e (4) libera o recurso após 2 segundos.

D: (0.5 val) Ao ser interrompido, cada processo descendente escreve no terminal o sinal de interrupção e termina com o código igual ao número de vezes que se apropriou do recurso.

E: (0.4 val) Depois de enviar os sinais, o programa principal espera pelo término dos processos descendentes, imprime os códigos de saída e termina.

Figure 1.1: Example of a laboratory examination

These assessments are the ones characterized by a somewhat simple enunciation, with a certain number of consecutive objectives students should achieve in a short period of time, usually inferior to the duration of a class. What is currently done, in most of the cases, is that in scheduled laboratory classes, students have a portion of the class to be assessed in one or more topics of the course: they receive the assessment and develop their answers, most of the times resorting to an IDE (Integrated Development Environment) in order to simplify some of the process - like compiling and syntax verification - and, when they finish, they send their code and solution through email or a submission tool so, later on, teachers can assess their solutions. The type of programming evaluation that our work tries to improve, does not consider projects, since these have a long duration and are not developed in a single timeline.
The generic forms that existing platforms provide (like multiple choice, free text or even linking words and images) are not appropriate for lab examinations. Lab examinations can be described as a set of consecutive questions without correct answers for any of them. As previously mentioned, teachers could adapt one of the existent forms, the free text ones, to use as an assessment for lab examinations. But that would imply that the students would use that platform application to answer them and not an IDE, as usual, losing all the features these provide.

When analyzing existing software, was observed that those would only store the latest answer for each question even though most of them allow students to repeat assessments, if teachers define so. This sounds reasonable for the usual assessment types but with lab examinations, having an history of what was done, could be beneficial.

In summary, there is not an e-learning tool that has a form for lab examinations, whilst allowing them to be solved in IDEs and storing multiple answers per question.

1.2 Objectives

Now that the problem is defined, this work proposes to solve it by creating an e-learning platform that addresses those issues.

The solution will start with the same base that other existing platforms, broadly referred as LMS have.

The problem is that existing platforms lack a more flexible Delivery system (which should allow integration with external tools), since those they implement are rigid and mostly unable to be integrated with other platforms, tools or plugins. Because of that it is not feasible to integrate their Delivery system with an IDE, which facilitates lab examinations resolution. These platforms are built to work only with their own web-tools. Some of them do not even have an API (Application Programming Interface) and those that do are not oriented to extend the default systems provided.

The requirements, that a solution needs to implement, are:

- Integration with external tools, so students can still use their IDEs to answer the lab examinations;
- More Flexible Delivery System, that allows integration with said IDEs;
- Storage of more than one answer per question, which should provide a deeper understanding of student’s difficulties;
- Oriented towards lab examinations, since they are not addressed in any existing tools;

1.3 Results

A tool that fulfills those requirements has been developed.

Firstly were defined the objectives, presented next in a more simplistic and visual form:

- Oriented towards lab examinations (not projects);
• Create a simple LMS tool that:
  – Has all the core modules of a LMS platform:
    * Database;
    * Assessment Manager;
    * Delivery:
      · Deliver assessments to authorized applications;
      · Deliver students’ solutions to instructors;
    * Authoring;
  – Allows the integration with external applications (through REST calls); [8]
  – Is capable of storing multiple answers per question, per student;

• Create an IDE plugin that demonstrates the potentialities of the developed LMS tool;

Then a tool was developed, DevEval, which has a server-client architecture with an REST API, which allows the development of plugins and other applications to integrate with it, and that implements the core functionalities of a LMS platform. DevEval is also able to store, in its database, more than one answer per question, of lab examination.

To demonstrate the capabilities of DevEval an IDE plugin was also developed, for Eclipse, that requests information about the assessments, presents them to students and registers their coded answers to send to the server.

For the teachers, DevEval provides a web-interface that allows the creation of new assessments and shows them the code history of students’ answers.

### 1.4 Document Structure

This thesis is organized as follows: the next chapter will present the state of the art of e-learning tools and other types that could provide some solutions to the problem presented. Firstly will be presented some history on the topic and then multiple tools and platforms will be analyzed, that work as briefly explained in the previous section and at the end of each one will be stated why each specific tool will not solve the proposed problem. There will be a variety of tools presented within LMS, Version Control and Automated Evaluation contexts.

The third chapter will present the proposed solution: how it was architected, which data models and communication channel were used and how it was implemented.

To finalize there will be a summary of all the information gathered and conclusions taken during the development of this work.
Chapter 2

State of the Art

As briefly mentioned in chapter 1, the e-learning subject has already more than 30 years of existence. This section discusses the history of this subject since its beginning, with PLATO, until today.

Although the term "e-learning" came to existence only in 1999, when it was utilized at a Computer Based Training (CBT) systems seminar [9], the concept has been documented throughout history. The first primitive form of distance learning was first recorded in the 1840’s when Isaac Pitman taught his students via correspondence, sending them work to do and receiving their answers.

In 1954, a Harvard professor by the name of BF Skinner invented the "teaching machine" which allowed the administration of programmed instructions to students and in 1960 the first CBT program was introduced.

This program was known as PLATO [10], abbreviation for Programmed Logic for Automatic Teaching Operations, and was originally design for students of the University of Illinois but ended up being used in other schools as well.

PLATO was created in the 1960’s when professor Don Bitzer, who had became interested in using computers for teaching, mustered some colleges to found the Computer-based Education Research Laboratory, CERL. Collaborating with colleagues and students they built a system that was ahead of its time, using TUTOR, a special-purpose programming language to write education software.

PLATO is a timesharing system, one of the first to be operated publicly. Both authors and students use the same display terminals that are connected to a central mainframe. Until 1972 it was a small system that supported only a single classroom of terminals. Then PLATO transitioned to a new generation of mainframes that would support up to one thousand users simultaneously.

Each student was provided with his own keyset and display. the first enabled him to control the sequence of materials presented, by the machine, as well as to transmit his answers to the computer. Besides presenting the textual materials to each student at a rate determined by him, the computer could also pose questions and the answers, that could be in the form of numerals, algebraic expressions or phrases, were judged by the computer without revealing the correct answer. The computer would also keep detailed records of each student's progress. In terms of display, the system had two communication modes: electronic book where the instructional material was on a set of slides; and electronic blackboard.
where the machine would also write figures or graphs, to the display.

It was required that the student would go through a fixed sequence of slides, answering each question posed in its course. The computer did not allow the student to skip slides, where it requested an answer, if the student had not answered all questions.

When the student indicated that he had completed his answers he pushed a “judge” button on the keyset and depending on whether it was right or wrong the computer would respond “OK” or “NO”. If the answer was wrong the student could revise it. And when he could not solve the question there was a contingency button, “help”, that would present a supplementary slide sequence. [4]

Although the time of appearance of PLATO was an era where technology advancement was not in its prime, this system was complete, versatile and innovative. This system laid the bases for e-learning technologies to build and improve upon.

The remaining of the chapter presents more recent systems, techniques and theories related to the e-learning and assessment of Lab examinations.

2.1 Question and Test Interoperability

When the first personal computers appeared, in the 1980s, people had it easier to learn about particular subjects and develop certain skills, supported by computers, and within the next decade, with the appearance of the internet, virtual learning tools began to thrive since several schools had set up the delivery courses online, reaching people with time and geographic constraints. In the 2000s even businesses started to use e-learning to train their employees.

With this progression in the industry, it was necessary to define standards and specifications and here one of the most important was the IMS QTI. QTI, that stands for Question and Test Interoperability, is a specification that is the standard in the industry nowadays. It had its first release for discussion in 1999 and in May/June of 2000 it was released QTI v1.0. Since then several updates were made and the last version is QTI v2.1 (August 31, 2012).

“The IMS QTI work specifically relates to content providers (that is, question and test authors and publishers), developers of authoring and content management tools, assessment delivery systems and learning systems.”

The main points, around which QTI is designed [5], are:

- To provide well documented content format, independent of the authoring tool that created them, for the exchange and storage of items and tests;
- To support the deployment, in a single LMS, of items and tests, from diverse sources;
- To provide systems with the ability to report tests results;

It is a specification that enables the exchange of data heterogeneously between all compliant systems and tools. The binding of the QTI abstract models is made with XML, eXtensible Markup Language. For each question element, named as items, QTI allows the attachment of a set of rules used to process
the responses to that item [1]. It also allows, on the assessment level, the definition of score rules and of score aggregation.

It is important to introduce QTI because of its role in the industry as the major specification for e-learning content. Almost all LMS tools integrate the QTI specification and part of them integrate also a more recent IMS specification, the Learning Tools Interoperability specification, LTI [11]. Its principal objective is to define a standard way of integrating learning applications (often remotely hosted and/or provided by third-party services) with LMS engines, education environments and other softwares. The simplest use case is to allow the connection of web-based/externally hosted applications to platforms that present them to the final users. This means that if a school develops an application, it can be used in any LMS engine that implemented this specification, without the need to develop and maintain custom integration for each engine that wants to use the developed application.

What the LTI specification allows is one LMS tool, the consumer, to integrate assessments from another, the provider. The integration happens on the level of data presentation, similarly to execution redirection. The problem is that data does not pass to the consumer and so it is not possible to extract information relevant to the assessments, within the consumer.

2.2 Cognitive process in programming

This section will provide a different view to the importance of storing multiple answers per question. As mentioned at the end of section 1.1, the usual assessments that these platforms focus on providing, do not require multiple answers at all. With programming development, being it either lab examinations or projects, that analysis can be different.

Several studies have been made relating programming with human cognition. In some of these [12] it is argued that cognition is a kind of computation, since both human and computer output result from operations carried out on symbols.

Cognitive processes include perception, attention, memory usage, understanding, problem solving and reasoning. These operations are also performed by a CPU with associated memory in order to obtain an output and like in a computer, information goes through different stages of cognitive processing and storage to give an output. It is also represented in symbolic form, as in computers. [13]

Programming is a high-level cognitive task since it requires a number of lower-level ones to be carried out at the same time. Cognitive systems lack the resources to perform multiple tasks at the same time and since the number of processes involved in programming is large, memory and attention have to be divided amongst each other, therefore limiting performance.

On one hand, controlled processes are similar to interpreted program code that still requires translation into machine terms at run-time. These require attention and are of limited capacity. [14]. But on the other, there is an analogy between compiled programs and automatic processes (cognitive-wise) since both are carried out directly: with automated knowledge of programming skills, resources can be released to be used in other programming tasks. While studying cognitive skills, in programming, is usual to compare performances of novice and experts. They differ in multiple aspects of problem solving: how
to represent the problems, chosen strategy and organization of knowledge about the problem.

