Faculty of Science and Engineering

Improving the Usability and Scalability of a Student Assessment System

by

Ivan Giancarlo Rodriguez Asqui - 43365914

A project submitted in partial fulfilment
of the requirements for the degree of

Master of Information Technology

Macquarie University

2015

Supervisor

Dr Christophe Doche

Department of Computing

ABSTRACT

The Department of Computing at Macquarie University uses a Randomized Quiz System designed to evaluate the understanding of basic programming concepts by students in their first year computing courses. Even though this application has been in use for several sessions, it was determined that the current implementation was still lacking in some features and it needed changes to be able to provide the business value it was originally designed for. The aim of this project was to implement a series of improvements to the system so that it can fulfil its potential as a pedagogical tool for the lecturer and a self-learning tool for the students. These changes were gradually implemented using an iterative development process and test driven development techniques. We used virtualization to set up a server environment with similar characteristics to the target platform. We created a test harness to be able to analyse the current state of the system, document it, avoid the introduction of any bugs after our changes were completed, and to provide a baseline for new developers studying the system. Then we developed new modules with the objective of improving the usability of the system and to provide more business value to its users. Finally we measured the performance of the system and made changes in its architecture with the objective of a better response time and an overall improved user experience. The results are very promising, as now lecturers have a very user friendly editor that they can use to produce quickly prototype and test programming questions, and the new architecture of the system provides a significant improvement in response time and performance of the application. Also, the changes in the architecture and the extensive test suite provide a foundation for future developers to further improve and extend the system.

Table of Contents

List of Figures

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Dr Christophe Doche who helped me to define the scope of the project, provided me with the initial material and all the information required to get started, as well as guidance, direction and advice throughout the development of this project.

Also many thanks to Dr Peter Busch who helped me with a continuous set of milestones and feedback cycles that ensured the progress and completion of a comprehensive and accessible final document.

In general I would like to acknowledge all the instructors from the Department of Computing at Macquarie University whose knowledge and expertise delivered in the lecturers I had the privilege to attend contributed in one way or another to the techniques and ideas used in this work.

I would also like to thanks the Secretariat of Higher Education, Science, Technology and Innovation of the Republic of Ecuador for awarding me the scholarship that gave me the opportunity to visit this beautiful university which contributed with my personal and professional development.

My deepest thanks to my family for all their support while I was away from home. Even though the geographical distance and time zone differences where overwhelming, my mother, my father and my sister provided me with sufficient encouragement and peace of mind to be able to success and give my best in this challenging and rewarding experience.

Chapter 1: Introduction

1.1 About the student assessment system

The student assessment system used as a base for this project is called the Randomised Programming Quiz System (Evans, 2010). It is a web application developed in PHP and deployed on a standard LAMP stack (Linux, Apache, MySQL, PHP). This quiz system was developed to help students understand and practice programming concepts, and to inform the educator to which extent these concepts are being understood. The project was started in 2010 with support for programming questions on C++. The project became available online on GitHub on 2012, and by this time it already had support for Java programming questions (Evans, 2014). Some work was made on 2013, but starting on 2014 there has been no activity on the project up until the start of this study on 2015.

The way the system works is that students log in to web application and start a new quiz session. When the quiz is started, a programming question is generated and displayed to them for each step in the quiz. To pass the test, the students have to predict the output of the program. When students submit their answer, the system compares it with the already precomputed correct answer, and if there is a match, the student completes this question and moves on to the next one, until all the questions of the quiz are completed.

Unlike systems based on a predefined set of questions, this system uses question templates created by the lecturer, which are essentially parametrized Java programs. So, before students access to the system, lecturers have to create and upload these question templates to the system. Within these templates there is information about the question and the programming question itself. They are based on standard Java programs, except that portions of the code have been replaced by placeholders, as illustrated in Listing 1.

```
public static void main(String[] args){
    int `va1s1` = `numbers`;
    int `va1s2` = 2 `op` 3;
    if( `va1s1` < 0){
        System.out.println("`va1s1` is < 10");
    }else{
        System.out.println("`va1s1` is > 10");
    }

  }
```

Listing 1: Extract of a sample question template.

The symbols between backtick (`) characters indicate placeholders. The lecturer associates each placeholder with a set of possible values or an expression that will be evaluated to obtain a concrete value. The substitution process takes place when a student access the system and requests a question. This creates a new programming question. Results of this generation process are in Listing 2 and Listing 3.

```
int foo = 3;
int x = 2 + 3;
if( foo < 0){
    System.out.println("foo is < 10");
}else{
    System.out.println("foo is > 10");
}
```

Listing 2: A generated question.

```
int bar = 7;
int y = 2 - 3;
if( bar < 0){
    System.out.println("bar is < 10");
}else{
    System.out.println("bar is > 10");
}
```

Listing 3: Another generated question.

This process has not received any major change since its inception. However this leads to one of the several usability problems discovered in the system and that will be discussed in the following sections of this report.

1.2 Motivation

At the moment Dr Christophe Doche, Head of the Department of Computing is the only lecturer using the system. We had several meetings to discuss the aims of the system, its current state and the shortcomings he has noticed after using it in previous courses. He commented that in some aspects the application was difficult to use, did not work as expected, or requirements have changed since its original design years ago. After these consultations we collected information about the application's features, how it is used, and areas of improvement. As mentioned in the previous section, the typical sequence of events during the development of a quiz is the following: a lecturer loads a groups of question templates to the system, creates quizzes based on these questions, and students are requested to solve the quizzes. This process appears to be straightforward but there are some caveats.

During the creation of new questions, the system cannot be used to design quizzes with all the types of questions that it was initially designed for. The expected types of questions are: multiple choice, output and fill in questions. But only the output type is in use. The other types present bugs or were not implemented at all. This limits the types of problems that can be included in the quizzes.

The authoring process of new questions presents a barrier of entry for new educators. The questions are designed creating Extensive Markup Language documents (XML), using a custom set of XML tags. These tags are used to enclose question information like concepts covered, instructions for the student, estimated time, difficulty and the source code of the problem. This source code usually contains characters that conflict with the syntax of well-formed XML documents (W3Schools, 1999). For example, in Java source code, relational operators and generics are expressed using angled brackets (the < and > characters). These characters are also used by XML parsers to identify the beginning and the end of XML elements, so there is a semantic conflict as XML parsers are unable to tell a tag delimiter from a Java symbol, producing errors. To avoid this problem, Dr Doche has to replace these conflicting characters with their equivalent XML entities. For example, instead of the relational operator <, the entity `<` is used instead. As the number of replacements increases, the legibility of the original code diminishes, making it difficult to understand. When the author finishes editing a question using an external editor, he has to upload the file to the web application directory and import it to the system. Once imported he would test the generation of a programming question and if everything went well the process would end

there. However, when there is a problem, the system would simply fail saying that there was a problem, but no significative error message is provided to determine where the error occurred, which could be in the XML syntax, Java source code, or a combination of substitutions that made just that particular generated code impossible to compile. After a time consuming debugging session that required to repeat the steps of edition, uploading and importing, the template is ready to use. This makes the authoring process slow and inefficient.

Another characteristic of the system is that at the moment it is only used to perform a basic diagnostic assessment of the student's knowledge. This is the only assessment performed during the session. This is not what the system was designed for and represents an inefficient use of a resource that could provide valuable information to the educator as the course progresses. Ideally the system should be used to assess the students every week, introducing and testing new concepts according to the outline designed by the lecturer.

On the topic of continuous assessing of students, it also was determined that there is no way to group the quizzes in a logical sequence. Quizzes themselves are groups of questions, but there is no connection or prerequisite order between the quizzes other than the unit they are associated with. So in conversations with Dr Doche, the ability to arrange quizzes in a sequential order with clear dependencies was requested.

To be able to support a long term sequence of quizzes we have to ensure that the system can perform properly during a complete semester. This has been a matter of concern for the current system, and as it was explained earlier, Dr Doche has opted to use the system only for one assessment task. Usually during the development of software projects issues like performance testing are left to the end or not planned at all (Molyneaux, 2014), therefore we determined that a proper set of performance and stress tests were required to understand how the system would behave under extended use.

1.3 Aims and Significance

As the previous exposition indicates, we have determined that the system has value for the Department of Computing but it could be improved. As mentioned by Conole & Warburton, computer assisted assessment have become a very common approach to assist students in their learning, and the technologies that enable these assessments will continue to evolve and contribute to the learning experience of students in the modern classroom (Conole &

Warburton, 2005). So the current system may have provided good enough functionality when it was originally designed, but time has passed since the original assumptions were made, and after several real life tests its limitations have been exposed. During this time the technological landscape has changed, new architectures have emerged, others have matured and new best practices are now available. Therefore it is important not only to address the current usability problems in the system but also examine the architecture and prepare it for future enhancements.

The significance of these changes is that the Department of Computing can benefit of a reliable system that can be used to better evaluate students and create quizzes designed not only for a diagnostic assessment task but during the entirety of the session as a formative and summative assessment.

We propose that the manual edition of XML files should be avoided or minimized, and that the creation of question templates should be better integrated within the web application. This way lecturers can be more productive, as they could be able to rapidly experiment with combinations of substitutions and expressions, and quickly visualize the results of their experiments. The XML format used to store the question templates is an implementation detail that should not distract the lecturers from this authoring process. So a rich, high level editor was designed to reduce this barrier of entry for new and existing lecturers.

Students exposed to an outline set of quizzes during the entirety of their session can feel more engaged and motivated to continuously interact with the quiz system in order to practice and explore new concepts and obtain better results on their examinations.

