|  |  |
| --- | --- |
| Polytechnic University of Bucharest | |
| Software Engineering  Quiz Generator  Software Requirements Specification | |
| Team:  David Suarez Caro, gr.1262E  Irina Talmaciu, gr.1262E  Salvador Canas Moreno, gr. 1262E  Teodora Mohai, gr.1262E |  |
| Coordinator:  As. Lucian Pestritu  As. Alexandra Todiruta | Date created:  Tuesday, November 13, 2012 |

# Delivery Report

(will be delivered along with the project)

Name Group Project implementation [%, reason] Signature

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Delivery date:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Table of Contents**

I. Politehnica University of Bucharest 1

II. Software Engineering 1

III. Delivery Report 2

IV. Table of Contents 3

V. Requirements Analysis 4

1. Introduction 4

1.1. Purpose 4

1.2. Scope 4

1.3. Definitions, Acronyms and Abbreviations 4

1.4. References 5

1.5. Structure 5

2. General description 6

2.1. Product Description 6

2.2. Product Functions 6

2.3. User description 7

2.4. Constraints 7

2.5. Assumptions and Dependencies 8

3. System Requirements 9

3.1. External Interface Requirements 9

3.2. Functional Requirements 9

3.3. Performance Requirements 11

3.4. Design Constraints 11

3.5. Software System Attributes 11

3.6. Other System Requirements 12

VI. Appendices 13

A1. Interview with the customer 13

A2. System diagram 13

A3. Use Cases Diagrams 13

A4. Class Diagrams 13

A5. Sequence Diagrams 13

A6. State Diagrams 13

A7. Document Evolution 13

A8. Report regarding team meetings 13

A9. Conclusions regarding the activity 13

**Requirements Analysis**

According to the IEEE STD-830-1993, *IEEE Recommended Practice for Software Requirements Specification*.

**1. Introduction**

**1.1. Purpose**

The purpose of this document is to describe the software requirements for a Quiz generator. It’s meant to help the team developing the application to have a good understanding of the functionality of the program and how it should behave.

**1.2. Scope**

The Quiz generator is a small application meant to ease teachers’ work during exam period. This application should pick random questions from a database and automatically generates a unique quiz for each student. The Quiz Generator shouldn’t be complex, a simple command line would be sufficient for achieving its purpose. The possible questions will be stored in a database and the generated quizzes will be saved as XML files.

**1.3. Definitions, Acronyms and Abbreviations**

**XML**- Extensible Markup Language

**SQL**-Structured Query Language

**UML**-Unified Modeling Language

**Definitions**

**Connection** **String** = string used to connect to a database, containing server name or IP, database name, database user name and password.

**Data** **Source** = data retrieved from the database based on an SQL query, used in displaying structured information.

**Query** = primary mechanism for retrieving information from a database and consist of questions presented to the database in a predefined format. Many database management systems use the SQL standard query format.

**Java** = object-oriented programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform.

**Class =** a construct that is used to create instances of itself – referred to as class instances, class objects, instance objects or simply objects.

**Method =** subroutine (or procedure) associated with a class. Methods define the behavior to be exhibited by instances of the associated class at program run time.

**Entity – Relationship model** (**ER model**) = an abstract way to describe a database. Diagrams created to design these entities and relationships are called entity–relationship diagrams or ER diagrams.

**Pattern** = software solutions that solve a recurring problem within a given context. The use of patterns has many advantages. Patterns encapsulate a design expert's time and expertise to solve a software problem.

**1.4. References**

Nicolae Goga (2011) Software Engineering Courses

Internet: Resource sites

**1.5. Structure**

**Chapter 1: Introduction** briefly presents the purpose of the document and also of the application and gives some references to the documentation and resources used for the development process.

**Chapter 2: General Description** of the product and its functionalities, user functionalities, constraints and dependencies on hardware or other applications.

**Chapter 3: System Requirements** gives a listing of requirements for the interface, functionality, performance and other possible requirements. Also it contains the constraints and advantages for the proposed product.

**2. General description**

**2.1. Product Description**

Quiz-Generator is a tool for making different quizzes for the final exams in order to prevent cheating. Quiz-Generator has to pick random question from a database and automatically generate unique quizzes.

Its main objective is to make easy things in order to simplify the hard work of making this manually. It doesn’t have a graphical interface because it is not necessary but it has simple commands to manage the program.

**2.2. Product Functions**

The application must provide the following functions or features:

**Extract questions from a pool**

The application allows user to **choose the allocation** of the database to extracts the data from the question pool to be able to make the quizzes. This function allows user to change the source file.

**Select the quiz format**

The product will allow users to **choose the format** for making the question exam. Of course, the application must allow the users to modify the number of question of each type should the quiz contain.

Also the user should be able to enter the number of different quizzes that should be generated.

**Select the output location**

The application will allow the user to select the output file where he will want to save the results.

**Generate the output file**

The program randomly picks questions from the pool for each quiz. For each multiple-choice question that is selected the generator must also randomize the choices every time.

To facilitate the grading process, each quiz and its answers are identified by a serial number printed at the top of the page.

**2.3. User description**

Quiz-generator is aimed at **users with any degree of computer science background**. The primary objective of the application is allowing beginners to easily create exams and quizzes.

Quiz-generator is best described as a **teaching tool**.

This makes our application best suited for the following user classes:

1. **Teachers**

Using Quiz-generator teachers will be able to better prepare exams and questions in various science fields. They will not waste so much time thinking the questions and the will avoid all kind of cheating.

**2.4. Constraints**

**2.4.1. Input constraints**

First of all the input format file must bean **XML file**. The user must be able to choose the location of the file and change this location.