In programming, novices tend to work forward: they write their programs line by line; but experts work backwards: they break the program goal into modular units. Even more, it was studied that insufficient short-term memory resources have been accountable for errors made by novice programmers. [15]

The importance of having multiple versions of a students code (like a timeline of their code) can be integrated in the cognition and programming subject. With more than just the final version, teachers can observe where students had more difficulties, which code they re-wrote, and other information, leading to better teaching methods and a possible personal approach to each student referring towards their main errors in the development of the assessment presented.

With the importance of what was just referred in mind, multiple versions of the answers of students in lab examinations are an extra that could come in handy. Even if not all teachers want to use that tool, some might and in the future having that information available could prove important for other reasons.

2.3 Student’s Behaviour

In a study made at University of Kent [16], regarding first-year Computer Science students, it was recorded and analyzed their programming development. During the study, the assignments always started with some template code and students had four weeks to complete them. Resorting to BlueJ [17], a pedagogic IDE intended to support the learning of Java, whenever students compiled their programs, “snapshots” of those were taken: it includes the complete source code, output information from the compiler, metadata, etc.

With that information it was possible to draw conclusions regarding novice’s ways to program and it was possible to learn:

- Which were the errors more encountered;
- That the majority of the time students spend on correcting syntax errors was towards only a few different error types;
- That students tend to recompile quickly: “it is more likely for a student to tweak and recompile their program in less than 10 seconds”;
- That students appear to write lots of code and then try to correct all the errors at once.

With the study it was also observed that many students will move on from a problematic piece of code, regardless whether it is corrected and without syntax errors. Stronger students will ignore syntax errors to work on other parts of the program and later return to those errors to fix them, while, when the other students do the same, they end up creating more, unrelated, syntax errors which is prone to get them stuck.

Student behaviour can infer students’ cognitive processes and so it is our understanding that allowing the storage and observation of multiple answers per question, or in other words multiple “snapshots” of students’ code, it is an asset.
Consider a student, solving a lab examination, which revolved around using *for* and *if* statements. When the instructor is assessing them, he notices that student $X$ failed defining the *for* boundaries. He would think that it probably was a distraction, and doesn’t think more of it. But, if the instructor had the answers history, he would see that, in each one, the *for* had a different definition, which could mean that it wasn’t a problem of distraction, but that student $X$ has some difficulties understanding who the *for* statements work. This is a good example to explain the importance of the answers’ history. With that information, the instructor can approach student $X$ about his difficulties in a more personalized manner.

Even though some of the instructors that would use DevEval might not use that option, it is still present for those who do.

2.4 Related Systems

This section evaluates existing tools and platforms, usually used in the assessment of programming and general skills.

Besides evaluating LMS ones, firstly will also be evaluated Version Control systems, since one of the points of the presented problem is that LMS tools nowadays do not store more than one version of the answers per question. And after will be evaluated Automatic Assessment of programming assignments.

2.4.1 Version Control

As mentioned, version control systems, VCS, implement a solution to one of the presented problems: managing multiple versions of the same file. In terms of applicability in e-learning that would mean having multiple answers per question.

When using these tools programmers commit their code to a repository whenever they feel it is appropriate (like after correcting a bug or adding a new feature) [18]. They can also synchronize their commits with their colleagues. This is one of the main benefits of using VCS: when working in a team programmers will not be stepping on each other’s code, since the system will either merge the changes, between commits made by different people to the same files, or warn every actor about it. Every time a programmer commits a change a new version of the corresponding file is created, hence the multiple versions per file, the feature referred previously.

The forty year history of VCS can be divided into three generations [19]:

- In the first, concurrent development was handled with locks, thus only one person could be working on a file at a time.
- In the second, users can modify the same file but they must merge the revisions before they commit.
- In the third, generation it is allowed to merge and commit separately.

The systems in table 2.1 are evaluated in the remainder of the section.
Table 2.1: Three Generations of Version Control

<table>
<thead>
<tr>
<th>Generation</th>
<th>Networking</th>
<th>Operations</th>
<th>Concurrency</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>None</td>
<td>One file at a time</td>
<td>Locks</td>
<td>RCS, SCCS</td>
</tr>
<tr>
<td>Second</td>
<td>Centralized</td>
<td>Multi-file</td>
<td>Merge before Commit</td>
<td>CVS, Subversion</td>
</tr>
<tr>
<td>Third</td>
<td>Distributed</td>
<td>Changeset</td>
<td>Commit before Merge</td>
<td>Git, Mercurial</td>
</tr>
</tbody>
</table>

Adapted from A History of Version Control [19].

Concurrent Versions Systems (CVS) was launched in the 80s and originally it would only allow the latest version of the code to be worked on and updated [20], as mentioned for second generation VCS. Nowadays it can handle branching and allowing the software to diverge with the option to be reconciled later. Since it has been in use for many years, has the advantage to be considered “mature technology” but its branch operations are “expensive” and moving/renaming files does not include a version update.

Still regarding the second generation, Apache Subversion (SVN) was created as an alternative to CVS fixing some of its bugs but maintaining compatibility with it. Both are free and open source. To prevent corruption in the database, SVN implements atomic operations: either all the changes are applied or none. SVN was designed to allow branching and so it has not the cost to do it. Another advantage is that it has a wide variety of plug-ins for IDEs, but still has bugs relating to renaming files and directories.

In the third generation comes Git with a different and radical approach from CVS and SVN. The original concept was to have a faster, distributed revision control system. It is primarily developed for Linux. It comes with a variety of tools to help navigate the history system and each instance of the source contains the entire history tree. Its branching model is what makes it stand apart from other tools. Programmers are encouraged to have multiple local branches independent of each other and the executing of the usual operations, like creating, merging and deleting, takes seconds [21].

Last but not least, also from the third generation, Mercurial that was originally created to compete with Git. It shares some features with SVN as well as being a distributed system and its documentation is more complete which facilitates learning the differences faster, thus being a good solution for those that already know how to work with SVN but want a distributed system tool. But users ca not merge two “parents” and is implemented with extension-base rather than scriptability. [20]

In table 2.2 it can be observed a comparison between all these tools, relatively to the proposed problem. The information obtained is that, in terms of being a solution to the problem, all fulfill the same requisitions and also fail on the same ones.

In summary there are a lot of VCS out there, which only a sample was evaluated, and each one has its own pros and cons but, although they can manage multiple versions of the same file they do not answer completely the problem presented, they only answer to part of it, even though the majority have IDE integration. The problem with these tools is that they are not LMS and so they do not have the e-learning common architecture, neither they apply e-learning specifications. If correctly integrated with a LMS they can be used as a code submission protocol.
Table 2.2: Comparison of VCS tools, relative to the proposed problem

<table>
<thead>
<tr>
<th>Tools</th>
<th>Lab examinations</th>
<th>LMS Core Modules</th>
<th>External Integration</th>
<th>Multiple files</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVS</td>
<td>Does not apply</td>
<td>Does not apply</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SVN</td>
<td>Does not apply</td>
<td>Does not apply</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mercurial</td>
<td>Does not apply</td>
<td>Does not apply</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Git</td>
<td>Does not apply</td>
<td>Does not apply</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

From these tools what can be important to retain is the technology and methodology of managing multiple versions of files.

2.4.2 LMS

E-learning offers the ability to share learning material, to conduct online classes and to communicate with professors via chat or others messaging applications [22]. Most of the times, it is a free solution to provide users the ability to learn in a personal tempo and around each one's lifestyle. It can be used by someone that has all day available to learn new subjects and topics, or by someone that only has one hour a day. Overall, and comparing to traditional learning, e-learning can be cheaper, less time consuming, or at least with a broader time span.

In this section some existing LMS tools, that could provide answers to the proposed problem, will be evaluated.

The general objective, that all these platforms have, is to provide a featured tool that implements e-learning technologies and specifications and that is simple and intuitive for the end-user. In figure 2.1 can be seen the standard model for the architecture of LMS platforms.

From all the presented modules, the core ones, that every LMS tool should have, are:

- **Database** where all the assessments, tests, questions, students' answers and grades are stored;
- **Assessment Manager** that manages all the queries for the mentioned items in the previous point;
- **Delivery System** that manages how students' answers are dealt with;
- **Authoring System** or **Import/Export System**, to create/obtain assessments and questions;

The other systems can also be important and so most of the LMS tools nowadays feature all the illustrated systems by default.

One definition for the previously mentioned LMS platforms can be: An platform that "allows you to create, distribute and track training anywhere, on any device" [23].

The traditional application of LMS is in educational institutions. They are used in order to deliver courseware and to popularize e-learning. But in the last decades companies have started using LMS to train employees and customers.

There are close to 600 tools of LMS available and each one is unique and is built to meet the needs of trainers and educators. But all of them have common grounds:
Figure 2.1: Generic architecture of a LMS [1]

- Registration Control and Distributed instructor and student base;
- Document Management: upload or creation of documents containing the assessments/curricular content;
- Assessment and testing;
- Grading and Scoring;

These are the main functions that an LMS platform must perform. But there are also other that will differentiate them and that depend on the main target of the platform itself. And like many IT innovations, LMS platforms have a number of benefits, such as:

- Easily adaptable and reusable material;
- More choices of delivery methods, techniques of evaluation, design of materials, etc.;
- Improvements in professional development and evaluation;
2.4.2.1 TAO

Based in the requirements defined earlier, TAO presents itself as one of the most advanced QTI authoring tools available [24]. It is the first platform to achieve QTI compliance in all four certification categories: Authoring and Editing Systems, Delivery Systems, Item and Test Bank Systems, and mostly importantly, the QTI content itself.

TAO started to be developed in 2002 [25] as a joint project between the University of Luxembourg and the research center Henri Tudor. The first prototype was released in 2004 followed by another in 2005. It is an Open Source engine, giving administrators access to the source code and control over the engine’s features and functionality.

When using it, users can create tests, register test-takers and deliver the tests online. It can also allow to edit items, preview them and to define scoring. It facilitates in the selection of items to group them and create tests, but allow the user to import and export QTI-compliant tests. It achieves all this with an intuitive user interface.

TAO comes even with an REST compliant API enabling integration with third party applications, such as Computer Adaptive Test engines (CAT) [26].

When looking for LMS tools to implement a solution for the proposed problem, TAO appeared to be the perfect tool for the job. It has an REST API and that would allow the integration with IDEs, with would only be necessary to create a module/plugin that would deal with that part. And so it was decided to use TAO as the solution.