As for the maintainability of the project, it is important to have a set of automatic tests that cover at least the critical parts of the system. With this test suite in place it will be possible to make modifications to the application in the future without breaking existing functionality. Not only it is important to verify that the application works correctly, it should also have a good response time, otherwise users will not have a good experience when interacting with it. Therefore a set of load test were performed on the system to obtain a baseline to verify the results of the changes we developed during the project (Cecchet, et al., 2011).

We proposed that some parts of the system, in particular the Java compilation and execution process, should be performed on a specialized service suited for that specific task, and not on the web server. We built a prototype of this proposed architecture and obtained good results.

As a result of this decoupling, future proof scalability of the system could be explored by moving some of the decoupled services to different computing nodes (Almonaies, et al., 2011).

1.4 Methodology

The following steps were performed during the project:

1. A code review of the existing source files. Even though we started with an existing code base, it did not have a test suite that allowed us to make changes confidently without breaking other parts of the system.

2. The careful and continuous development of functional and unit tests covering the critical features of the system. During this stage, some of the existing current bugs were identified and isolated in specific tests cases.

3. For the implementations of the proposed changes in the user interface of the system, a series of prototypes were developed iteratively in several meetings with the project supervisor until the desired functionality was achieved. In every iteration a complete working system was delivered.

4. For the improvements in the application performance and scalability, we investigated how to measure the performance and response time of the system. Only after that we could gather performance information, and identified the slowest parts of the system. We explored alternatives to improve the performance of the application and compared the results of the new approaches.

5. Similarly, opportunities to decouple task intensive sections of the code and distribute them in independent processing units were determined. For instance the compilation process during the question generation was moved to an external services that could be invoked from the main application.

During the entire development of the project and at the same time as utilities and libraries were identified and used in the system, a server configuration management script was developed to provision a virtual machine with all the required components to reproduce the target server environment. This enables further ease of maintenance and development in the future.

1.5 Outcomes of the project

The following outcomes were achieved at the end of the project:

- A user friendly web based editor that facilitates the creation of question templates for the quiz system.
- A quiz dependency system to be able to make certain quizzes dependent of others. In other words, implementation of a prerequisite system, so certain quizzes cannot be started until a determined set of quizzes is completed first.
- Implementation of a student progress display that quickly visualizes where the student is in the planned quizzes outline designed by the lecturer.
- A more organized codebase with simple setup instructions for new developers.
- A reproducible development and testing environment that can be used in any host operating system using virtualization.
- A reference implementation of the Java compilation process decoupled of the main web application, with notes about future expansion of this model.

The process of how these outcomes were achieved will be developed in the following chapters of this report.

Chapter 2: Related Works

In this section we will examine some of the previous works related to the area of computer aided assessment systems, their approaches and findings. We will also discuss some of the principles used in this project to be able to modify the existing quiz system and improve its architecture.

2.1 Review of Quiz Systems

As indicated by (Ellsworth, et al., 2004) the use of quiz systems is not a new idea. In Ellsworth's case, the Quiver System was used as a teaching aid and evaluation tool for software engineering courses using Java and C++. This system was not integrated with an authorization service, so students had to be loaded to the system manually. The administrators defined quizzes importing Java or C++ source code that acted as the correct implementation and used for verification. They also include a description and test cases. When a student selects a quiz, they are required to read the instructions and are provided with test cases. They have to write and compile their code, and once completed, they submit their answer for verification. This latter process is performed using a unit testing library. This consisted in comparing the output of the correct code, with the output generated by the student's code. It is interesting to note that as early as this work, automated testing was being used as a verification tool for student submitted code. Sessions were performed in a closed lab environment. The results were that students who used to develop code that only worked for "normal" cases, were challenged with additional "limiting or abnormal" cases that were designed by the instructor. This helped them to improve their programming skills.

Moving forward we find the work developed on QuizPACK (Brusilovsky & Sosnovsky, 2005). The system generates and evaluates parameterized programming questions related to the C programming language. QuizPACK is focused in what they call code-execution questions, which essentially means students have to predict the output of a given program presented to them. Randomization is achieved by the parametrization of certain numbers that are part of the body of the question. On each student session, the system generates the question, accepts student input and provides feedback to the student. All this through a web based interface. The system is designed as a self-assessment tool that students can optionally use to improve their programming skills. The authors of QuizPACK found that students that used the system benefited from the practice and performed much better on their exams and

obtained better grades. On the other hand, being voluntary they found out that not many students were using the system, so they modified the web interface to make it more usable. The created a wrapper around the original QuizPACK and called it QuizGuide. This composite application utilized visual cues to help students to determine what quizzes to take. They call this adaptive annotation technology, which was implemented by providing a navigation bar on the left side of the screen with a list of programming topics like loops, conditionals or logical operators. They found out that students using the QuizGuide version of the system were much more motivated and actively using the system compared to those that just utilized the QuizPACK version.

IRONCODE (Bailey, 2005) is a type of exercise that takes a different approach in programming quizzes, focusing in the code-once solutions. While many of the other quiz systems don't make explicit the size or complexity of the programming problems they provide, IRONCODE problems focuses on solutions that can be written in ten or less lines of code. Similar to other systems, problems are provided to students through a web interface with a description and a text area where they submit complete programs. When a solution is sent, the backend compiles the program. If the program does not compile, students have the opportunity to make modifications. Once the program compiles, a test harness is used to test the correctness of the program, and no additional input by the student is allowed. The study finds that with this kind of exercises students complete their first programming course with better understanding of the fundamentals of programming language semantics.

The infrastructure described by (Piccioni, et al., 2014) is quite interesting. They created a small private online course (SPAC) to teach programming concepts to students at ETH Zurich. The programming language of choice was Eiffel, and the quizzes were delivered through a web interface, in addition to lectures in the form of embedded videos. To create programming exercises a lecturer is required to upload both a template program code and a test suite. A web service displays the problems to the students providing a code editor and an output window which displays the results of the test suite run against the student submitted code. A quiz consists of several questions, and randomization is provided by shuffling the order of the questions on each student session. Interestingly enough, to motivate the students to complete the quizzes, they introduced gamification elements. Electronic badges are awarded to students for the completion of quizzes, and there were no penalties for multiple quiz attempts. Even if the completion of quizzes is not accounted for in the final grade, students were motivated to prove themselves and win all the badges.

2.2 Test Driven Development

As explained by Kent Beck, Test Driven Development (TDD) is a technique were tests are written continuously to express our assumptions about the behaviour the code we write should have (Beck, 2002). Big upfront designs are discouraged. Design decisions are made organically as new running code that passes tests provides feedback on every step of the development effort. Also developers have to write their own tests as they can't wait for another team or staff members to do it. The overall system architecture must be composed of highly cohesive, loosely coupled components.

The iterative process development is driven by the tests. A tests is written that represents the new functionality desired. Initially this tests fails because no new code has been added to the system yet. Afterwards a minimum viable solution is written, such that the required specification is met. As new edge cases and problems are detected in the design, new tests are written to account for them. Once the tests validate the behaviour desired, the code is refactored to eliminate duplication.

As Beck explains in his book, TDD is a method of managing fear. Not fear in a negative way, but the legitimate fear of modifying and complex system and introducing bugs in the process. For an existing system, with TDD it is possible to learn it concretely and document its behaviour. This process is repeated continuously until all the complex behaviour of the system is documented. Once a test is working, it will work forever, and they can be used to refactor complex portions of code into smaller units that are easier to manage.

2.3 Summary

As we can see the techniques and approaches in the development of quiz systems varies among the various implementations examined in this section. Some asked to write full programs while others just asked to predict the output of a program. The problems were delivered through a web interface and provided immediate feedback to the students. The rapid response, continuous feedback, and a careful designed set of questions engaged the students to try them and improve their programming skills. The results obtained in the understanding of programming concepts by students.

Test Driven Development as a technique to manage the complexity of code bases is based in the principle of using tests to continuously document, monitor and check the behaviour of a

system. The analysis and extension of legacy systems can be benefited of these techniques, and as we will later discuss in this following chapters, we used this technique extensively to be able to manage the complexity of the original code base of this project, refactor it and modify it in a set of independent loosely coupled components.

Chapter 3: Usability Improvements

In the section we will discuss the usability changes made in the system from the end users point of view: lecturers and students. We also discuss the rationale behind these changes, the design decisions that were taken and how they were implemented.

3.1 Question Template Editor

As a new user of the system in the role of a lecturer, one of the requirements that quickly draw our attention was the use of XML files to store the question templates. This process requires from the lecturer an intimate knowledge of the structure used for storage. This is completely unrelated with the activity of designing an interesting programming question. We considered this a distraction of this ultimate goal and that if possible, it should be minimized if not completely eliminated. Listing 4 illustrates the contents of a complete question template.

3.1.1 Problems with the current process

As it can be seen, even with the use of a modern text editor like Notepad++, the syntax highlighting feature of the editor is acting on the XML parts of the document, however this feature is lost for the Java code within the template, making it difficult to visualize. Furthermore, angle brackets have been replaced with XML entities. This process was not automatic, it had to been done manually by the lecturer, and the resulting code is even more difficult to understand. This process is completely unrelated with the programming question, and has to be done because otherwise the XML syntax of the document is invalid.

The insertion of placeholders in the code is an inevitable process as it is required to have portions of the code randomized by the system. However we also find difficult having to keep track mentally of where the placeholders have been inserted in the code. We also have to scroll up and down in the editor between the code section and the substitution section to check what a substitution meant, or if used at all.