**2.4.1. Output constraints**

The output shall be in the form of print-ready HTML files. Each quiz has two HTML files:

- The quiz itself

- The correct answers

The user must be able to choose the location of the file and change this location.

**2.4.2. Portability constraints**

The software must run on **every major operating system** and it must also provide a way to run the tool **inside a web browser**, via an applet or other kind of technology. This means that it should have as few implementation dependencies as possible. Some of the major operating systems today are:

* *Windows from Microsoft, Mac OS X from Apple, Linux, Android from Google and iOS from Apple.*

**2.4.3. Identify the quizzes**

To facilitate the grading process, each quiz and its answers are identified by a serial number printed at the top of the page.

**2.4.4. Quality constraints**

Being simple to use is not enough to acquire a large user base. The command must be as simple as possible.

**2.5. Assumptions and Dependencies**

Besides **running on current hardware and operating systems**, Quiz-generator must be able to run on older hardware too. **Systems up to 10 years old** must be able to run our application.

Quiz-generator is developed in the Java programming language, requiring **JRE version 1.7 or higher**. In essence any platform that can run a Java Virtual Machine should be able to run our application.

*Any other implementation dependencies must be avoided: third party libraries or modules.*

**3. System Requirements**

**3.1. External Interface Requirements**

Information is picked from a pool of files and unique quizzes are generated. The input files contain the questions and answers in an XML format. The output files are pairs of HTMLs with questions and answers.

The program is developed using a specialized IDE (like Eclipse) and written in Java. This requires the user to install a Java Runtime Environment.

Consequently, for the software interface are used specific xml and html packages like: java.xml.parsers and javax.swing.text.html. This, also, needs to be included in the existing JRE on the user's machine.

Having a command line interface, as hardware, a functional keyboard is required.

**3.2. Functional Requirements**

**3.2.1. Product Requirements**

In order to generate unique quizzes for each student, the user interface must allow the following operations:

**Extract questions from a pool**

The user enters the location of the question pool and the questions are extracted randomly generating the quizzes. The program must read from XML files and access any element from it.

The possibility to choose the directory from where to pick the questions allows the user to change the source file accordingly to the necessity of the person.

**Select the quiz format**

The user has the ability to select the quiz format. One quiz contains a number of questions chosen by the operator, this means that there are different subjects and from each subject the handler chooses a number of questions.

The user also introduces a number of students which represents the number different quizzes to generate.

**Select the output location**

After selecting number and format of quizzes, the location of the output files is passed to the application. The processing results in a number of HTML files that are used as questions and the counterpart answers.

**Generate the output file**

Processing the quizzes implies selecting random questions from each quiz and, in case of multiple choice responses; the possible answers are printed in random order in the output HTML file.

Facilitating the grading process, the quiz and its answers will be identified by a serial number printed at the top of the page.

**3.2.2. Use Case Description**

**3.2.2.1. Quizzes Generate**

|  |  |
| --- | --- |
| **Use Case Name:** | Quizzes generate |
| **Brief Description:** | This operation permits to:  - choose the xml files sources;  - select format;  - select quizzes;  - select output file. |
| **Priority:** | Essential |
| **Trigger:** | The user starts the application. |
| **Preconditions:** | The xml files must exist and be accessible. |
| **Basic Path:** | 1. Choose the xml file;  2. If the file is not in the correct format generate an error message and return to  point 1;  3. Choose the number of questions from that file;  4. If the number is greater than the total number of questions from that file, generate an error message and go to point 3;  5. If the input is correct, choose to continue or go to point 1;  6. Choose the number of quizzes;  7. Choose the output directory path, where the generated quizzes will be saved. |
| **Alternate Path:** | n/a |
| **Post condition:** | The files are saved in html format in the output directory. For each quiz there are two files: one with the questions and the other with answers. |
| **Exception Path:** | If the input xml files or the output directory suffer modifications during processing, the behavior is unknown. |

**3.3. Performance Requirements**

Being a simple application which is not using a graphical user interface, the user will run the program from his computer. The program can be run simultaneously from different computers. It only needs to have as input parameter a xml file and, at the end, the operator has to obtain as output file a html file.

The program must run every time it will be called and as estimated the application must complete in less than 10 seconds.

**3.4. Design Constraints**

Because there is no graphical user interface, the program must assure it is compliant with the rules described in chapter 2 – point 2.4.

**3.5. Software System Attributes**

Confidence

Maintenance

Tenacity

Portability

Fault tolerance

Easy of handling

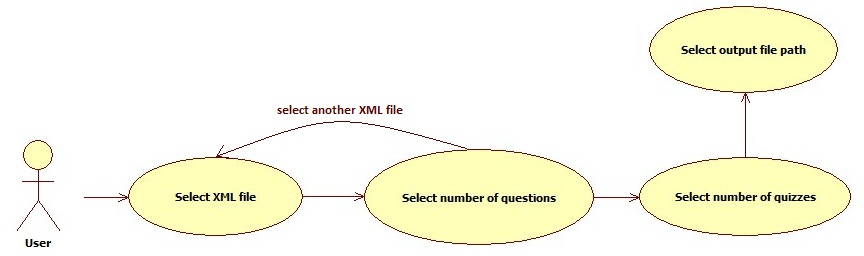
**3.6. Other System Requirements**

**Appendices**

**A1. Interview with the customer**

**A2. System diagram**

**A3. Use Cases Diagrams**

****

**A4. Class Diagrams**

**A5. Sequence Diagrams**

**A6. State Diagrams**

**A7. Document Evolution**

**A8. Report regarding team meetings**

**Date**:

**Location**:

**Participants**:

**Summary**:

**Activity**:

Title

Description (4 lines)

**Results**:

Element

Description (2-3 lines)

**A9. Conclusions regarding the activity**