Although it has been in development for more than ten years, when this project started the current version was TAO 2.6 [27] and was considered a Beta version. We later found out, this was particularly relevant since its API implementation was not complete: some of the default modules had the REST API built but others had not, and generally the documentation was not detailed, specially in these cases. But this information was not discretely available.

The main idea, when using TAO, was that teachers would use its default systems for authoring, grading, test-takers registration and grouping. What was needed was to create a plugin that would interact with the Delivery system in order to obtain data from tests and items and send them to any application that would require them, such as a plugin for an IDE, thus allowing TAO to integrate with IDEs for lab examinations. The developed plugin would also work with the database in order to store more than one answer per question.

During development a significant amount of problems arouse which precluded the usage of TAO as a basis for a solution. Some of those problems were related to the incomplete REST API and lack of documentation. Later on this work, this topic will be further explained.

With the knowledge obtained from the experience with TAO, was formulated a comparison basis to evaluating other LMS platforms so that the problems encountered were assessed before deciding for any other tool.
Moodle is one of the most used, if not the most, LMS platforms. In Portugal a wide variety of schools, colleges and universities use it as their base webservice.

In 1999, when Martin Dougiamas, the founder, started trialling prototypes of a new LMS, he came up with the idea for Moodle. In 2001, the first Moodle site was set up by Peter Taylor: Peter was Martin’s supervisor in his doctoral research.

The first version, Moodle 1.0, was launched in 2002. Users started discussing about it in a forum and module contributions started appearing. In 2004 companies started applying to become partners and by 2007, with improved documentation and certification Moodle established itself as a leading open source LMS. In 2010 Moodle 2.0 came out and after that, every six months, new features are delivered.

In its core Moodle is a learning platform designed to help educators and learners to create personalized learning environments with a single “robust, secure and integrated system”. It was over 65 million users worldwide across both academic and enterprise levels. It has a simple interface that implements drag-and-drop features, well documented resources and provides a flexible tool-set to support online courses. Since it is open source, it can be customized and tailored towards specific needs and with its modular set up it allows the creation of plugins and integration of external applications.

In fact, Moodle offers a wide range of solutions for a wide range of necessities and it has API implementation for most of its systems, so it could be a good base to build a solution for the proposed problem. The truth is that, in face of the problems encountered with TAO, an analysis into Moodle’s API leads to believe that is not suited since it has, at least, one of the problems TAO has: the Delivery system.

In spite of all the APIs provided a simple look into the Delivery related one shows that, the “Question API” that should provide, is not fully developed and that the documentation is “in a bit of a mess” has can be observed in figure 2.2.

Although the “Question Bank” API has the import/export calls, with the previous experience with
incomplete APIs and documentation it was decided that there was no need to revive all the troubles that we had with TAO.

2.4.2.3 <e-QTI>

*e-QTI* is a highly modular QTI assessment engine that allows instructors to author, deliver and import/export QTI assessments [1].

For the authoring action *e-QTI* provides a user-friendly special-purpose editor, in order to increase productivity, since the other option would be to deal directly with QTI XML binding. This tool supports several question types: True/False, multiple choice, fill in the gaps, matching lists and essays.

This platform also controls the presentation of the assessments as well as the interaction between the learner and these assessments, through a Delivery system. It allows learners to partially complete an assessment and finalize it later, to review the evaluation and displaying feedback and gives information about remaining time to complete the assessment.

It also supports grading of assessments. Its Grading system is highly flexible in order to allow and accommodate different types of evaluation styles, ranging from the automatic correction of simple items to explicit involvement of instructors. This system can still present a final report to the learners, regarding their grades.

The import/export module allows teachers to import or export QTI-formatted assessments.

In the end *e-QTI* is a platform that improves the usability of users by hiding the QTI details and by using templates to improve productivity.

Its important to notice that this tool has no API and its calls are all internal to the platform, thus making it not viable to integrate its features with an IDE. Thus, although it is a good LMS tool, it is not capable of solving the presented problem.

2.4.2.4 TalentLMS

In 2012, Epignosis Ltd. founded TalentLMS. Its mission was to provide access to high quality e-learning tools, for companies of any size, in order to democratize learning [30].

It was built to increase users satisfaction and fun from online learning and training and so it was built to be easy to use. There is also no need to install a server, since it is a cloud powered LMS. It is also capable of delivering reports that are simple and with comprehensible analytics [31].

This platform includes Branches, gamification, social media sharing, video conferencing and authors have the possibility to sell their courses. The platform is optimized for iPad and mobile devices to provide users with access anytime. As a LMS tool, it has the base modules of authoring, assessment management and delivery.

*TalentLMS* provides a REST API which leads to over 300 applications developed to improve functionality [30].

But, once again, looking into the documentation of the REST API [32] it is observed that they do not provide complete access to their delivery system calls, regarding to external tools. Developers can
request answers for tests and surveys, but they cannot request questions created or assessments.

The problem with this tool, regarding the proposed problem, is similar to Moodle and TAO. All three have REST APIs but none allows developers to interact fully with their delivery system.

2.4.2.5 Edmodo

Founded in Chicago, Illinois, in 2008 by two school district employees, the basis on which Edmodo [33] was founded was to bridge the gap between how students learn in school and how they live their lives. It is now the number one social learning network in the world for K-12 (from primary to secondary education). It is presented in a simple and intuitive interface which has for the main interaction form a wall (similar to Facebook).

As with all the tools presented until now, it has the capability of creating different groups of students and of dividing them into subgroups. Its base is one of continuing evaluation which can be qualitative or quantitative. Teachers can achieve this through multiple types of tasks that they publish on their “wall” for students to solve. As TalentLMS, it does not require installation, since it provides its services from the cloud. Another big difference, in terms of implementation of concept of an LMS, besides the “wall” posting style similar to social media networks, is that this platform allows the distinction between teachers, students and parents. When the platform is being used by under age students, it allows parents to have access to their kids’ walls.

The biggest disadvantage is that it does not allow solving the assessments in the platform itself [34]. Students cannot send private messages, neither individual ones. Everything shared is public. In the end, Edmodo is the fusion between a basic LMS platform, without the delivery system part, with social networks.

Since it does not even have a complete delivery system, it cannot solve the proposed problem, but it is an important tool in the context of e-learning and LMS, since it is one of the most used worldwide and with a different approach than the others.

2.4.2.6 Discussion

Since there are more than 600 LMS platforms, those just evaluated were the ones that had the most probable contribution to give into the presented problem.

Similarly to the previous section, the table 2.3 compares the presented platforms towards the goal that is pretended. It is easy to conclude that, as expected, all evaluated platforms provide the LMS’ core modules. Although all of them have a multitude of formats for the creation of assessments, none has a specific for lab examinations as it has been already stated. It is interesting to note that some tools already facilitate external integration, but as it was discovered, the delivery system is not usually in available APIs and when it is, the available calls have little flexibility. To finalize, as it was also expected, none of them deals with multiple versions of the same file, or in other words, with multiple answers per question.
<table>
<thead>
<tr>
<th>Tools</th>
<th>Lab examinations</th>
<th>LMS Core Modules</th>
<th>External Integration</th>
<th>Multiple files</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAO</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>Moodle</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>&lt;e-QTI&gt;</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TalentLMS</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>Edmodo</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

2.4.3 Automated Programming Assessing systems

There is another field of e-learning, more focused on programming development, that tries to provide automated programming evaluation. These systems do not have a base architecture similar to LMS tools, they mostly do not provide the authoring, assessment management or delivery modules, but they focus on code evaluation and analysis and are mostly able to integrate with IDEs.

Out of the scope of e-learning tools, these are the ones that still focus on helping assessing, specifically on programming.

The first time automated programming assessment was referred was in 1965. [35]

One of the main points to be considered when talking about this field is the difficulty of students to get programming skills and the main path to deal with this has been raising the quantity of programming exercises accompanied with feedback. But this overloads the teaching staff with an excessive workload that is somewhat repetitive, which is not mostly good.

Many studies have been made towards development of tools that automate that evaluation process. These tools would guide students and free teaching staff to do a more focused help instead. This is the main objective for these systems. Then, each implements features that will distinguish themselves from the others, and since new gaps are constantly appearing, there is always margin for a new tool with a feature that complements it. The most common gaps are referent to plagiarism detection, secure test environments or even diversity of grading rules.

The common goal is to improve programming skills in students, which will happen through solving multiple programming exercises.

Some tools even integrate with a LMS platform to improve the overall assistance of the assessment process [36].

2.4.3.1 Web-CAT

Web-Cat defines itself as a flexible automated grading system that is designed to evaluate programming assignments [37].

This tool provides all its capabilities through a web interface, since it runs on a server. It distinguishes itself, from other existing automated programming assessment tools by allowing students’ testing activities, besides the standard judging by comparing output results. This tool also allows students to “try out” their code and tests often in order to receive timely feedback. Which means that some of Web-CAT’s
focus is around student’s capability of testing their own code.

As of 2006, there were no mainstream tools in this field, despite the numerous works in automated grading, and so Web-CAT has been evolving to fill that role. The belief that this system is still the appropriate for the role is based in the four core strengths of the tool itself:

- **Security**: the user module incorporates a plugin based authentication approach, and since the tool has an open API, administrators can choose one of the multiple available authentication strategies or even provide their own authentication adapter. Besides that, services have user-specific permission levels, access control system and the system even provides detection and protection of the students’ erroneous programs or malicious code. With the implementation of a relational database, with specific security policies, the data integrity is secured.

- **Portability**: since Web-CAT is written in Java it gives the system a high degree of portability: it can be distributed as a Java servlet application. Portability is also implemented through the client service that only requires the use of a web browser.

- **Extensibility**: this tool was designed with an architecture based on plugins. This allows innovative features to be added to the system without changing its code. Furthermore, instructors can define the processing of assessments resorting to a amount of existent plugins, even having the possibility to group more than one.

  Web-CAT is also language-neutral and this means that it is capable of handling submissions in Java, C++, Scheme, Prolog, Standard ML, Pascal and can be adapted for others.

- **Manual Grading**: this tool also has built-in support for manual grading that follows the automated one. Course staff can visualize students submissions, comment on them, give suggestions, deduct points or give extra credit, they can also keep track of the students progress through multiple file submissions.

  These features are not exclusive to Web-CAT, but this tool is “the only automated grader published in the CS education literature that combines all of these strengths in one system” [37].

  This is a versatile system that empowers e-learning of programming courses, but this tool does not directly solve any of the defined problems: IDE integration is achieved by some developed plugins, but the orientation of those is towards the submission system. This means that, although there is no existent plugin that does what is requested, it could be built.