There is also metadata about the question template in other XML tags, like the estimated time, the concepts tested, the difficulty and the instructions for the student. In order to not to break this structure, what a user would normally do is copy and paste the contests of this template and make modifications in a new file hoping that it will work correctly.

```
<question type="output">
  <estimated_time>100</estimated_time>
  <concepts>
    <concept>_AL00</concept>
  </concepts>
  <difficulty>1</difficulty>
  <instructions>
    What is printed on the screen?
  </instructions>
  <problem>

import java.util.ArrayList;   //HIDE

    class ExampleProgram {                        //HIDE
      public static void main(String[] args){       //HIDE

ArrayList&lt;Integer&gt; `s1` = new ArrayList&lt;Integer&gt;(10);

`s1`.add(`val0`);
`s1`.`s2`;
`s1`.`s3`;
`s1`.`s4`;

System.out.print(`s1`.size());

    }//HIDE
    }//HIDE
  </problem>

  <substitutions>

    <substitution val="s1">return
randset(array("list","myList","l","theList","thisList"));
    </substitution>

    <substitution val="s2">return randset(array("add(1)","add(-
1)","add(0)", "remove(0)", "remove(0)", "add(2)","add(3)"));
    </substitution>

    <substitution val="s3">return randset(array("add(1)","add(-
1)","add(0)", "add(2)","add(3)"));
    </substitution>

    <substitution val="s4">return randset(array("add(4)","add(5)","add(7)",
"remove(0)", "remove(0)", "add(11)","add(9)"));
    </substitution>

    <substitution val="val0">return rand(-5,5);
    </substitution>

  </substitutions>

</question>
```

Listing 4 A sample question template

Furthermore, once this edition process is completed, the author has to know where in the remote file system to upload this file using an FTP or SCP client. At minimum this could be solved with a file upload mechanism, but all the other problems would still be unsolved.

3.1.2 The new integrated editor

Considering all these shortcomings we decided to create an integrated editor as part of the web application.

For this to work we performed a thorough study of the current template parsing process. While we don't expect to replace the richness of editors like Notepad++, we wanted to create a form based interface with domain related fields and functionality to be able to create templates for this application. Figure 1 shows a capture of the editor at the moment of the writing of this report.

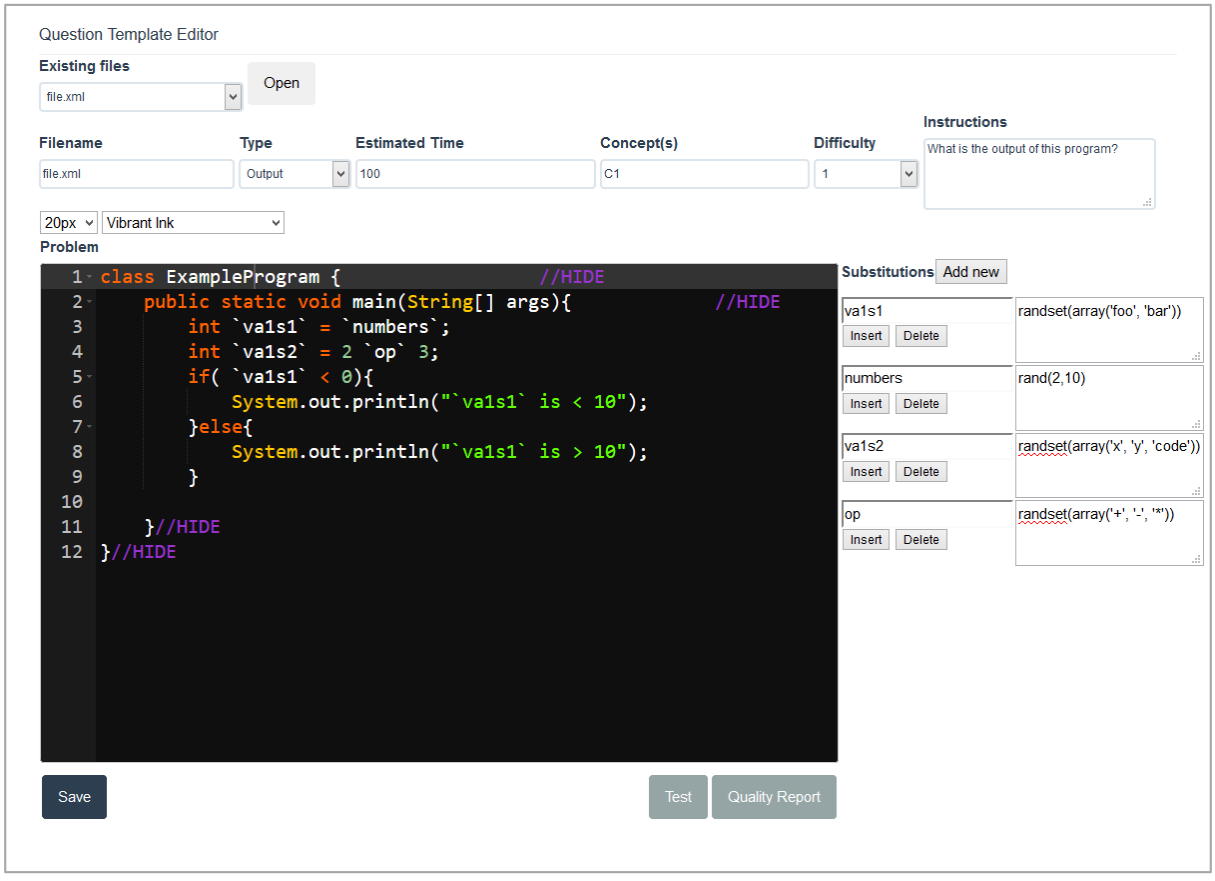


Figure 1: Question Template Editor

As we can see a large portion of the view has been devoted to an integrated code editor. The editor is based on the ACE editor project (ACE, 2015), a JavaScript based code editor that

can be embedded in any web browser. With it we have recovered the highlighting of the Java code, and the editor can support many other more programming languages if required in the future. The font size and the colour theme can be adjusted to the lecturer's preferences.

In the superior region there are fields for the setting of the question metadata information mentioned earlier, so the author does not have to deal with XML tags anymore.

Substitutions are listed on the right side of the window, and new ones can easily be created by pressing the Add New button. The lecturer has the option of providing a new identifier for the substitution or accept an auto generated one. To construct these dialogues and handle the event management we used the jQuery and jQuery UI libraries. If there was code selected in the editor, this selection is immediately replaced by the placeholder. New instances of the placeholder can be added anywhere in the code by pressing the corresponding Insert button. The editor takes care of entering the corresponding backtick characters. Furthermore, when a substitution field is selected on the right side, all the instances of the substitution are immediately highlighted in the editor. This provides a global visibility to the substitutions that would be very difficult to achieve in another editor.

For every substitution created, there is an expression field at its right, where any valid PHP expression can be written, as long as it provides a single output for the substitution process. This structure is the same for every substitution so we used Mustache (Lehnardt, 2015), a JavaScript template engine, to create a template of this combination of fields and HTML, and render it every time a substitution is created.

Figure 2: Sample of a rendered HTML created every time a new substitution is added to the document.

If a substitution field is not required anymore, it can be deleted with the specific Delete button. The editor will scan and highlight the source code for any instance of the substitution to be deleted, and will warn the user if it is in use at least once.

Another benefit of the editor is the save functionality. When the user presses Save, an Ajax request is initiated, sending the current contents of the editor to the server. If any required information is missing, a popup window is displayed, indicating the fields that are missing. Previously this type of errors would only be noticed at compilation time, and even then the precise nature of the error was not clearly specified. If everything is correct, a new XML file is created on the server transparently, with all the symbols and special characters handled by the server, so lecturers do not need to make any manual replacements of XML entities anymore. They do not even need to know that the templates are being saved as XML files behind the scenes, neither leave the application to use an FTP client.

Once the template has been saved, with the Test button a new window will open to test the generation of a question. Even though this module previously existed, it had to be reviewed because, as mentioned earlier, it did not provide sufficient information in case of an error. We inspected the code and found a way to capture the error message of the compiler, and now this information is displayed to the user.

```
int foo = 10;
double foo = 0.69;
if( foo < 0 && foo  > 0 ){
    System.out.println("foo is < 10");
}else{
    System.out.println("foo is > 10");
}
```

Figure 3: Generated code with an identifier used twice

Compiler errors:

```
C:\Vagrant\itec810\tmp\143307712021820.java:4: error: variable foo is already
defined in method main(String[])
    double foo = 0.69;
           ^
1 error
```

Figure 4: Compilation errors.

Figure 3 shows a generated code with problems, and Figure 4 the compilation error. Now the lecturer can quickly go back and make adjustments to the substitution logic to fix this error.

In this case, because of the randomization process and the set of identifiers used for each substitution, there was a possibility of generating invalid code, like assigning the same identifier to two different variable declarations. One or two tests may not produce an error, so to have more certainty we also added a Quality Report button that will try to generate many questions consecutively and keep track of how many were successful and how many were invalid. If a question has a low percentage of success rate, the author can keep doing adjustments before deciding to accept it in the system.

3.2 Quiz Sequence Editor

Another feature that was considered early on was the introduction of the concept of quiz sequences. This means that several quizzes can be defined and arranged in a sequence that implies a dependency, meaning that to start a quiz in the sequence, all the previous quizzes had to be finished first.

3.2.1 Student Module

Figure 5 illustrates this concept for a student accessing the system. There are in total eight quizzes. The order they appear has been decided by the lecturer early on.

