Technical University of Moldova

Chair Computer Science

Report

On Designing Informational Systems

Course Thesis

Done by: st. gr. FAF-091 Melniciuc Constanin
Bibilici Victor

Checked by: Zarea Ivan

Chişinău – 2012

Contents

Task 1.

1. Creating the system vision

1.1 Bussiness context

Quizz tool Kit is a program which will generate quizzes based on a given subject. It will be used by teachers to test they're students, also will be used by students to prepare for their exams. It must be set on a Windows Platform System, this of course will be something new for teachers and students, and they will have to acknowledge with the interface and the possibilities of the program.

1.2 Bussiness oportunities

This program have a great future, it can be used by anybody, for creating tests on any subject you have. All you need is the material to study, some plain text files which contains necessary information. This is a great idea for making the work easier for teachers and educating a necessary work flow for teachers, this means that any paragraph will have to be specially tagged so that the program would understand that based on this it should a create a question. This is however may seem complicated but this is really important for modern world, because any report and material have to be well organized. This will organize as well the people and they will find easily what they need, taking into consideration that they organized really well everything else.

2. Stakeholders

Table 1 – Stakeholders and their role in the project.

Stakeholder	Role	Description
Teachers	Creates the tests	Introduces necessary material as in input.
Students	Study the material	
Developers (Melniciuc Constantin / Bibilici Victor)	Develop and sustain the program	- Monitoring the workflow of the program - Solving the problems appeared. - Creating the system
QR Code generator	Secure every single test	Generates unique QR Code
QR Code Database	Contains all QR codes	It's a database where all generated QR codes are kept.
Question generator	Generates Tests based on inputted material	Taking the input material and based on the specially tagged themes generates questions.
Input file	Material database	Contains the necessary text based on which tests are created.
User Interface	Making the work easier	Has an friendly user interface
PDF Generator	-	Generates .pdf files with questions.

3. Functional requirements of the System

Quiz tool Kit Software

- The operating system required for use is MS Windows 2000, MS Windows XP, or MS Vista.
- Require any browser to be installed.
- The quiz toolkit is properly loaded in browser.
- Supporting the pdf format for scan.
- Use of QR code for later recognition.

Quiz tool kit Hardware

- The application can use local or networked printers.
- Will not open from outside without using browser.
- Opens correctly from inside browser.

System interfaces

- The application provides a general introduction to the Quiz tool kit. It also provides links to the validation documents.
- Is properly formatted for printing.

Regulatory requirements

- Quiz tool kit have the ability to generate accurate and complete copies of records in both readable and electronic form.
- Users are able to select search queries.
- Users are able to customize the criteria used to view data from the search.
- Users are able to export data to pdf format.

4. Non-functional requirements

Process-oriented attributes

- Maintainability: changes to functionalities, repairs.
- Readability: of code, documents.
- Testability: ease of testing and error reporting.

- Understandability: of design, architecture, code.
- Integrability: ability to integrate components
- Complexity: degree of dependency and interaction between components

Reliabilty measures

- The precision of calculations shall be at least 1/106
- The system defect rate shall be less than 1 failure per 1000 hours of operation.
- No more than 1 per 1000000 transactions shall result in a failure requiring a system restart.

Availability measures

- The system shall meet or exceed 99.99% uptime.
- The system shall not be unavailable more than 1 hour per 1000 hours of operation.
- Less than 20 seconds shall be needed to restart the system after a failure 95% of the time.

Security measures

- Examples of requirements
- The application shall identify all of its client applications before allowing them to use its capabilities.
- The application shall ensure that the name of the employee in the official human resource and payroll databases exactly matches the name printed on the employee's social security card.
- At least 99% of intrusions shall be detected within 10 seconds.

Testability measures

- The delivered system shall include unit tests that ensure 100% branch coverage.
- Development must use regression tests allowing for full retesting in 12 hours.

5. Selecting architecture

I see that as a Service Oriented Architecture. Because every building block can play one or both roles:

a) *Service provider* – This application has a Web-Based interface, it contains all the needed information. All independent blocks in this application can be used as provider either as consumers.

b) *Service consumer* – The application's consumer is the teachers and students. Basing on the interface it will be used either to recheck the answers either to create quizzes for students. This is a powerful tool for any teacher, as well as for students.

Main reason why I've chosen this architecture type is because it reflects mostly the way the system works, and it will be the easiest way (in my opinion to be shown).