2.4.3.2 Marmoset

In 2006 Jaime Spacco started developing an automated programming assessment tool for his Ph.D. thesis, under the direction of Bill Pugh [38]. Based on the premise that one of the problems with Computer Science (CS) courses is that instructors only had the final submission of students assessments to evaluate their knowledge and skills, they built an Eclipse plugin to address the lack of feedback between students and instructors [39]. This plugin had two main features:
• Use a CVS repository to store the complete state of students’ projects every time they saved, added or removed files, producing a fin-grained history of the development of each project.

• The plugin allowed students to easily submit their projects to a central server that would perform tests on each submission and return feedback on how that submission performed in face of the requirements.

The developed plugin had a direct interaction with the CVS repository: whenever the plugin captured an action towards a file of the open project it would automatically call CVS commands to add or remove the file in question. This brought a thinner granularity of the history of projects than those that have been obtained with other tools and system. Although there could be an even thinner granularity, since between saves students could have written large blocks of code and then delete it, they believed that the granularity they provided was sufficient for obtaining comprehensive data on how students develop their code.

The plugin also has a submit button that submits a zip of the project’s files, with a timestamp, into the server. This server, connected to a database with records of information for each submission already made, works with an external tool to build the submissions. After that the submission goes through three categories of tests:

• **quick tests**: tests that are distributed to students;

• **release tests**: these are not immediately available to students, although they can request the server to perform them. When they do, the answer comes in the form of the number of passed and failed tests and the name of the first two failed tests. The idea is that tests have a semi self-explanatory name, so students might infer by the name what type of test was made so they can try to locate the errors. The number of request of these tests is restricted: each student has three tokens and requesting this type of testing consumes one of them. After 24 hours the consumed token is regenerated. This is a way to incentive students to develop good development and testing habits since they cannot "spam" these tests.

• **secret tests**: the results of these tests are only available to students only after the final submission that comes with the project’s deadline.

Instructors are allowed to re-define many aspects of these tests, like the number of tokens available or the amount of time to regenerate. They can also simplify the names of the tests in a way that it can slightly help students or in a way that they are not congruent with the tasks required from the project assignment.

The instructor has, of course, access to all submissions results.

As of October 12, 2015, the Marmoset Project has suffered some structural changes, especially towards in-browsing usage of the system.

Currently it is hard to understand if the Marmoset Project is still being updated and/or provided. The main website [38] is still up, but the demo server is down and information about the project is hard to find.
after 2011. Still, in 2011 there was a major revision of the project’s structure with a focus on in-browser code review.

Although Marmoset was the project that came closer to solve the presented problem, evidence points towards the end of the project some years before this work started. Even if it was still being developed/updated, the latest evidence was not clear enough towards the usage of the original Eclipse plugin that recorded the history of the development. This is inferred by the structure revision and in the fact that, in 2011, David Hovemeyer which is one of the authors of the project, created an Eclipse plugin named “SimpleMarmosetUploader” [40, 41, 42]. In fact, in David Hovemeyer’s CS 201 course of 2015 [43] there is a link to “Marmoset (submission server)” but that redirects to the implementation of 2013 [44].

2.4.3.3 Syde and Replay

One problem that is common in long-term projects is how easy it can be for a developer to forget what he worked on before the weekend or what his teammates worked on. Furthermore, when in parallel development, problems that cause delays are most times caused by communication breakdowns, difficulty on tracking the introduction of a defect, solving conflicts, etc.

Replay [45] is a tool developed to try solving those problems by helping developers finding in an easier fashion the source of the problem they are facing. Replay is incorporated into Syde [46], a previously software developed by one of Replays’ authors. This tool provides the records of fine-grained changes by tracking them in the IDE. Syde is implemented as a client-server application where the server records the mentioned changes, manages the current state of the project and publishes information about the teams’ activity. On the other hand, the client is a collection of plugins for the Eclipse IDE that track changes.

The focus is on object-oriented systems, specifically Java, because of the ease at storing and analyzing classes and methods, instead of lines of code and files, if it was not an OO language. The current state of the system is represented per developer. When a change appears it is interpreted as a transition of the system from one state to the next. There are five core components that define Syde:

- **Inspector**: this is the plugin responsible for scouting changes on the IDE. When one appears it is translated and sent to the server. This component also provides an API facilitating the creation of new applications to extend its capacities and usefulness.

- **Collector**: This module deals directly with the previous one, since it is the one that receives the changes reported by the Inspector. It then proceeds to save them and it also keeps, in each developers’ workspace, the state of the system.

- **Notifier**: This component is responsible for broadcasting the information about any changes that occurred or are occurring. It receives that information directly from the Collector.

- **Conflict Detector**: As the name states, this module scouts for potential conflicts that might appear from changes on developers’ workspaces.
• **Viewer**: This component is composed by a collection of *Eclipse* plugins that use all the information gathered to visually show it. This is where *Replay* enters, as it allows developers to visualize past changes in the order they occurred.

*Replay* groups all the atomic changes that *Syde* gathers, and they are grouped by three criteria: timestamp, developer and artifacts (compilation unit or package). Each group is only allowed to have one change on one entity, in order to maintain the granularity.

*Syde* in conjunction with *Replay*, create a very complete system that facilitates development. Although this project has no defined focus in e-learning directly, its implementation is a good study case, because of their approach to fine-grained changes analysis. They perform the history part that is defined in the proposed problem and even have integration with an *IDE*. But, like the problem in the previous tool, they have no relation with an *LMS* tool: they do not provide the delivery of assessments; furthermore this tool focus on long term projects while the frame of the defined problem is small, short-timed *lab examinations*.

### 2.4.3.4 Blackbox

Under the premise that automatically recording the programming behavior of novices is a computing education research technique, project *Blackbox* was created. [47] This project makes use of the availability of “always-on” internet access to collect large portions of data from cross-institutional studies, instead of other similar solutions do, since they are applied only at single institutions or on a smaller scale.

*Blackbox* was develop to interact with *BlueJ*, a *Java IDE*, oriented to students. In 2012 had, at least, 1.8 million users, which is the reason why this *IDE* was chosen.

The main objective is to collect huge amounts of data to be available to researchers in the field of computing education research.

To achieve what is proposed, *Blackbox* draws information from the user pool of *BlueJ* and records the code they write, as well as their interactions with the *IDE* (in a similar fashion of what the *Eclipse* plugin of *Marmoset* did).

After being launched, in June 2013, the first time any user loaded *BlueJ* was presented with a dialog asking them to opt-in to the *Blackbox* project. As long as users are programming with *BlueJ* and opting-in *Blackbox*, it can be considered a “perpetual data collection project”.

In order to obtain the maximum amount of users to opt-in, *Blackbox* works anonymously. This way users have no reasons to suffer negative consequences from participating. However, if researchers want to utilize this tool in a smaller environment, like a course or department, they could configure it to request information that identifies the users. This data is never sent to *Blackbox* servers and the researcher is responsible for ethics approval, participants consent and data’ safeguarding.

Although, in terms of “registration”, *Blackbox* is anonymous, thus not requiring any, it is not possible to provide 100% anonymity, since in the code, comments, function’s names, variables, etc., users can use personal information. To reduce this undesired effect, the tool hashes the project path, except its name (removing retrievable user data from it), and blanking comments in classes headers, since this is
were, usually, authors put their names. Furthermore, there is a notice, when opting-in, informing users of this.

In terms of recorded data, it includes a persistent unique identifier, assigned on opting-in and that persists across sessions, although the system does not track how many users use the same computer with the same identifier or, on the other hand, how many computers, thus, how many identifiers some users have, since they could have one for their personal computer, another in the school laboratory, etc. It also includes start and end times of every session as the use of all the IDE tools, like compilation and execution - each event has its own time stamp and relevant details. Furthermore, it is also recorded the editing behavior, by collecting information at source line level. This is implemented by awareness on when users are editing a line and then move the cursor out of that same line.

To conclude, this is a powerful tool regarding research of computing education. It can be used to provide up-to-date information for researchers. In terms of being a solution to the presented problem, Blackbox implements the history part and also the IDE integration. But it is only functional for Java, limiting the courses that would want to use such a solution. Besides that, this project presents the same problem as other tools in this system class: it does not have the components of a LMS tool, especially the Delivery system of assessments.

2.4.3.5 AutoLEP

Usually, in this field of automated assessment of programming development, the existent solutions focus on automating the assessment of the final submissions of projects. AutoLEP [3] goes further and also assesses the source code. This type of assessment is important because students can solve the assessments without using the requirements but still obtaining the expected output and tools that only assess that output will evaluate the project as correct when it is not.

Figure 2.3 represents the states which students’ programs go through to be evaluated.

For the static analysis, there is a model program (defined by the instructors) which is compared with students code. This is accomplished by transforming both into system dependence graphs.

A system dependence graph, SDG, is a directed graph where the nodes represent the program statements and predicates, and the edges represent the dependencies between them. These graphs allow the analysis of programs properties. [48]

After that transformation, both results suffer semantic-preserving transformations, thus eliminating program variations so that syntactically different programs, but that have the same algorithm, can match with the model provided. This matching defines the score of the submitted project by taking in consideration the criteria defined in the model. The downside is that assessment correctness is proportional to how detailed the model is. That effect can be reduced by following, if necessary, with a dynamic testing. This process is similar to the processes other tools pass programs through: running the submitted program through a battery of tests and then comparing the output with the expected results.

Developed in Harbin Institute of Technology, it has been in use since 2004 on that same institution in C programming courses.

Although this tool does not present a viable solution to the presented problem, it is important in a
broader scale since it can help to provide better assessment to students and at the same time alleviate the workload instructors are under.

2.4.3.6 ClockIt

*ClockIt* [49] is a tool developed to help students with bad work habits in programming courses, since the authors believe that it was the cause for capable students to be unsuccessful. The idea is to have a data collection and analysis toolset that allows teachers to measure students’ development practices and compare them to more experienced developers. To achieve this, the developed toolset can be used to determine the full time frame of student’s development: from when they begin to when they end the assessment.

One of the components of this tool is a Data Logger/Visualizer extension for *BlueJ* that captures events that occur during development, adding them to a log file, and that displays graphs of the captured data. The extension registers events of many types: compilation successful, warning and error, package open and close, invocations, project open and close and file change and delete. Each event, when it gets registered in the log file has an identifier, a time stamp and a description.

With this information is possible to estimate how much time and effort, the student being evaluated, has spent editing a file before compiling and running.

The visualization part allows the user to select from a wide range of views: since a high level report...
to a detailed one. It can be seen how many events happened during a specified time span, which can
lead to a representation of how the student develops his projects: short burst with many events, long
periods with few events, long sessions right before the deadline, etc.