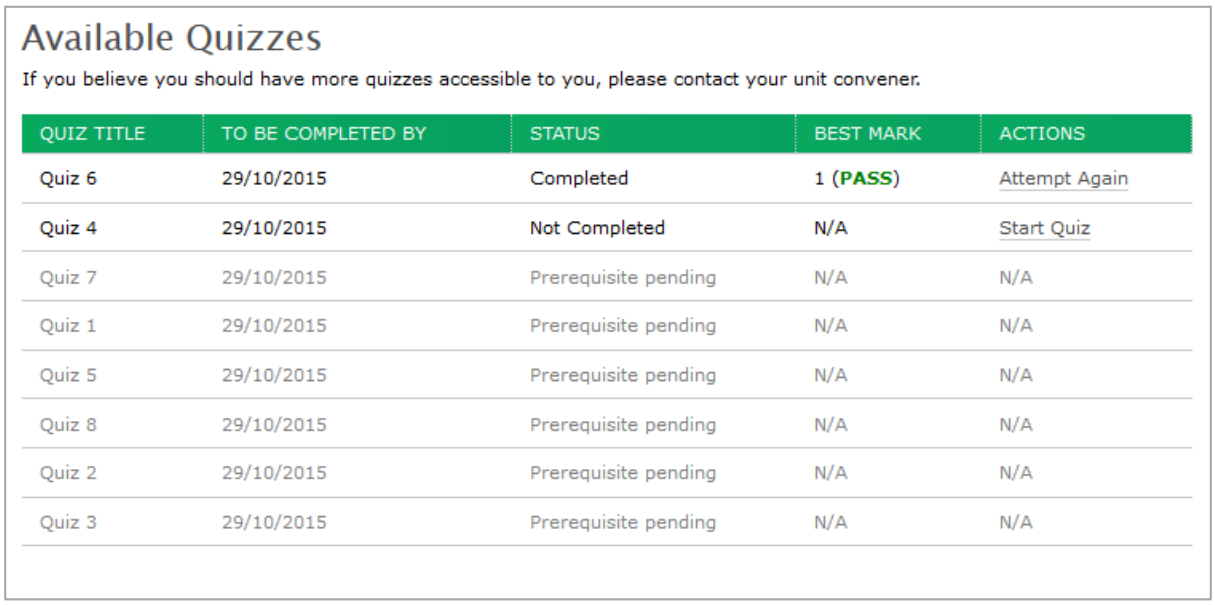


Figure 5: List of available quizzes.

In this particular case the student had to start with Quiz 6. Once completed, Quiz 4 was unlocked, representing the current state in the sequence. This way only one quiz is available at a time, and the others can't be accessed, until their respective prerequisite quiz is cleared.

3.2.2 Administrator Module

To be able to model this, some changes in the database schema were required. There are two modules to this achieve this functionality, an administrative module for the lecturer to be able to edit the sequence of quizzes, and a front end module for the students to be informed about their current progress in the sequence.

Since this was a change that required modification in many intertwined components first we created automatic tests to make sure that no existing functionality was broken during the addition of the required new functionality. For instance in one test we wipe out the database, import a fixture of question templates, create quizzes and associate them with concepts. All these steps are done with a simple command line.

With this process captured in our tests we decided that the best way to implement the sequence was to model it as a container. The sequence consists of quizzes that the administrator will be able to add, remove and reorder within the sequence. Figure 6 shows a capture of the proposed interface for the administrator module.

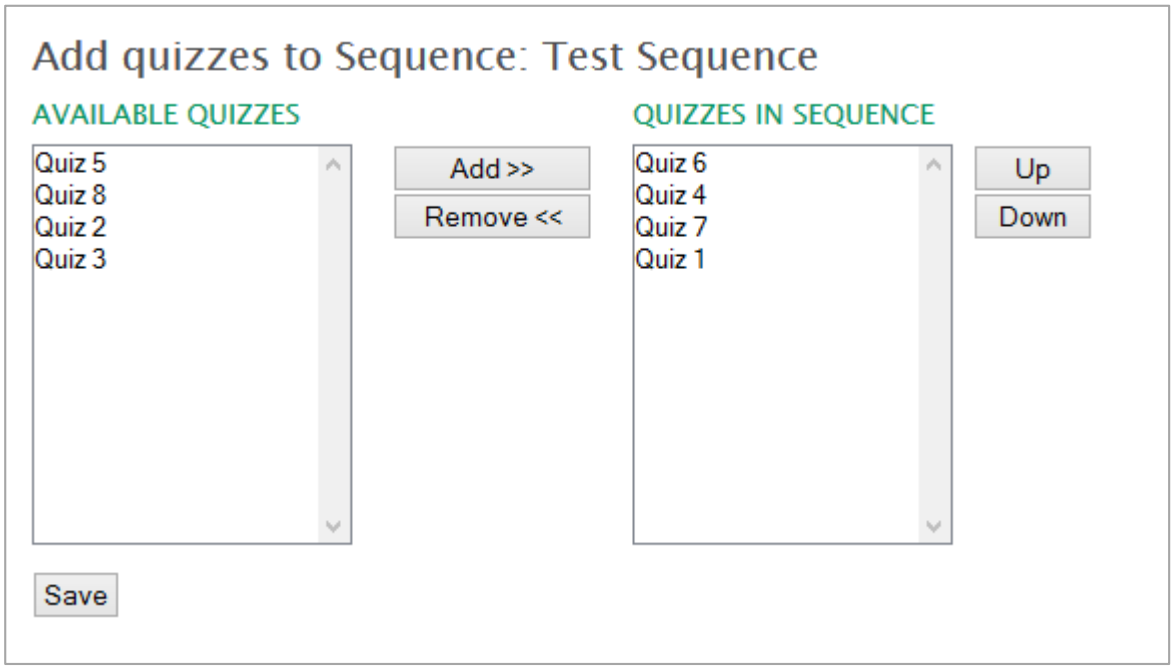


Figure 6: Administrator interface to define the sequence of quizzes.

Initially all the quizzes are listed as available in the list of the left. Then they are progressively added to the list of the right. This order on the latter list signifies the order they will appear when the student visualizes the page we saw on Figure 5.

Chapter 4: Robustness of the system

When we talk about usability of an application, we usually think of its end users, in this case lectures are students. However there is another set of users that have to interact with the system frequently: developers. We found considerably difficult to get started with the project because of all the steps required to get it started. Also some dependencies didn't allow us to work with the system unless we had an internet connection available or provided a directory service based on a large server product. Finally the remarkable amount of modules required many hours of inspection before we could implement our changes with confidence. We don't think this should this way so in this section we lists the changes we made to the system to make it friendlier for new developers.

4.1 Offline development

One of the first problems that had to be solved was user authorization. Information about the user is not stored in the quiz database.

4.1.1 LDAP Authorization

The system is designed to retrieve user information and login credentials from a LDAP based directory service. For local development we would have to supply our own directory service or use the real one at Macquarie University and request for test users to be created for us. Figure 7 illustrates this scenario. We believe that both alternatives incur in unnecessary installation or administrative time.

Figure 7: To be able to login into the application, a developer requires credentials from a directory service.

4.1.2 Removal of the dependency

Since the topic of user managers and credentials was an aspect that was not part of the project, we decided to design a workaround that would allow us to work locally by removing the dependency of the directory service. However we still needed the concept of students, administrators, and student groups.

It turns out that very little information was actually retrieved from the directory, so we located the parts of the code where direct calls to the directory service were performed, and replaced them with and indirection layer in charge to intercept these calls and call the actual directory for us. This way the system interacts with the interface and not directly to a particular directory service. Then it was a matter of supply our own implementation of the interface with hardcoded usernames, groups and credentials.

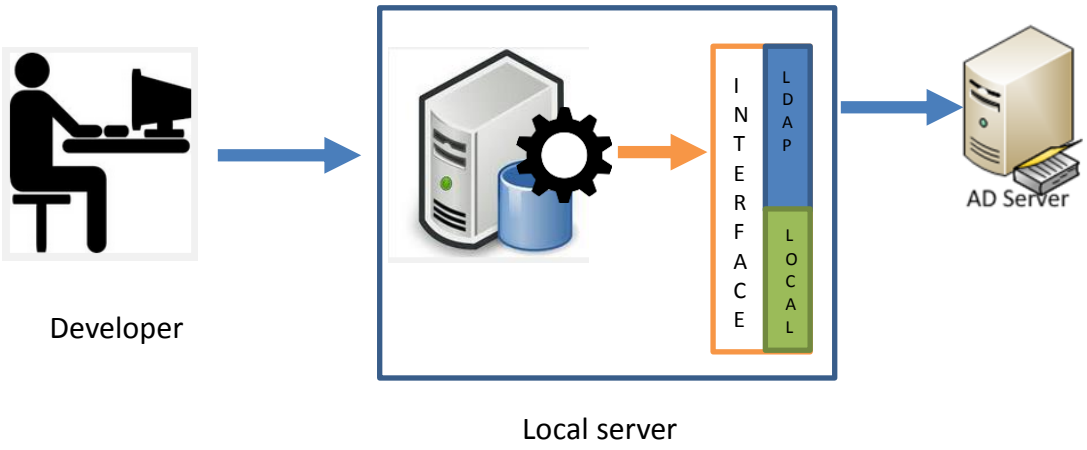


Figure 8: Introduction of an interface to disconnect the system from direct contact with the directory service. The green block represents our local implementation.

This way, depending of the environment we are, we provide a concrete implementation of the interface, one for production which connects with a real directory service, and one for development, that returns information from a fixed list. In our case we used a simple list with small arrays representing user and group information.

4.2 Reproduction of the Server Environment

4.2.1 Manual setup

When developers have to set up the environment for the first time, they are required to install and configure a long list of server software. Depending of the operating system on the development machine, this might be more or less complicated, even though the instructions in the original project are specific for the Ubuntu Linux platform. This made us think what would happen if the developer is using another platform, like Windows or OSX. Let alone that even if they decide to install the required software, it may conflict with already existing software or configurations.

Also as the project has had some time in development, the versions of Apache, PHP, MySQL and other software libraries used to develop it may not match what is available online at the moment, complicating matters further as developers are forced to search for the exact versions to avoid the introductions of errors for incompatibilities with more recent versions. We planned for new developers to have the ability to setup this server environment with the minimum amount of effort and time regardless of their development platform of choice.

We could solve the problems mentioned above with virtualization, but that alone doesn't solve the problem of installing all the required server software. It becomes an exercise of manual installation and configuration, and if anything goes wrong with the virtual machine, the install procedure has to be repeated again.

4.2.2 Automatic installation and configuration

What we need is a way to reproduce the target platform in a way that is isolated from our current operative system, and with the correct versions of the server software required. Also this procedure must be repeatable and automatic.

So for this task we used a software called Vagrant (Vagrant, 2015), which allows the automatic configuration of virtual machines. Installed in the host operating system, this software is capable of setting up a virtual machine with the desired guest operative system. The configuration of the virtual machine can be made through shell scripts. In our case, Ubuntu is a Linux distribution, so it can use setup script written in the Bash scripting language, which is a command line interpreter for Linux distributions. This script contains instructions to automatically install all the server software required for our system: Apache, PHP, MySQL. It creates the project database and test users. It installs utilities like phpMyAdmin and PHPUnit. Enables URL rewriting, and MySQL logging. Finally it installs and configures a Java interpreter. All these steps are expressed as commands in the bash scripting language. This configuration script is distributed as part of the final project.

Thanks to this, the steps have been reduced to download and install Vagrant, clone the project repository and run `vagrant up` on the command line.

This initializes the virtual machine and installs all the software required for us. At first it takes a while to download all the software, but we don't have to do the installation steps manually anymore, and once done further initialization processes are no different than those of a normal virtual machine. If something goes wrong with the virtual machine, it can be easily destroyed and created again from scratch. Furthermore, by having at hand a virtual machine that closely resembles the target platform, we are in better position to detect bugs much earlier in the development process, in highly contrast of the alternative of doing changes in an different operating system, and hoping that our changes don't produce bugs in the target platform.

4.3 Test Suite and Coverage

4.3.1 Legacy code

In this section we will describe the current architecture of the system, the code base, and the steps take to improve the work with it for future developers.

An initial analysis of the current code was required to be able to determine directions of development. The original project was developed using a Model View Controller (MVC) architecture using custom made classes and the ported to the Zend Framework (Zend Technologies, 2015). MVC frameworks are used to speed up development by providing guidelines and best practices for common requirement for web applications, like abstracting the complexities of the HTTP protocol, recommending a layout for the source files in the file system and providing useful libraries.

For a project that has been in development for quite some time we were surprised to find that there were not any kind of unit or functional tests covering the critical parts of the system. Code bases like this are difficult for new developers to get started with and modify because it requires extensive analysis of the system before major changes can be made, otherwise we run the risk of introducing bugs that we did not notice because we forgot to make some manual test that could expose it.

Consider the following scenario, when we wanted to add support to sequences of quizzes. First we had to make sure we understand the current process, and to keep things simple, we studied the following scenario: “When a lecturer creates a quiz with 2 questions, verify that the student can complete the quiz session”.

To be able to test this, the following actions have to be performed manually:

1. Reset the database to known state. This step ensures that this test is not influenced by the data left by any previous test.
2. Login as admin.
3. Import some XML templates.
4. Create a quiz.
5. Add two tested concepts to the quiz.
6. Logout
7. Login as student

8. Start the quiz. Verify the question is being displayed.
9. Send the correct answer. Verify the result view is displayed
10. Click Continue. Verify new question is displayed.
11. Send the correct answer. Verify the result view is displayed.
12. Finally verify the quiz results screen is displayed.

There are several problems with this process: Firstly, during repeated testing, the first step is probably not performed, if ever done at all, as it requires to go through some external database software to clear the involved tables and load into them some known data. Secondly, it requires a human using a web browser to perform all these steps manually. Finally, this is just one of many scenarios that could be conceived to verify that the changes we made did not break existing functionality. For this particular scenario, on every step there are many other actions and branches that the user could take, and testing them all becomes tedious and error prone.

4.3.2 Automated testing

The solution to these problems is to use automated testing. For this we used a library called PHPUnit (Bergmann, 2015), based on xUnit testing frameworks. With it, manual steps like those listed in the previous scenario can be expressed as code, stored in a file as part of the project and run as many times as required.

On complex software systems, a change in a module may have repercussions in another. So the objective of automatic tests is to run them frequently and be able to detect any introduced errors. By being automated, developers are more likely to run them often, unlike manual steps that may be neglected or avoided for being time consuming tasks.

Since we started with an existing system with little documentation, there was a high probability of introducing errors if we made changes in the code without care. So we took an approach of methodically writing test cases for critical parts of existing modules, observe the results, being these the creation or update of database records, and write down our observations as test assertions (Almonaies, et al., 2011).

With PHPUnit we could write tests at different level of details. For instance, a system level test is to simulate an HTTP request sent from a user to our system. The tests verifies that an expected condition is present in the HTTP response, which could be the existence and contents of an HTTP header, a status code, or the existence of specific content in the response

body. With this technique, we ensure that the expected behaviour is documented. The significance of this is that in place of requiring a human to constantly check the correct behaviour of the application, now we can run these tests with a command line, as many times as we want and during the entire development process. A more intimate test consists on ensuring that a certain class or function behaves correctly. In this case we are testing at language level, but similarly we write assertions that document the expected behaviour. All these tests can be run periodically and they provide a safety net we can rely on to alert us of broken functionality.

As the number of tests grow, we can still run them all with a simple command line. With manual testing it is easy to forget about tests we did weeks or month ago to test some part of the system. However, with automated testing, once written down as a test, it will not be forgotten, and it will always be run together with all other tests verifying our assumptions, providing immediate feedback on defects.

Automated testing was a definite factor that enabled the speed up of development and the confidence to make all the changes we completed. The complete set of unit tests have been delivered as part of the project. It is not uncommon for a test suite to grow as large as or even bigger than the code base under test. The writing of automated tests requires discipline, but it leads to more maintainable code, and acts as documentation of processes and functionality, therefore we expect the suite to further grow as new requirements and functionality are developed.

4.4 Initial improvements and bug fixing

The automated tests from the previous steps allowed us to repeat the execution of tests than in another situation would have been tedious and error prone to do manually. Having said that, we still needed a mechanism to be able to observe the paths of execution of the code. For this we set out to implement a logging system. We identified two cases: logging at the application level and logging at the database level. Independent of our application, we can enable logging at the MySQL server, but this only provides us certain information about what queries are being executed against the database, not where and when in the code the query was requested. What we needed is a system that shows both application level events and database queries.

4.4.1 Application level logging

At the PHP application level, we were able to introduce a logging class. With this class we insert calls to our log class on specific points of interest of the code. This way when the code is executed we can verify in which order the different functions and methods of the system are being called. The log calls contain information like the current class and method, the current value of parameters or local variables.

The logger produced by this logger was invaluable, because as external users it is very difficult to understand in which order and what arguments are being passed to a function during execution. Used as an exploration tool in combination with automated testing, we can set the application in some known state, then manually perform an action, and observe the execution in the log. We used this technique extensively to understand the file based compilation process, the generation of source code from templates, the substitution process, the files generated and the naming convention of these files.

```
[2015/05/28 12:53:45] Model_Shell_GenericQuestion::getCorrectOutput:
Attempting to generate correct answer for Question.
[2015/05/28 12:53:45] : Entering Java Compilation
[2015/05/28 12:53:45] : Temporary Folder Given: C:\Vagrant\itec810\tmp .
File Prefix: 143278162536511
[2015/05/28 12:53:45] : Source passed was:

    class ExampleProgram {                        //HIDE
      public static void main(String[] args){       //HIDE

        int x = 29;
        System.out.print(29);

      }//HIDE
    }//HIDE

[2015/05/28 12:53:45] C:\Vagrant\itec810\tmp/143278162536511.java
 [2015/05/28 12:53:45] toExec: "C:\Program
Files\Java\jdk1.7.0_67\bin\javac.exe"  -g:none
"C:\Vagrant\itec810\tmp\143278162536511.java" -d
"C:\Vagrant\itec810\tmp\143278162536511" 2>
"C:\Vagrant\itec810\tmp\143278162536511.error.txt"
[2015/05/28 12:53:48] execResult:
[2015/05/28 12:53:48] Model_Shell_Compiler::java_win:
C:\Vagrant\itec810\tmp\143278162536511.bat
```

Figure 9: Output of logger during compilation process

As Figure 9 illustrates, we were able to visualize the substitution process of placeholders, the generation of temporary source code files, and the command line innovations used to compile and run the Java code and capture the output. This was very difficult to visualize just by looking at source code as in some cases it used very long methods with nested `if` structures, and unfortunate naming conventions.

Once the process was understood and expressed as automatic tests, we were able to refactor it and modularize it. One of the major changes was the capture of compilation errors, which required changes in the rather complex concatenation of strings to build the command line arguments for the Java compiler.

4.4.2 Logging of SQL queries

Having two separated sets of logs, one for the application level and another for the database is not very useful, as we cannot immediately see the relationship between the lines of both logs. The database queries are after all synchronous. So we need a single log with precise ordering of all the logged events happening during execution: application events and database queries.

To achieve this we had to investigate how the framework performs database queries. It turned out that there was an abstraction database layer in charge of executing queries against the database. Using a similar technique as described in section 4.1.2, we determined that we could use a layer of indirection to intercept these calls at application level and introduce our own implementation that called our logger before sending the query to the database server.