The other component is a web interface that allows access to the database in order to visualize, in a
graphical form, computed measures of the information obtained from the log files. Students can access
their own data and instructors can access the data of all the students enrolled in their courses and also
have the possibility of visualizing the average of a class.

In a study done with CS students and faculty at Appalachian State University, base for the develop-
ment of this tool, they asked them if they though that they (or their students if it was being questioned a
instructor):

- tested thoroughly during the development;
- compiled frequently;
- developed incrementally;
- rewrote messy sections of code;

The disparity between students and faculty answers were very large. One example is that 92% of the
students thought they compiled frequently, but only 25% of faculty believed it. This survey motivated the
development of this tool, since it can help answer those questions and with those answers students and
faculty have a easier job tackling the more apparent difficulties for each case.

As with other tools in this field the response this tool provide towards the defined problem is relative
to the IDE integration and history of development. The lack of LMS components continues evident in
these tools.

2.4.3.7 Discussion

Once again, table 2.4 compares these tools. It is easy to verify that these tools have a different focus than
LMS ones, showing that they could be complementary. This is evidenced by comparing, in both classes
of systems, the component that all tools of that class provide: LMS provide always its Core modules,
thus a Delivery, Authoring and other systems, but neither of them is oriented to lab examinations; on the
other hand, Automated Assessing tools are oriented to those types of assessments but do not have the
modules LMSs have.

2.5 Discussion

Even when grouping the best tools of each one of these classes, regarding the presented problem, it is
evident that, alone, none can solve it. Table 2.5 displays this comparison and adds, in the final row, the
objective for the tool to be developed.
Table 2.4: Comparison of Automated Assessing tools, relative to the proposed problem

<table>
<thead>
<tr>
<th>Tools</th>
<th>Lab examinations</th>
<th>LMS Core Modules</th>
<th>External Integration</th>
<th>Multiple files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web-CAT</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>Marmoset</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Syde/ Replay</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>BlackBox</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>AutoLEP</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ClockIt</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.5: Comparison of the best tool of each system class

<table>
<thead>
<tr>
<th>Tools</th>
<th>Lab examinations</th>
<th>LMS Tool</th>
<th>Multiple files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Core Modules</td>
<td>External Integration</td>
</tr>
<tr>
<td>Git</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Moodle</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
</tr>
<tr>
<td>Marmoset</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
</tr>
</tbody>
</table>

**OBJECTIVE**

Version Control systems are effective in managing multiple files and are easily integrated with other tools, plugins, etc., but they are not e-learning tools, although they can be integrated to perform their part.

**LMS** tools are closer to what is wanted since they have a database and system for managing all the assessments, which are created in the Authoring systems by the instructors, also have Delivery systems that can display the assessments to students and even display students’ solutions to instructors and some of those tools even have an **API** that facilitates development of applications to interact with the existing system. Although some **LMS** tools allow a certain degree of external integration, upon further analysis, it was concluded that their **APIs** do not allow alterations to the Delivery system in order to implement multiple answer per question, as wanted for the solution. Besides this, none of the evaluated **LMS** tools has assessments forms for **lab examinations**.

The last class of systems analyzed was the Automated Programming Assessing tools. These are solely focused on assessing programming development, which solves one part of the presented problem. The majority even have some degree of external integration since they are applied directly in an **IDE** in order to obtain students code. It is understandable that those who do, also allow multiple files. If those tools did not needed multiple versions of files from an **IDE** but only the final project, they would only need a final submission and that process does not has necessarily to be done by a plugin or application of an **IDE**, since it could be manually done by the students.

In the end none of the evaluated tools and none of the existing, as far as our knowledge goes, can implement all the requirements needed to achieve a solution for the proposed problem.
Chapter 3

DevEval

Looking back at table 2.5, specifically to the last row, the one corresponding to DevEval, it displays which components this tool has to implement, in order to be a solution for the proposed problem.

The solution should allow instructors to create assessments and to watch student’s development history. For students, the tool should allow them to solve the assessments in an IDE, allow the choosing of an assessment and then show them all the questions, while allowing them to answer in whichever order they prefer. The system should only present assessments when they are in the date range. These are the requirements that a solution needs to fulfill.

The objective is then to create a tool that is focused on programming assessments, as tools like Marmoset and Web-CAT are, with one difference: those tools are most of the times focused on long-term projects and the focus required, in this project, is towards lab examinations.

The solution requires the implementation of all the LMS architecture’s core components. Instructors need an authoring tool so they can create the assessments. It is important that those assessments be stored in a database and managed correctly: only to be displayed to students with access. The delivery system has to be capable of sending assessments to those applications that ask for it - if properly authorized-, to store students answers and display them to instructors, similarly to what has been implemented in all LMS existing platforms, like Moodle or TAO. Furthermore, it should be able to store more than one answer per question, which has not been done by any LMS platform. This is one of the critical points with existing platforms: their Delivery system is rigid, does not allow changes to that functionality; they might have an API to interact with this system but they do not allow its modification or even feature adding to it, thus being impossible to save more than one answer per question, only the final one is stored.

To facilitate the use of these type of tools by students it was defined, as part of the problem, integration with applications: the created tool needs to have a form of communication with external applications. This can be achieved by making the tool as RESTful as possible.

In the end, to test all the implemented features and to provide an application that could be used by students in an IDE, it was also defined the development of a plugin.

Summing it up, the developed tool has to:
• Be oriented towards *lab examinations* (not projects);
• Be built as a *LMS* tool that:
  – Allows the integration with applications (through REST calls);
  – Has all the core modules of a *LMS* platform:
    * Database;
    * Authoring;
    * Assessment Manager;
    * Delivery system:
      • Deliver assessments to authorized applications;
      • Deliver students’ solutions to instructors;
      • Be capable of receiving multiple answers per question;
• Create an *IDE* plugin that interacts with the developed tool by requesting assessments and ques-
tions, and by providing all students’ answers, not only the latter ones;
• Create a web-interface that interacts with the developed application to provide information to in-
structors;

The tool developed, to fulfill all these requirements, was named *DevEval*.

On a final note regarding the objectives it is important to refer that security, although it is always
important, was not the main focus for this project.

### 3.1 Architecture

Figure 3.1 shows the overall architecture of *DevEval*.

The server controls the integration between all of its modules. Both the Authoring and the Delivery
modules have two parts: one that is an interface for external communication and the one that is in the
server and deals with all the data provided by the first. In terms of the Assessment Manager, it deals
directly with an Academic Management Software, where it retrieves information about courses, students
and assessments’ dates.

Because of the reasons listed in the previous chapter, it was decided not to use existing tools to
develop this system.

The Academic Management Software chosen to be interacted with, in *DevEval’s* implementation is
*Fenix*, from *Instituto Superior Técnico*. The server was built using *Spark*, a Java 8 framework for creating
web applications. [50] For the applications examples, that would provide a visual demonstration of the
server capabilities it was created a plugin for *Eclipse*, an *IDE*, and the server was also prepared to
receive HTTP requests and so any browser can be an application that communicates with the *DevEval’s*
server. The *Eclipse* plugin is the interface for students while the web-interface is for instructors.
In terms of requests, the server answers to HTTP, either those related to the designed webpages, like images, javascripts or css, and JSON ones. In terms of code organization, it was distributed in four sections:

- **HTTP GET**;
- **HTTP POST**;
- **JSON GET**;
- **JSON POST**;

### 3.2 Data Model

Since the objective was to build a LMS tool - with a different focus on the types of assessments but a LMS tool nevertheless - the data model was designed to be similar to the existent models. Figure 3.2 illustrates the data model scheme.

*DevEval* is built to be integrated with IDEs and so answers will always come from external tools, which means that anyone that wants to built an application, to interact with the server, has to build the answer message to be compliant with it.

The assessments should be constructed with:

- a string "title", which defines the assessment’s title to be presented to the students;
- an integer "quantity", which should define the number of questions present in the assessment. This value is a security measure, in order to verify possible data corruption, since it has to match the size of the next field;
• an array "questions", where the questions themselves are stored. Similarly to the array "files", each element, that represents one of the questions, has two fields:
  
  – an integer "id", which has the id of the question (the first question starts with id zero);
  – a string "text", which contains the question itself;

The answers should have:

• a boolean "final", which must be set true when it the student finishes his assessment;

• an integer "linkage", which should contain the question id, pertaining to which answer it was sent from;

• an array "files", where the code is stored. This array has as many elements as files the student has in his IDE with code. Each element, that represents one of those files, contains two fields:
  
  – a string "filename", which should contain the filename that the student specified;
  – a string "content", which should contain all the code/text present in said file, already escaped (the ASCII representation of characters when present special cases like the code for the paragraph, "\n", or the tabulator "\t");

### 3.3 Implementation

As it was mentioned previously, DevEval is a server which provides information for two types of clients: webpages and any other application that can preform HTTP requests. This section will present the three systems developed: the server and the two clients.
3.3.1 Server

In section 1.2 were listed all the requirements necessary for a tool to solve the proposed problem. DevEval’s server was built based on those premises.

First and foremost, let’s explain how data is stored.

Since the core part of data used are files - like students’ answers or assessments - the Database was setup and created based on directories.

This means that each course has its own directory. Inside each of those directories are listed directories for each semester where the course had any assessment created with DevEval. In each of those are listed directories for each student enrolled in the selected course in the selected semester.

![Diagram of server's directories layout](image)

Figure 3.3: Example of the server’s directories layout

The first time a instructor accesses a course in a specific semester, the server automatically creates the directory for that semester, contacts Fenix in order to retrieve all the enrolled students, and creates a directory for each one. This is done on the premise that every enrolled student intends to be assessed on the course. If some student end up not being assessed, the only difference would be that it was automatically created a directory for them that was not used.

Inside each student’s directories are listed directories for each assessment the student already performed, and inside each one of those are the answers the student gave during his resolution of the
assessment.

Inside every semester directory, there is also another directory, besides the enrolled students, named "Assessments List", which contains all the assessments files created for that course in that semester.

This layout can be more easily understood by analyzing figure 3.3 (the values presented are examples just for ease of understanding). Each box represents a directory and the paper sheet with the corner folded represents files.

The way DevEval's database was built is similar to a MongoDB [51] one, which stores data in the form of documents. The core fundamentals are the same. The documents MongoDB stores are similar to JSON with name-value pairs, but in reality are BSON (a binary representation of JSON), and are analogous to structures, in programming languages.