This concept is illustrated in Figure 10. As we can see, now both application level events and database queries are logged in a single file, and the exact time and order they occur during application execution. This gave us the information about what kind of queries are executed every time a model class is instantiated, and which function or method is executed before or after the query, and the values of parameters used.

As a development tool, there is also a setting to enable or disable this logging, and to specify the use of the original database layer without any logging in production. Even if minimal, logging introduces some extra overhead to the application so it should not be enable during normal use.

```
[2015/05/28 12:53:45] QUERY: SELECT * FROM concepts where
concept_name='Concept_1'
[2015/05/28 12:53:45] QUERY: SELECT qb.question_id FROM question_base qb,
question_concepts qc WHERE qb.question_id=qc.question_basequestion_id AND
qc.conceptsconcept_name='Concept_1' AND qb.difficulty='1'
[2015/05/28 12:53:45] QUERY: SELECT * FROM question_base where
question_id='123'
[2015/05/28 12:53:45] QUERY: SELECT DISTINCT qb.question_id FROM
question_attempt qa, question_base qb, question_concepts qc WHERE
qc.conceptsconcept_name='Concept_1' AND qb.difficulty='1' AND
qa.question_basequestion_id=qb.question_id AND
qc.question_basequestion_id=qb.question_id AND
qa.quiz_attemptquiz_attempt_id='1021'
[2015/05/28 12:53:45] : vQuestionBase: 1
[2015/05/28 12:53:45] : Generating... from q1.xml
[2015/05/28 12:53:45] Model_Quiz_GeneratedQuestion::fromQuestionBase:
Seeing if there are any Pregenerated Questions for Question Identifier 123
[2015/05/28 12:53:45] QUERY: SELECT generated_id FROM generated_questions
WHERE question_basequestion_id='123' AND generated_id NOT IN(SELECT
generated_questionsgenerated_id AS generated_id FROM question_attempt)
[2015/05/28 12:53:45]
Model_Quiz_GeneratedQuestion::generateNewFromQuestionBase Using path:
C:\Vagrant\itec810\application/../tests/fixtures
```

Figure 10: Complete logger output with database queries and application level events.

With all this infrastructure in place, the exploration, development and debugging of the application became more straightforward. We used this tooling to develop all the modules mentioned in the other sections in this report. These measures were implemented at the start of the project, and while they took some time to set up, once in place they helped to boost productivity in the long term.

Chapter 5: Compiler Service Architecture

When an application performs slowly, it degrades the user experience. This is an attribute that affects all end users: lectures, students and developers. Especially for students, if they find the application slow, they will probably try to avoid to use it, which defeats the purpose of creating it in first place. After ensuring that the application works correctly, we analysed how to make it perform fast.

Manual use and user perception can be subjective so we decided we had to capture some baseline data before we could do any major change in the architecture. This data can be used to compare the results of our changes (Padilla & Hawkins, 2010).

5.1 Profiling

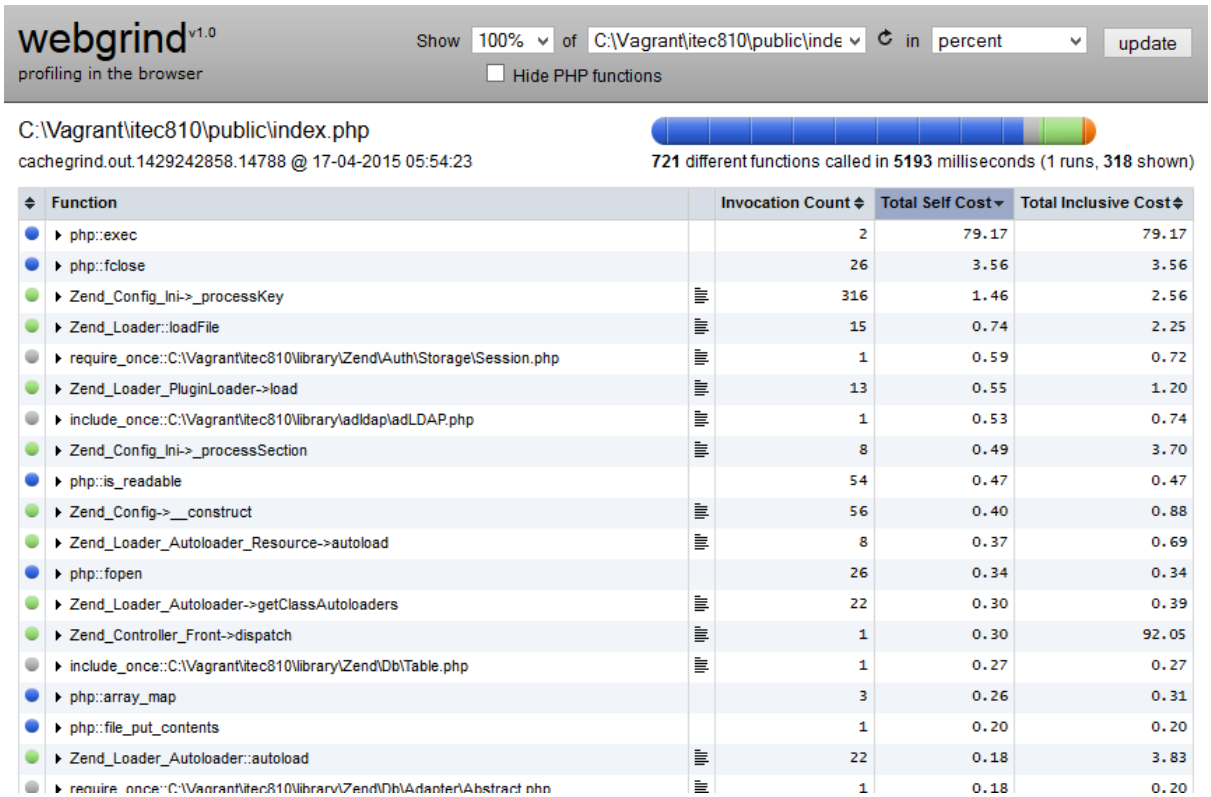
From regular usage, we noticed that one of the slowest parts of the system was the compilation process. To investigate the reasons behind this we used a specialized tool tailored for PHP systems called XDEBUG (Rethans, 2015), which provides the following information (Padilla & Hawkins, 2010):

- Memory consumption of the executed script
- The number of times a function is called
- The time a function takes to execute
- The stack trace of such functions

We enable the profiler for the page that presented the performance problem. We used the test generation page, which is used by lecturers to test the generation of a question by a given template. The results are shown in Figure 11. As we can see, about 80% of the time is spent is calling the PHP `exec` function. This function is used by PHP to execute system calls to perform actions that PHP can't do. In this case, as expected, PHP is unable to compile Java code, so it relies on the Java compiler installed in the platform where the application is running. So the steps the application performs to compile and run are the following:

1. Generate the programming code string.
2. Write the string to the file system as a text file with the .java extension.
3. Perform a system call `exec` to compile the generated source code.
4. Generate a runner which will run the compiled code and capture the output to disk.

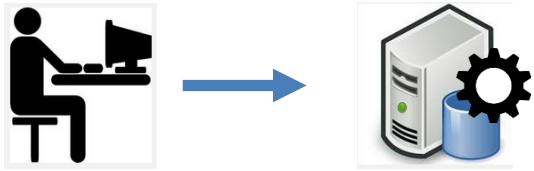
5. Perform a system call `exec` to the runner
6. Read the output from disk and save it as the correct answer.



Function	Invocation Count	Total Self Cost	Total Inclusive Cost
php::exec	2	79.17	79.17
php::fclose	26	3.56	3.56
Zend_Config_Ini->_processKey	316	1.46	2.56
Zend_Loader::loadFile	15	0.74	2.25
require_once::C:\Vagrant\itec810\library\Zend\Auth\Storage\Session.php	1	0.59	0.72
Zend_Loader_PluginLoader->load	13	0.55	1.20
include_once::C:\Vagrant\itec810\library\adldap\adLDAP.php	1	0.53	0.74
Zend_Config_Ini->_processSection	8	0.49	3.70
php::is_readable	54	0.47	0.47
Zend_Config->__construct	56	0.40	0.88
Zend_Loader_Autoloader_Resource->autoload	8	0.37	0.69
php::fopen	26	0.34	0.34
Zend_Loader_Autoloader->getClassAutoloaders	22	0.30	0.39
Zend_Controller_Front->dispatch	1	0.30	92.05
include_once::C:\Vagrant\itec810\library\Zend\Db\Table.php	1	0.27	0.27
php::array_map	3	0.26	0.31
php::file_put_contents	1	0.20	0.20
Zend_Loader_Autoloader::autoload	22	0.18	3.83
require_once::C:\Vagrant\itec810\library\Zend\Db\Adapter\Abstract.php	1	0.18	0.20

Figure 11: Profiling results of a page that requires compilation of Java code.

As we can see there are two system calls and several file system accesses. These types of operations are many orders of magnitude bigger than operations executed directly in memory within PHP. This is especially troublesome during the quiz evaluation of a student, since the Java compiler and interpreter are called for every question displayed to the student on every step of the quiz.



iq.science.mq.edu.au
(localhost)

- Apache
- PHP
- MySQL
- javac
- java
- File system based compilation and execution

Figure 12: Application architecture

On closer inspection of the architecture displayed on Figure 12 we can see that the web server is doing too many things. The web server performs well in the tasks it is specialized for: request handling, xml processing, view rendering, application logic, and data access. These tasks are handled very well by Apache, PHP and MySQL, the LAMP stack. For Java compilation however, the web server has to resort to expensive system calls.