Because of these similarities, it could be possible for a developer to integrate a MongoDB by recurring to DevEval's API to obtain the files. This way, instead of them being stored in the default database, that is directory-oriented, they could store the data in a MongoDB.

It was mentioned in section 3.1 that this server was built using Spark. This framework has the core connection commands and functions already created and so it is only necessary to call them when needed. This simplified the process of server creation since it was not necessary to be worried with multiple thread creation, multiple connection listeners or other base server-client connection specifications.

The server addresses HTTP requests and was built to be RESTful (this point is further developed in section 3.4). As stated, the server answers either with HTML or JSON, the first one when dealing with web-applications and the other for applications like an IDE's plugin.

Before the server answers any request, it verifies if the user making it is already authenticated and with the correct permissions.

From this point onward, it is important to take into consideration that since the server implements resource identification through URI, every information the server needs to navigate through the database is obtained that way.

The next module implemented was the Assessment Manager. This module deals with all the requests related with acquiring assessments, questions, answers, etc. In terms of code, this part is the one that interprets the requests and verifies if the item requested is available and if the user has access to it.

As mentioned in section 3.2, these items are all stored as JSON files. Every answer in JSON format that the Manager creates concatenates a boolean "success" parameter with the information that was requested, that is already in JSON (e.g. the title and number of questions of a specific assessment; the question id and text from a specific question). If by one of multiple reasons - like problems connecting with Fenix or failure to read the requested file - the server is unable to respond in accordance with what was requested, the answer message will have the "success" field as false, and it is defined another parameter "reason", a string, that describes the problem that occurred and why the server failed to answer as expected.

When an assessments list is requested (by any interface for students), the Manager will start by verifying which of the known assessments are occurring during the current semester. The next step is to
verify which of those assessments date frames are within that day's date. Only after this filtering process is the resulting list sent.

The Authoring tool is part of the same component that the Manager is. It is responsible for translating the data received from the web-application, when creating a new assessment, into a JSON file. It then sends the file back to the central component which deals with it appropriately.

The Delivery System combines the capabilities of both of the two previous components presented. It receives the requests, interprets them, and sends a message to one of them, depending on the request. This system is directly responsible for the responses to every HTML GET request.

When dealing with JSON GET requests, this component either deals with them or sends the process to the Manager tool. As mentioned, when a student's application requests a list of available assessments, the Delivery system sends that process to the Manager, but when it is requested a specific assessment, it deals with it itself.

When an assessment is requested, after the usual authentication verification, the Delivery system will start by communicating with the Database, by grabbing the specified assessment. Then it opens it and retrieves the amount of questions existent and sends that information to the client application. It is supposed that right after that request follows another one requiring the first question - the one with id zero. Whenever a question is requested the Delivery system will once again communicate with the Database to retrieve that specific question, sending it in the format described above - concatenating the "success" boolean field with the question itself.

There is also the possibility of receiving a JSON POST request. This type of request is also dealt with by the Delivery system and it is the only way any application can send a student's answer.

When one is received the system will firstly verify if the student's application has already sent any of its answers to the server, regarding the selected assessment. If it did not it will create the new folder for that assessment in the student's directory (taking in consideration the hierarchy presented in figure 3.3). After that it will verify its integrity and then sends it to the Database to be stored. The default name for a new answer file is a defined string value, plus the related question number, plus the submission counter value (e.g. "submission-1-0", would be the first submission for the second question, since questions ids and the counter start with zero). If, in the previous verification, there was none answer it to this assessment, then it will be stored with the counter value as zero. Otherwise, the system will verify how many files, related to that specific question, have already been stored, incrementing the counter with each one. Since the answer already comes in the JSON format, the Delivery system only needs to send it to the Database with the correct path to be stored within. E.g. "Database/CourseA/14-15-Sem1/Student12345/Test1/submission2-1.json" is the file’s pathname for the second answer to the third question of the assessment "Test1", by the student with id "12345", enrolled in the course "CourseA" in the first semester of 2014/2015.
3.3.2 Students’ Client

As previously mentioned, the server is prepared to deal with two types of clients: webpages and applications that can communicate through HTTP. To exemplify the capacities of DevEval was built an Eclipse plugin. The other client interface is accessible through any browser.

For the student’s client, the main objectives are that it allows students to login with their student id, to choose an assessment from those that are available, have the questions from the chosen assessment displayed, and to be able to solve them in an IDE.

Since Eclipse is an IDE and that was a core part of the problem (integration with IDEs), it will first be described how this example application was built to interact with the server.

Eclipse ended up being the chosen IDE because of all the support it has for plugin development. It has a vast, well developed and documented API which allows the capture of the content in all the files being used in the IDE.

When a student opens Eclipse, if the developed plugin is installed, its icon will appear on the IDE’s toolbar, as showed in figure 3.4.

![Figure 3.4: DevEval’s plugin toolbar shortcut](image)

After clicked, it will be prompted a login menu, as seen in figure 3.5. As mentioned earlier the focus was not on maximizing the security of the system and since this is a simple plugin just to prove the functionality of the developed server, the authentication system is a simple one, although it is required that the student logs with his Fenix id, otherwise the server cannot verify if he is enrolled in any courses. Besides that, it is in the best interest of any student to solve the assessment with his own id, otherwise he will not be graded. Verification of any student tries to pass by another does not compete to the system. It has to be done presently by the vigilant instructor.

When the authentication is successful the user will presented with a list of available assessments. In section 3.3.1 was mentioned the process of obtaining this list. It is assumed that when it is presented to the user it is already limited. Figure 3.6 illustrates this list.

Whenever a student selects an assessment he cannot perform, a warning message will appear, has shown in figure 3.7. Whether he clicks in the “Ok” or “Cancelar” button, he will not progress further in the assessment’s presentation.
This happens if the student tries to solve an assessment for a course he is not enrolled in. To facilitate the choice, since assessments from different courses can have similar names, in front of the assessment's name is detailed the course's abbreviation from which that assessment belongs.

When the user selects one he is able to perform, it will be then presented the first question of that assessment. An example is illustrated in figure 3.8.

At that point is expected that the student starts to solve the presented question. He will create a new file, where is coded solution will be written. Whenever he feels like he has solved that first question he has the possibility to go forward to the next one. This is exemplified in figure 3.9.

After he clicks on the “Seguinte” button two things will happen in the background:
The plugin will send all the open files’ contents to the server. This happens recurring to *Eclipse’s API*, it allows the plugin to obtain all the content from all files open in all the windows in all the open workspaces. This assures that no file is left behind, except if the student removes any from the *IDE’s* interface. If this happens it is assumed that the student does not want that file to be assessed. The format with which the files are sent was described in section 3.2.

• The plugin will then proceed to request the next question. This process is equal for every question request. It is done through an *HTTP request* for the *JSON* question selected.

Figure 3.10 shows the display of the second question, allowing the "Anterior" button to be pressed,
since it is no more being presented the first question, which has no question before it, thus do not allowing the “Anterior” button to be pressed.

Every time the user presses one of the two buttons those events will always happen, independently of which question is being displayed to the user. This gives the student’s code history to the server so the instructor later can watch it.

When the POST request with the file’s content is sent to the server, the url used contains information regarding which question is the user answering from. What this means is that if the user is going from question one to two, then it will say that the message is regarding question one, since it was the one he “left”.

If, for example, the student wants to go back to re-code his answer to question one, while being presented question two, when he presses the back button the message will inform that it is related to question two. This facilitates comparison of code by question. It is not a bulletproof method, since students can change code from a different question without going to read it again.

When the user presses next and there is no more questions, a warning message will pop-up announcing so. Figure 3.11 presents that warning message. As displayed in that message, if the user presses “Ok”, he is agreeing that he has finished the assessment, closing the plugin application. Otherwise he can go back and do whichever changes he sees fit. Whether he chose one or the other it will
always send a message to the server in the same way that was mentioned above. If it was pressed the “Ok”, finishing the process, the boolean field “final”, in the message (following the data model presented in section 3.2), will have a true value.

![Figure 3.11: Warning of assessment's end](image)

This plugin was develop to show the potentialities of the DevEval's server. It interacts with all the REST calls available for students, since the IDE integration is supposed to be used by students and not instructors.

3.3.3 Instructor's Client

As mentioned earlier, the interface for instructors is provided through any browser, by accessing the server’s address. Figure 3.12 presents the index page. In this page there is a brief introduction to the project DevEval. On the left is the menu, that is always the same for all the website. It presents two options:

- "Inicio" which redirects the user to the index page (the one he is actually in, in this case);
- "Listar UCs" where the instructor can access a list of all the Curricular Units (courses).

Since this server has integration with Fenix, the courses presented are all pertaining to Instituto Superior Técnico. Furthermore, and for this example only, the possible courses presented on that list are all pertaining MEEC.

Whenever the "Listar UCs" option is clicked, the user is redirected to an webpage with the properties mentioned. Figure 3.13 shows an example of that webpage. The page first lists all the courses which already have been created in the database. After that there is a search box. This search box will activate a script that, from a list of all the courses active in the current semester in MEEC, finds courses that match the words written in it. What this means is that the instructor can write only one or two letters and find the course among those with those letters, or write the whole word.

In the figure mentioned is possible to visualize this, since it is only written “Pro” and a list of multiple courses, that have “Pro” included in its name, is presented. This real time query is done through
JavaScript. The browser receives the list of all the courses and as the instructor fills the search box a JavaScript process queries the list for all those that contain the written word.

3.3.3.1 Authoring Interface

For all the presented courses in that list, instructors have two options:

- Access that course database;
- Create a new assessment for that course;

If the instructor chooses to create a new assessment, he is redirected to a webpage with a form to do so. This is the part of the web-application that interacts with the Authoring tool. Figure 3.14 displays the mentioned form.

It has four text-fields that are mandatory to be filled. The first one is where the instructor names the assessment. The final is the question field, where the question (it is necessary at least one question to
create an assessment) is written. The other two are the initial and final dates for the assessment to be performed. There is a specific format in which the date has to be written (identified in the webpage), and if the instructor writes it in a wrong format, a warning will pop-up.

![Empty new assessment form](image)

Since all the four fields are mandatory, while all of them are not filled there will be warnings whenever the user tries to submit the assessment. One verification made is if the initial date is previous to the final one. Figure 3.15 shows that warning.

All the warnings are in the same form as the mentioned in the previous paragraph. It can say “The assessment's name is yet to be defined”, or “The initial date is not defined” or even “There are no questions asked yet”. The point is that no incomplete assessments can be created.

![Warning shown whenever the initial date is posterior to the final date](image)
If the instructor wants more than one question he can press the button on the bottom left, “Nova Questão”, which will add a new question text-field. He can do this action as many times as he wants, giving the possibility to create an assessment that has all the points in just one question, a singular complete enunciation, or to spread all the points between various questions.

In figure 3.16 is an example of an assessment form filled.

Whenever the instructor feels he has finished his assessment, he just needs to click the “Submeter” button that will verify if all the requirements are filled, and then send the assessment information to the server, to be translated and stored.

![Filled new assessment form](image)

**Figure 3.16: Filled new assessment form**

### 3.3.3.2 Visualization Interface

The other possibility, in the courses list page, is to select the “Aceder” button, which will lead to access the database of that course. The user is then directed to a webpage which lists, for the selected course, all the semesters which have material in the server. If there is no semester for that course in the server, then it is created the base for the current semester. In figure 3.17 is shown a example of a list of semesters for the course “Programação”. Each course has in its front another “Aceder” button that gives the access to that specific semester.

When that button is pressed the user is redirected to a webpage that, as mentioned in section 3.3.1, in this level of the directory hierarchy, it is presented a list of all the created assessments and also a list
of all the enrolled students, both relative to that course in that semester. Figure 3.18 presents those lists for the course "Programação". As it has been done until here, every option of the list has a button to access that specific option.

If the instructor chooses one of the listed assessments, he will be redirected to a page with a list of all the students that have already answered that assessment, as it can be seen in figure 3.19.

On the other hand, if the instructor selects one of the students, it will present a list of all the assess-
ments that student has already performed, as seen in figure 3.20.

![Figure 3.20: list of assessments the specified student solved](image)

This gives instructors some flexibility, since they can go either way and obtain the same end result: they will get the visualization for the chosen student in the chosen assessment. It goes "full circle", whichever way the instructor started, he will ending with the option to visualize the students code history.

It is important to mention that, although instructors can do all this process through webpages and clicking buttons, links, etc., they can also do the same actions through URIs. Since the server was implemented in a RESTful way, it is possible to do so. If one were to verify each one of these figures, in this section, one would notice that each one of them has a different and identifiable URI.

Although it is possible, it is easier to just go with the flow of the webpages, since some URIs need specific parameters.

The only case where this does not happen is with the visualization page. All the visualization process occurs within one page, resorting to JavaScript to implemented the needed processes. The original page for the visualization, reached from each option of the previous two mentioned, has a small introduction that explains how the visualization tool works. This introduction is displayed in figure 3.21.

![Figure 3.21: Introduction of the visualization webpage](image)

When the visualization process begins the page is divided in two: a first part pertaining to the questions, and a second one pertaining the answers.
Both parts have four items present:

- The numbering of the action - number of the question for the first part, and number of the answer for the second;
- A text-field (with read-only properties) that presents either the question selected or the answer, depending on which part it is presented;
- A next button that will lead to the next question, if pressed the one in the top part, or to the next answer, if pressed in the bottom part;
- A previous button, that works in similar fashion to the next button;

Figure 3.22 presents exactly this: It starts with the first question and since there is only one answer, it is not possible to click neither button on the second part of the page.

![Figure 3.22: Presentation of the answer for the first question](image)

By default, the visualization will start by presenting the first question in the top half and the latest answer in the bottom one.

Since it begins with the first question, it is not possible to click the “Anterior” button, since there is no previous question. The same logic applies, in the reverse order, to the answer: since it begins with the latest, it is not possible to select the next answer. If there is only one answer, then it is not also possible to click the previous button.

This logic applies always that a limit is reached, either it being the beginning or end either of the questions or answers.
The text-field present in the bottom half starts by saying how many files there were stored in that answer. They are all separated by a line of “-”, as can be seen in the mentioned figure.

When the student clicks one of the buttons on the first half of the page, when available, it will start a process to gather from the server the selected question and after that the latest answer for that question. This process also occurs when the instructor presses the start button on the visualization introduction page.

Figure 3.23 continues the previous example and illustrates what appears after pressing the next button in the questions’ half. In this case, for the second question there are more than one answer, since the numeration of the presented answer is two.

If the user then presses the previous button on the answers’ half, there will be a connection made to the server but, unlike those made when a button on the questions’ half is pressed, it will only ask for the new answer.

The objective of visualization of, not only the final code, but also its history, is that instructors have a way to verify the process and alterations that students performed during all the process of assessment.

This can be observed by comparing the previous figure with figure 3.24 that contains the first answer to the second question of the example being used.

For ease of visualization in the image presented, it was circled the difference in the second figure.
3.4 External API

The communication used between the server and any possible clients is through HTTP. GET and POST requests where used for various actions. The server was also implemented as a RESTful web service:

- **Resource identification through URI**: the *URIs* would identify the resources needed (e.g.: to list the students enrolled in the defined course in the defined semester would be used the *URI* "/course/semester");

- **Uniform interface**: resources are manipulated using a fixed set of operations, with the use of *POST* and *GET* requests.

- **Stateful interactions**: every interaction with a resource is stateless, that means that request messages are self-contained, they have all the information needed so the server can answer correctly and the server never stores state values.

One other field of a RESTful web service is the *self-descriptive messages*, where "resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as *HTML*, *XML*, plain text, *PDF*, *JPEG*, *JSON*, and others" [8], but this was not completely applied. The *DevEval*’s server could respond to some formats but the resources dissociation was not completely present, the client would receive either an *JSON* answer or a *HTML* one.

*JSON* (*JavaScript Object Notation*) [52] is a lightweight data-interchange format. The objective is that it is simple and easy to read and write, for humans, and at the same time easy for machines to
parse and generate. It is completely programming language independent, although it uses conventions similar of many programming languages. It is built on two structures:

- A group of name/value pairs;
- An ordered list of values, also known as an array, vector, list, etc.;

The server’s component that answers a request, is somewhat decided by the type of request made, as it was explained in section 3.3.1. The possible HTTP GET requests, which answer is in HTML format are listed in table 3.1.

Whenever in the table appears a ":" it means that the value present in that path position is variable (e.g. "/HTML/PRO5517" is a correct URI for the examplified "/HTML/:uc" URI). The word after the ":" has only the purpose to be explanatory of which kind of information it is necessary to put in that position.

<table>
<thead>
<tr>
<th>URI</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>/HTML/index.html</td>
<td>Index page</td>
</tr>
<tr>
<td>/HTML/uchtml</td>
<td>Page that searches for courses</td>
</tr>
<tr>
<td>/HTML/ucslist</td>
<td>Answer from Fenix with the courses list</td>
</tr>
<tr>
<td>/HTML/:uc</td>
<td>Page with list of semesters for chosen course</td>
</tr>
<tr>
<td>/HTML/:uc/createassessment</td>
<td>Page for creation of new assessments</td>
</tr>
<tr>
<td>/HTML/:uc/:sem</td>
<td>Page that lists students enrolled and existing assessments</td>
</tr>
<tr>
<td>/HTML/:uc/:sem/:student</td>
<td>Page that lists all the assessments the chosen student has performed</td>
</tr>
<tr>
<td>/HTML/:uc/:sem/assess/:assessment</td>
<td>Page that lists all the students that performed the chosen assessment</td>
</tr>
<tr>
<td>/HTML/:uc/:sem/:student/:assessment</td>
<td>Page that provides the visualization interface</td>
</tr>
</tbody>
</table>

Table 3.1: Available HTTP GET requests that provide HTML answers

There is only a "POST" request which receives its answer in HTML format and is used by the Authoring page to send the new assessment data to the Authoring Tool, through the message’s body, which is presented in table 3.2.

<table>
<thead>
<tr>
<th>URI</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>/HTML/:uc</td>
<td>Data for creation of new assessment</td>
</tr>
</tbody>
</table>

Table 3.2: Available HTTP POST requests that provide HTML answers

Pertaining the JSON formatted answers for "GET" requests are listed in table 3.3. The logic for the names follows the explained above.

Lastly, for the JSON formatted answers, there is also only one that is a "POST" request. The name given to the first three variable path positions is self-explanatory, but for the last, what it should be put is the question for which the submission is related to. This URI is used when an application sends an answer to the server.
<table>
<thead>
<tr>
<th>URI</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>/JSON/:student</td>
<td>List of assessments available</td>
</tr>
<tr>
<td>/JSON/:student/:uc/:assessment</td>
<td>Number of questions in chosen assessment</td>
</tr>
<tr>
<td>/JSON/:student/:uc/:assessment/:question_id</td>
<td>Question text for the specified question id</td>
</tr>
<tr>
<td>/JSON/:student/:uc/:assessment/:question_id/:answer</td>
<td>Answer of the student to the selected question</td>
</tr>
</tbody>
</table>

Table 3.3: Available HTTP GET requests that provide JSON answers

The last URI is only used by the Visualization tool that presents students’ answers to the instructor.

<table>
<thead>
<tr>
<th>URI</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;/JSON/:student/:uc/:assessment/:submission&quot;</td>
<td>Data of a student’s answer</td>
</tr>
</tbody>
</table>

Table 3.4: Available HTTP POST requests that provide HTML answers

### 3.5 Fenix Integration

As mentioned earlier, since Fenix has an REST API, which allows the query of students enrolled in specific courses, courses information, personal information, etc., it was decided to integrate it with DevEval. This way part of the LMS system is already simplified.

Integrating with Fenix there is no need to build “classrooms”, which the other LMS platforms have - the name given is different between them, some call them “groups”, but the basis is the same - where the students enrolled in each course are listed. Instead the server just contacts Fenix and asks for the students list, when required. This happens, as mentioned above, either when a student tries to access an assessment - the server needs to verify if the student is in-fact enrolled - or when the instructor requires the students’ list for a specific course. The other occasion when the Fenix is contacted is when the instructors requests a list of all the existent courses.

As mentioned previously in this chapter, the security was not the focus of this work, otherwise it could have been implemented into the server the authentication from Fenix. With its API, Fenix allows the integration of applications, including its authentication system. The basics of this implementation is that the persona (has to be a valid user) that wants to create an application to interact with Fenix has a web-interface to define some information for his application, like the example displayed in fig 3.25. Then, through some functions from an available package (it is available for three languages: python, java and PHP, the application would redirect the user to the Fenix authentication system, that when correctly authenticated, would redirect the user back to the application with an authentication token.

In fact, it was tried to implement a Fenix authentication application within the DevEval serve, but there were some compatibility issues. There were some contacts with Fenix’s IT team, but since it was taking a while to resolve, and taking in consideration that the focus was not in the security of the system, it was decided to go with a simpler authentication system.