So we investigated if it was possible to perform this Java compilation in memory without accessing the file system. After all, many dynamic languages like Python, JavaScript, Ruby and even PHP are able to execute code built dynamically from strings in memory.

5.2 Java Compiler API

Effectively, we found out that there is a Java Compiler API (Oracle, 2014) that allows us to perform the actions mentioned above. To get started we used a project by (Trung, 2015) which contained a prototype, test suite included, of the compiler API in action.

This, however, is just one part of the equation, as PHP still has no way to directly call Java and send to it strings for compilation. Nevertheless, PHP is able to communicate with web services.

5.3 The Compiler Service

Web services allow the communication of two different platforms using standardized protocols. We argued that if we were able to expose the functionality of the Java compiler

API as a web service, we should be able to invoke it from our web server regardless of how it was implemented.

We can see the proposed architecture in Figure 13. Essentially what we have done is separate two responsibilities into specialized nodes of computing. The web service, illustrated at the right, exposes a single operation, `compile`, with a single argument, `sourceCode` and returns a single response: the result of compiling and running the Java program.

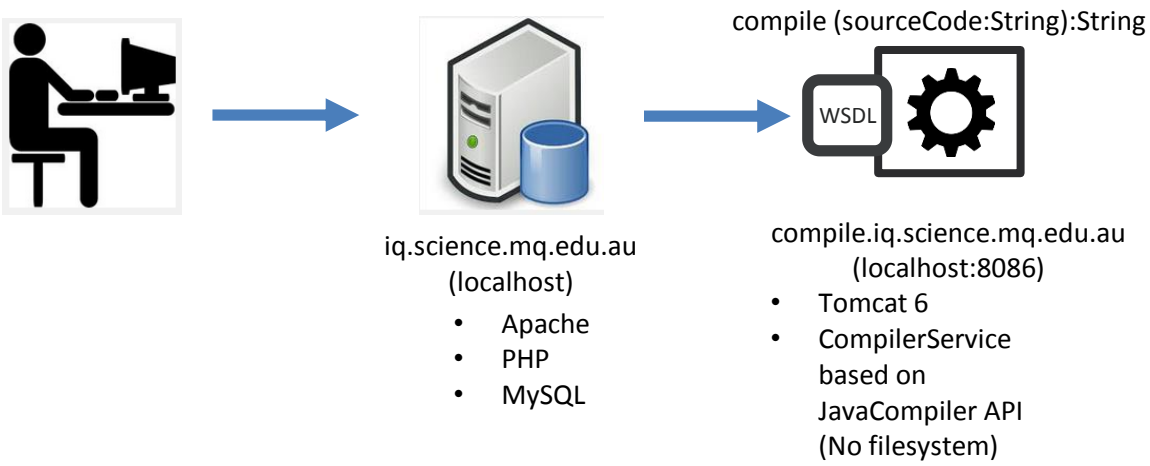


Figure 13: Proposed architecture using a web service

That's everything the Quiz System needs to perform the job of storing the correct answer of a generated question.

To create this web service we used the Apache Axis2 framework, and deployed it on a Tomcat web server. We can have all these systems locally running in our development machine, and with a configuration setting, the quiz system can be ordered to use again the system call approach if necessary. Having said that, we noticed that the system call approach used three different branches of logic, one for each operative system supported: Linux, OSX and Windows, since these platforms handle differently the process of generating files, compilation and execution. This web service eliminates the necessity of having to use and maintain all that code, since the same service call can be used regardless of the platform in use. This greatly simplifies the complexity of the PHP code base.

5.4 Benefits of the Compiler Service

The results look pretty promising. Repeating the experiment using the same page for profiling, and switching between system call and web service based compiling, we found a reduction in the time to complete a request from 2.99 seconds to 1.11 seconds, as shown in Figure 14. This means that the service is able to return responses 80% faster than the original system call approach.

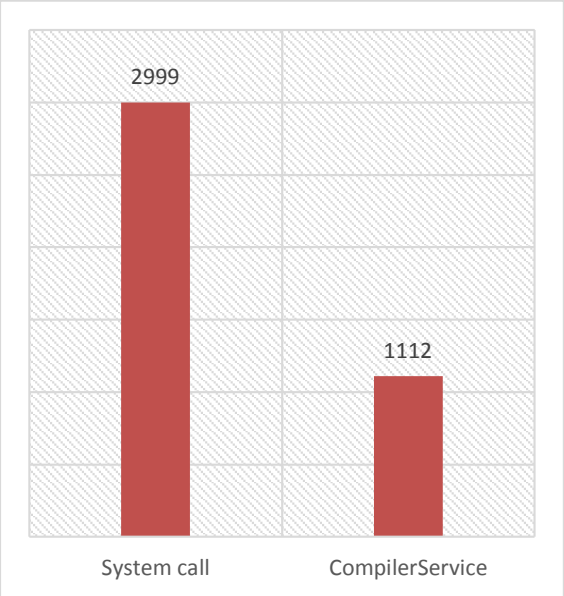


Figure 14: Time to complete an XDEBUG based request in seconds.

We used another tool to capture performance data, Apache Benchmark (The Apache Software Foundation, 2015), which is a command line utility that allows us to send HTTP requests to our application concurrently as if several users where using the system at the same time. From the administrator point of view, we are interested to know how long it takes for certain pages to load. In this kind of test we treat the system as an opaque entity, so we are not concerned about how it was implemented or what programming language it uses. We are only interested in the response time from an external agent point of view.

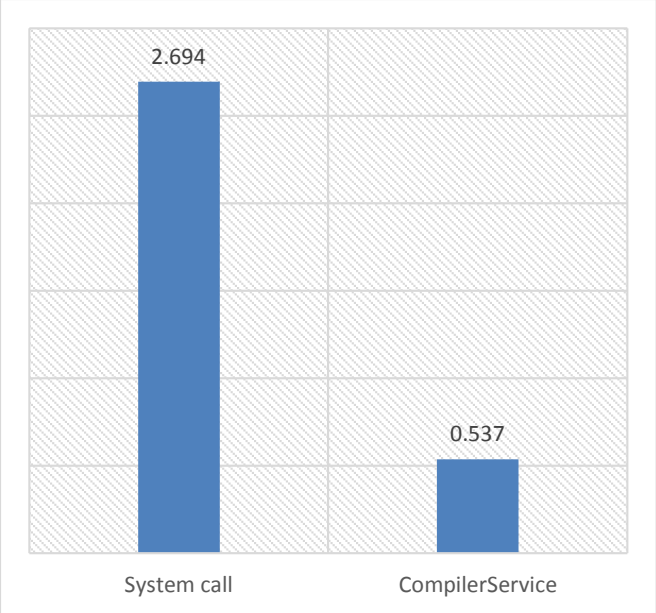


Figure 15: Using Apache Benchmark with 5 concurrent users.

We performed tests with 5 concurrent users, one using system calls and other using web services. The results were consistent with what was obtained on our previous tests. On average the time to complete the request went down from 2.694 seconds to 0.537 seconds, giving a performance boost of about 80%.

As we can see, the performance gains obtained with the compiler service have been quite remarkable. And we haven't lost any of the functionality provided by the system call based approach. When an error occurs, the web service returns an error status that the application can capture and display back to the user, exactly the same as it occurred with the file based compilation. This is an example of using the right tool for the job, and in the same way that an external directory service was used to handle user credentials, the application should not be very concerned with compilation details and hand off this responsibility to an external service.

Chapter 6: Future Work

In this section we discuss some changes that had to be left out to complete this iteration of the application. We did not want this to be just a very long session of bug fixing, and while many bugs had to be fixed, we decided to focus on new features that added more value to the system. New features open opportunities for new areas of improvements, so we list here some notes to be taken in future iterations the system.

6.1 Missing types of question

We focused on the type of question that has been in use thoroughly during the life of the application: output questions. However there are other types: Alternate Answers and Fill-in answers. These workflows have to be investigated and put under test to be able to properly fix any bugs that have prevented the use of this type of questions pervasively.

For the fill-in case, the Compiler Service provides an interesting solution to an old problem. In this type of question students are able to submit a complete program as an answer. Many things can go wrong in this scenario as the submission of an infinite loop, or the dangers of let arbitrary code execute in our servers. We saw in the original code many solutions that limit the execution time of a program, all of them platform dependent and incompatible. With the Compiler Service, all these security and time policies can be configured once within the server that hosts the service. This way the web server is not in charge of running complex commands and workarounds to be able to achieve isolation of the executed code, and delegate this responsibility to the web service who is in better position to do this, in particular in Java by using threads and scheduling APIs.

6.2 Question Template Editor

We also detected certain patterns in the type of substitutions that are commonly used in the creation of a question template. So, instead of having a text area to write arbitrary PHP code for each substitution, a dialog could be implemented where lecturers could specify that a substitution value comes from a random element of a list, and add or remove elements to this list visually. Another substitution could be a random number from a range, and provide the two numbers required to define the range.

6.3 Support for multiple programming languages

The original quiz system in theory supports any programming language, but only one at a time. We found settings reminiscent of its beginnings as a tool for C++ programming practice, and now it only supports Java. We don't think it should be that way, and with the externalization of the compiling logic, we think that the architecture shown in Figure 13 can be further extended to accommodate many programming languages, as show in Figure 16.