In section 3.3.3 was mentioned the listing of courses from MEEC. What happens is that the DevEval’s
server contacts *Fenix* to give a list of all the courses available in the current semester for *MEEC*. *Fenix* answers is in *JSON* format, with all those courses.

### 3.6 Evaluation

The developed tool, *DevEval*, incorporates all the objectives outlined. Table 3.5 displays the implementation of *DevEval* regarding each one of those objectives.

*DevEval*'s server is oriented for *lab examinations* and this is connected directly with another objective: the development of an *IDE* plugin, in this case, a plugin for *Eclipse*.

Relatively to the *LMS*’ core architecture, it was also build upon it. The system has a database, that is directory-oriented, as explained in section 3.3.1. Besides that, it was implemented an Assessment Manager that interprets all *HTTP GET* requests for *JSON* answers and a Delivery system that groups them all together, with its main focus on solving *HTTP GET* requests for *HTML* answers. This Delivery system is able to deliver assessments and questions to the applications that require them, as it is the case of the developed *Eclipse* plugin, as it can receive multiple answers from them, regarding the same question. Besides that it is able to deliver students’ answers, as it is implemented in the Visualization tool integrated in the developed web-interface.

It was studied the option to implement the *QTI* specification for the questions and assessments, but since there is not yet a form defined for just question presentation without the requisition of an answer, it was decided to implement a simpler model. An assessment, or question, when *QTI* compliant, is
Table 3.5: DevEval objectives verification

<table>
<thead>
<tr>
<th>Objective</th>
<th>Accomplished</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab examinations Oriented</td>
<td>Yes</td>
<td>Directory-oriented</td>
</tr>
<tr>
<td>Database</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Authoring</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Assessment Manager</td>
<td>Yes</td>
<td>HTTP GET requests for JSON</td>
</tr>
<tr>
<td>Deliver assessments</td>
<td>Yes</td>
<td>HTTP GET requests for HTML</td>
</tr>
<tr>
<td>Deliver students’ solutions</td>
<td>Yes</td>
<td>HTTP POST requests for JSON</td>
</tr>
<tr>
<td>Receive multiple answers</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Integration through REST</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>IDE Plugin for students</td>
<td>Yes</td>
<td>Plugin for Eclipse</td>
</tr>
<tr>
<td>Web-interface for instructors</td>
<td>Yes</td>
<td>Webpage</td>
</tr>
</tbody>
</table>

represented through XML. Each element has a set of rules that can be applied, concerning the answer and the scoring. Since the elements needed for DevEval do not require neither of those rules, the binding with which the data will be represented fell to JSON. This decision took into consideration the fact that the server is to be RESTful.

Part of that decision came also for simplifying the construction of future applications, that anyone can build and that would communicate with DevEval’s server. In the existing LMS platforms, all the elements are created by the tool itself, even the answers. The presentation of both questions and answers is simple for the user, but in the background the Delivery system, of those platforms, builds it into a element that is QTI compliant.

Both the IDE plugin and the web-interface where widely explained and demonstrated in their own sections, but the drawn conclusion is that they implement clearly what was requested of them.

Students log in the plugin, choose an assessment and are given the questions of that assessment, one at a time. Without them having to perform any special action, all their files used to answer the questions are sent back to DevEval’s server.

Instructors within the web-interface, search for the course they want to interact and have two options: either they create a new assessment for that course, which is done through a pre-defined form, that instructors only have to fill, or they can go and visualize students’ answers to a specific assessment. This is done through the Visualization tool that presents them, for a selected assessment and student, the answers the student gave along with the question. Since the number of answers given per question can be more than one, instructors only have to use the presented interface, that is simple and intuitive, to visualize the different answers available.

In the end, the develop tool, DevEval, fulfills the requirements necessary to solve the proposed problem, which where the objectives outlined for it. It has an API for external tools to integrate with, is
integrated with an Academic Management Software, in order to obtain information for courses, students and assessments dates, and, not least important, it allows the history of students’ code.

3.7 Reuse of existing LMS

As mentioned in section 2.4.2.1, TAO was considered the best tool to start with in order to achieve a solution for the proposed problem. This would be done by creating a plugin for TAO that would allow the interaction with IDEs and at the same time to store more than one answer per question.

To start developing this plugin we first looked into TAO’s forge website since there are tutorials on how to use their API. Soon it was discovered that, as mentioned earlier, the Delivery System of TAO was one of the default modules that had not yet the REST API available. This came as a setback but since TAO is an open source platform, we could try to mingle the existent REST APIs with the simple API of the Delivery system.

The first thing tried was to create a plugin that would try to call all the necessary modules in order to achieve what was required. The idea was to request simple API calls from the Delivery system module and work around them, but soon another problem appeared: the plugin integration was not completely developed; it was well prepared for CAT (Computer Adaptive Test) plugins integration, but since what was needed was not a CAT plugin but querying existing modules without administrator privileges (since it would be the students that would indirectly call this plugin), there were limitations that put the plugin creating out of the question. Even with administrator privileges there were still calls that did not respond as expected.

With that in mind, to achieve a solution through a different path, and taking in consideration that the documentation was not detailed enough, was necessary to read source code, test and try and interpret HTTP requests in order to understand how the platform worked in a normal usage. These mechanical tasks consumed a great portion of time but in the end was found a solution to the lack of REST API of the delivery system: we would change the source code in order to create REST calls that would provide the necessary information as the HTML counterpart did.

When trying to implement those calls and analyzing the HTTP requests needed, since the beginning of a test until the end of the process, was noted that the default calls made were requiring multiple modules of the platform, thus making it necessary to change not only the default Delivery System module but also multiple others.

Still, those new REST calls were made in all the necessary modules, but this was achieved greatly through hardcode operations mainly extracting information directly from HTTP code, since the platform did not provided it in any other way, in the level of depth that we were working on. As consuming as the analysis of this layer’s code was it was unthinkable, with the time frame in hands, to go read code from the base of the platform to obtain the information without being in HTML.

In the end all the calls were working and was possible to present, through REST ones, tests and items, leaving only the answer storage part to complete. This point, that showed once more the rigidity of the existent Delivery System, did not allow multiple answers per question.
With the past experience it was even thought about try and work around that problem, to look a bit further and try to find a way to modify the system to allow it, but in the end and along with the fact that to submit the developed "extension" was necessary to script changes in multiple default TAO's modules, all these problems that arouse led to giving up on TAO as a platform to implement a solution for the proposed problem.

On a final note the newest version, TAO 3.0, became available after it was decided TAO was not the tool expected, and it is not labeled as Beta version anymore. At that time and with all the problems that appeared during development of the necessary tool, the decision to leave TAO, as only a negative result, was maintained. This information was only obtained after the development of DevEval.
Chapter 4

Conclusion

When one looks to the existing tools related with e-learning, the numbers increase every year, the technologies implemented increase also and the focus becomes wider.

But there are still some blind spots, some lesser common assessment types that still are overlooked. Lab examinations were one of those. Existing LMS tools were not prepared to help evaluate programming assessments. They do not allow integration with IDEs, the primary tools used when teaching programming.

When studying which existent LMS platforms could be helpful, the conclusion was always the same. Although some platforms are more prepared for some types of integration, none is prepared for integration with their Delivery systems. They may allow integrations with the Authoring system, or the Import/Export one. They may allow integration with the Grading system, but they do not allow integration with the Delivery System, which is the system required for obtaining the assessments and questions data to be used in the IDEs.

The first approach was to try and create a component for one of those platforms, TAO, that would interact internally with the Delivery system and to do the needed functions to obtain and deliver assessments and questions from the platform to the IDEs and at the same time to receive the answers and store them in the Database. That task proved to be unsustainable, since it was necessary to change core components and functionalities of the platform to allow what was pretended. Even then more problems were arousing when it was necessary to implement a way to allow multiple answers to be stored, per question.

These platforms are great for their focus, but it is hard to mold them in order to have a different focus.

The other type of tools evaluate have a focus on lab examinations, but are focused solely in projects, which have a long duration. There was none that focused programming assessments with small duration.

A lot of the tools studied have great ideals and perform great actions, but none focus on the delivery and integration of programming assessments with IDEs.

With that in mind, the decision passed by creating a tool that would have those objectives in focus. Starting with outlining the objectives for the tool, it was looked to the problem definition and was decided that it was necessary that DevEval would be built as a RESTful server, thus allowing the inte-
egration with applications like IDEs. This was one of the points where existing LMS platforms failed.

DevEval was built to facilitate assessments authoring, providing an HTML interface for that task (similarly to what is done in most LMS platforms) and to have a Delivery System that allows storage of more than one answer per question. Since the database is directory-oriented, it is easy to identify which answers correspond to each student and each assessment he solved. This system is also able to send assessments and questions data to applications that so require - as the developed plugin does.

In the end, the system is prepared to solve the presented problem, focusing on a blind spot that other e-learning tools have not focused on, thus providing a solution for instructors that want an e-learning tool to help with lab examinations assessments authoring, delivering and visualization.

The developed tool could have a few more features, like those mentioned in section 4.1. It would improve its capabilities to implement a better authentication method or to integrate one of the many automated assessing tools or even to simply apply a differential between answers, in the Visualization tool.

4.1 Future Work

There are some features that easily come to mind that would improve this tool. In this section will be mentioned some of those features.

As it was just mentioned in the previous section, achieving an authentication system through Fenix would be more secure than just using the implemented simplistic authentication system.

Although it is not in our interest to build QTI compliant assessments, since QTI is the standard, if a QTI importer were to be implemented it could facilitate the use of this tool by instructors that already have assessments created in QTI compliant platforms, like Moodle or TAO.

In section 2.4.3 were mentioned a multitude of tools that can assess in an automated fashion programming assessments and projects. Integration with one of these tools would facilitate even more the workload present on instructors. With the field evolving every year, a conjunction of these two types of tools could be the future to help instructors on lecturing programming courses without having to evaluate individually hundreds of assessments each year.

Last but not least, it could be important to implement a comparison between answers for a specific question. This could be accomplished by adding a new part to the Visualization tool, where there would be presented in a text-box (with read-only properties) a comparison between the selected answer and the immediately previous one.
Bibliography


[31] TalentLMS. Talentlms webpage. "talentlms.com/index". "[Online; last accessed 08-10-2015]".


[34] CA Clarenc, SM Castro, C López de Lenz, ME Moreno, and NB Tosco. Analizamos 19 plataformas de e-learning: Investigación colaborativa sobre lms. grupo geipite, congreso virtual mundial de e-learning, 29-chacón-rivas, m., & solano fernández, i.(2009). Modelo de Calidad para la Evaluación de una Plataforma LMS.