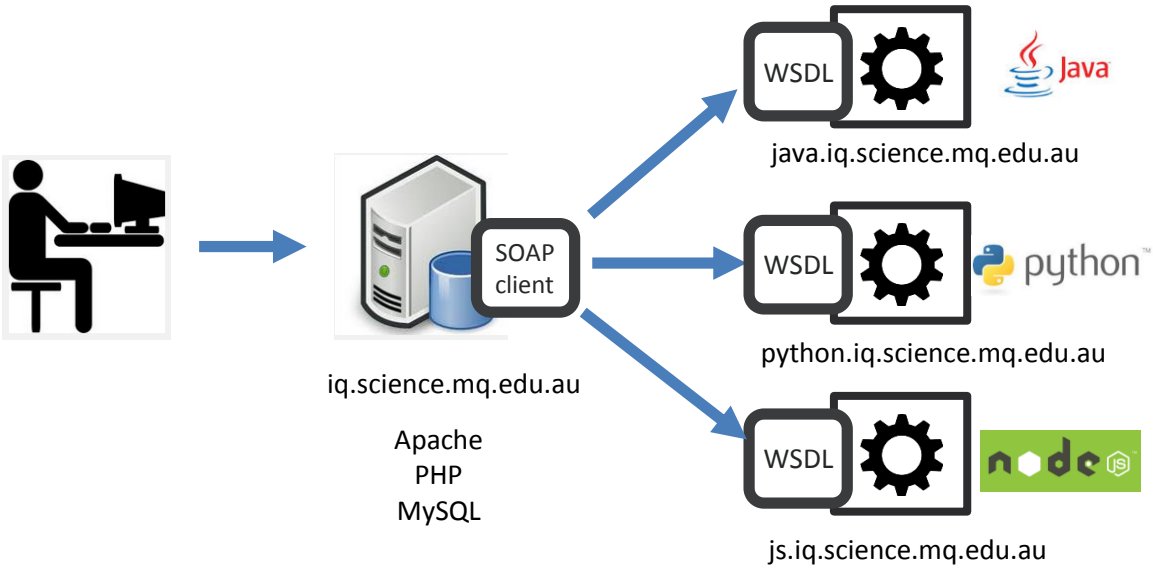


Figure 16: Architecture using multiple web services to support different programming languages.

In this model the quiz system becomes a quiz engine, completely unaware of what programming language is dealing with. We visualize the quiz engine working within the web server in charge of quiz administration, but delegating programming specific tasks to external services. As we can see in the previous figure, there are compiling services for Java, Python and JavaScript, all available at the same time. All these web services present the same operation, `compile`, with the same argument and return types.

This should not be very complicated in terms of creating new web services but for this to work the quiz system must be reengineered to work with any programming language in an abstract way, from question template creation to student evaluation, as many components and views of the system were designed to support a single language. The information about the programming language in use could be stored on the question template at design time, and then let the quiz engine dynamically adjust views and behaviours according to the particular

question it is processing, what compiling service to invoke, or what type of syntax highlighting enable in the editor at creation time.

Depending of the programming languages, there might be conflicts with the backtick character used as delimiter for the placeholders, so a setting at the question template level could be implemented to tell the engine what character to use as delimiter, and fall back to the backtick if nothing is set.

6.4 Security audit

Within the code we found many security vulnerabilities known in PHP applications, like directly accessing the contents of the $_POST and $_GET variables, or building SQL queries from strings and arguments sent by the client, instead of escaping them using prepared queries. This latter could lead to SQL injection attacks. We did the proper changes in the modules we worked with, but a more thoroughly audit is required for the entire code base.

6.5 Offline production of questions

There was some discussion about the production of a large number of questions in batches such that real time compilation is not required. We found traces of such a tool as a command line utility based on Zend command line controllers, but there was no indication that the questions generated by this tool were used when a student started a quiz. Real time compilation is always used during the production of questions for a student, and there is no evidence that a repository of questions is checked first, or where it is located. An investigation of the status of such tool should be performed to determine if it is a work in progress and the reasons its development was not completed.

6.6 Integration

The integration of the application with other learning management system like Moodle was discussed at some point. Macquarie University uses Moodle to power its iLearn platform. However we also were informed that this platform is managed by a third party provider and that the access to make the changes required to add the quiz system to such platform would be difficult. So for this project that option was not explored any further, but it could be considered again in the case the scenario changes. At the moment it is preferred to host the

application in our own private servers with the added liberty and flexibility to make the changes we consider best.

Chapter 7: Conclusions

In this report we have discussed the improvements in the usability of a student assessment system in use at Macquarie University.

We found several problems with the process to create new questions using XML files, so we built a prototype of an integrated question template editor that facilitates the creation of questions without the use of special symbols imposed by the XML syntax and with direct access to the file system of the remote server. The editor received positive feedback and it is expected to increase user productivity while reducing the learning curve for new users of the system.

There was no way to group several quizzes in a logical sequence with a clear prerequisite relationship. So we designed the quiz sequence system to provide the tools to create these relationships. Now the system can be used to design quiz sequences that last for the entirety of the session, instead of having a single quiz for diagnostic purposes. We expect students to feel challenged to complete all the quizzes in the sequences and feel motivated as they see their progress displayed on the application.

For current and future developers we removed the dependency to the remote authorization system. This allows us to work offline or outside of the university network. This speeds up development and testing process.

The Vagrant configuration and provisioning script allows the quick reproduction of the target platform development which can be used in the host operative system of preference of the developer.

The test suite was developed early on as an explorative tool to understand the behaviour of the system and to document this behaviour. This prevents the introduction of any bugs and avoids repetitive manual testing tasks. New test were added continually as new modules were developed and they can be run at any time easily from the command line. We believe that one of the main roadblock to evolve and innovate in a software product is that once the codebase reaches certain size or level of complexity, it becomes very difficult to change because of fear of breaking parts of the system that we didn't consider during our change. With the test suite in place this risk is minimized.

We proposed an architecture that greatly improves compilation time. We designed a compilation service, a web service specialized in the compilation and execution of Java code. We demonstrated with metrics that with this service based approach we obtained improvements in the response time of the application in average of about 80%. This will provide a more responsive application and better user experience for all its users.

Chapter 8: References

ACE, 2015. *Ace - The High Performance Code Editor for the Web.* [Online] Available at: http://ace.c9.io/ [Accessed 1 May 2015].

Almonaies, A. A., Alalfi, M. H., Cordy, J. R. & Dean, T. R., 2011. Towards a framework for migrating web applications to web services. *CASCON '11 Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research,* pp. 229-241.

Bailey, M. W., 2005. IRONCODE: Think-twice, Code-once Programming. *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education,* 37(1), pp. 181-185.

Beck, K., 2002. *Test Driven Development: By Example.* 1st ed. Boston: Pearson Education.

Bergmann, S., 2015. *PHPUnit - The PHP Testing Framework.* [Online] Available at: https://phpunit.de/[Accessed 15 03 2015].

Brusilovsky, P. & Sosnovsky, S., 2005. Engaging Students to Work with Self-assessment Questions: A Study of Two Approaches. *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education,* 37(3), pp. 251-255.

Cecchet, E., Udayabhanu, V., Wood, T. & Shenoy, P., 2011. BenchLab: An Open Testbed for Realistic Benchmarking of Web Applications. *Proceedings of the 2Nd USENIX Conference on Web Application Development,* pp. 4-4.

Conole, G. & Warburton, B., 2005. A review of computer-assisted assessment. *ALT-J - Research in Learning Technology,* 13(1), pp. 17-31.

Ellsworth, C. C., Fenwick, J. J. B. & Kurtz, B. L., 2004. The Quiver System. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education,* Volume 36, pp. 205-209.

Evans, B., 2010. *Randomised Quiz System for Assisted Learning,* Sydney: Macquarie University.

Evans, B., 2014. *nebev/randomised-prog-quiz.* [Online] Available at: https://github.com/nebev/randomised-prog-quiz [Accessed 28 February 2015].

Lehnardt, J., 2015. *{{ mustache }}.* [Online] Available at: https://mustache.github.io/ [Accessed 3 May 2015].

Molyneaux, I., 2014. *The Art Of Application Performance Testing.* 2nd Edition ed. Sebastopol: O'Reilly.

Oracle, 2014. *JavaCompiler (Java Platform SE 7).* [Online] Available at: http://docs.oracle.com/javase/7/docs/api/javax/tools/JavaCompiler.html [Accessed 28 April 2015].

Padilla, A. & Hawkins, T., 2010. *Pro PHP Application Performance: Tuning PHP Web Projects for Maximum Performance (Expert's Voice in Open Source).* 1st ed. New York: Apress.

Piccioni, M., Estler, C. & Meyer, B., 2014. SPOC-supported Introduction to Programming. *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pp. 3-8.

Rethans, D., 2015. *Xdebug - Debugger and Profiler Tool for PHP.* [Online] Available at: http://www.xdebug.org/ [Accessed 1 5 2015].

The Apache Software Foundation, 2015. *ab - Apache HTTP server benchmarking tool.* [Online] Available at: http://httpd.apache.org/docs/2.2/programs/ab.html[Accessed 1 5 2015].

Trung, N. K., 2015. *InMemoryJavaCompiler.* [Online] Available at: https://github.com/trung/InMemoryJavaCompiler[Accessed 1 May 2015].

Vagrant, 2015. *Vagrant.* [Online] Available at: https://www.vagrantup.com/ [Accessed 24 February 2015].

W3Schools, 1999. *XML Syntax.* [Online] Available at: http://www.w3schools.com/xml/xml_syntax.asp [Accessed 10 3 2015].

Zend Technologies, 2015. *Programmer's Reference Guide - Zend Framework.* [Online] Available at: http://framework.zend.com/manual/1.12/en/manual.html [Accessed 1 3 2015].

Appendix: Source Code

All the project files and source code are available in the following GitHub repository:

https://github.com/ivanres/itec810