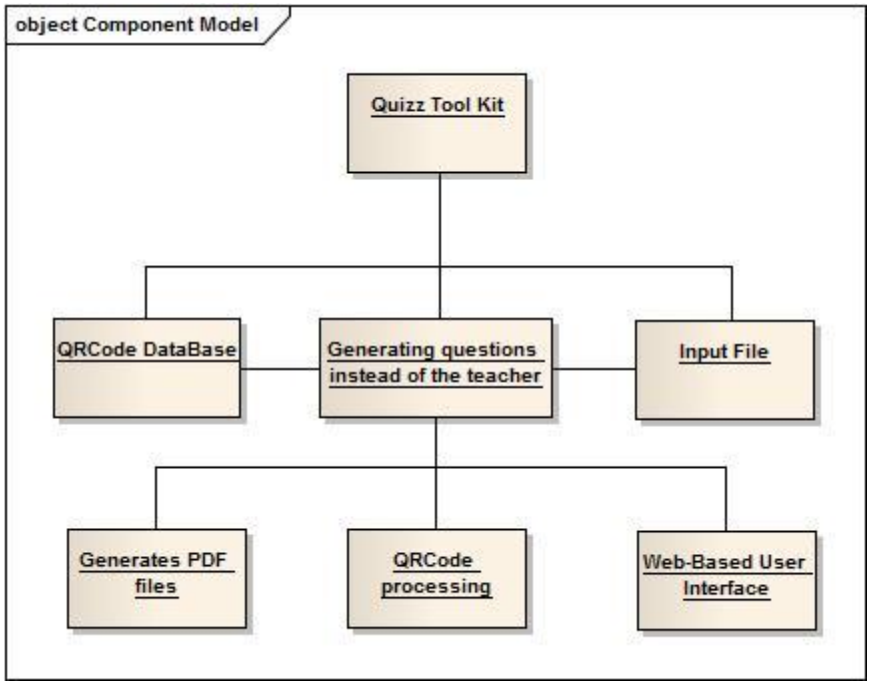

Figure 1 – Service Oriented Architecture represented graphicaly.

6. Architectural Sketch

To understand the workflow of the program, I will explain everything in the above shown diagram.

We need the generated .PDF file which contains the question for a quiz. As a consequence we need a “Question Generator”. On its order Question Generator need the “Input File” in which the information is well-organized.

Also we need the QR code processing part which generates and then recognize the QR code, used instead of a signature. This I a great security issue. QRCode is kept in the DataBase for later recognition if necessary.

The User Interface – is an independent module which specifies where to find the input file for Question Generator.

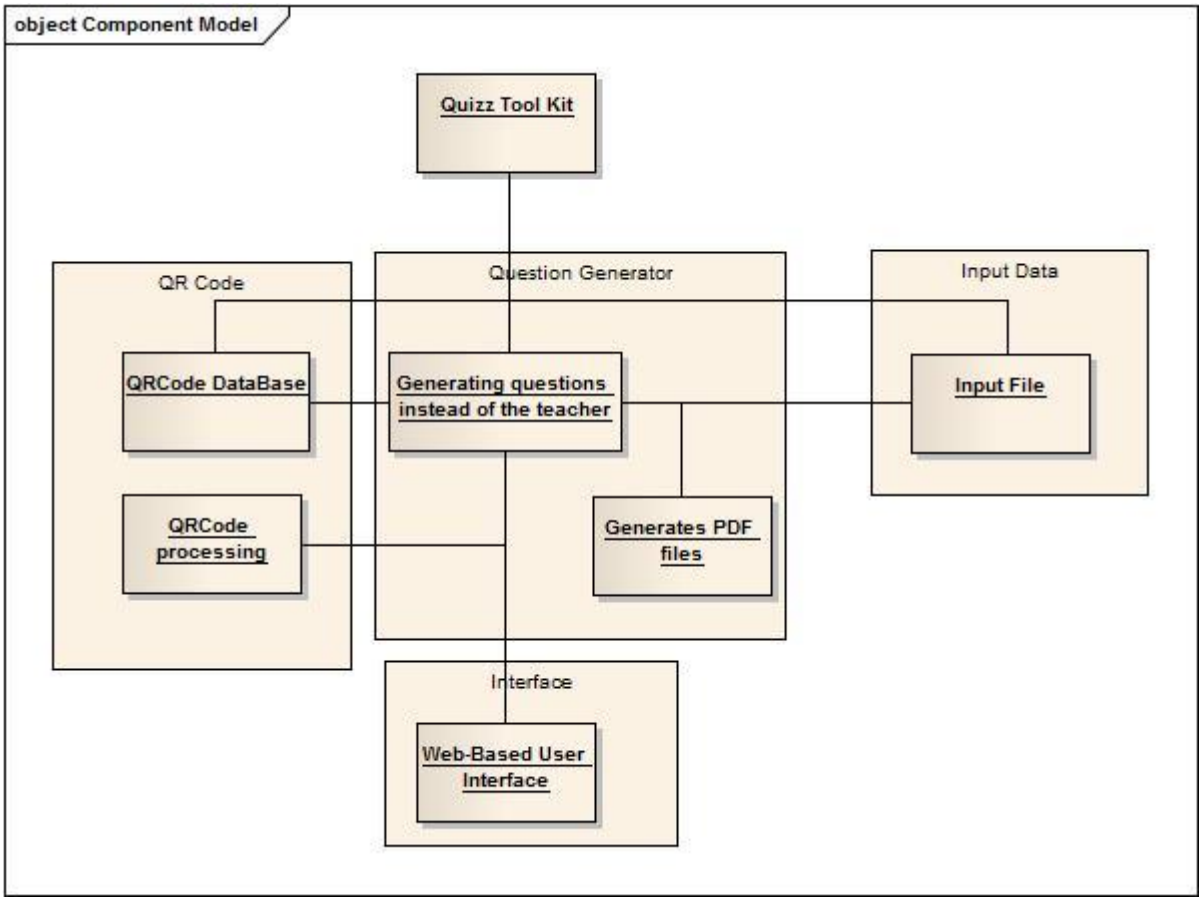

Figure 2 – Classified Version of Service Oriented Architecture represented in graphical manner.

Task 2

The main objective of this task is to try to describe three features of our application, each of them consisting of at least two scenarios of usage. Based one our scenarios, we designed an end-to-end test for that feature and write it using the available tools in our language. We tried to implement this stuff using TD development. In few words we will describe the technique, and the way we used it.

Well, the first step is to quickly add a test, basically just enough code to fail. Next you run your tests, often the complete test suite although for sake of speed you may decide to run only a subset, to ensure that the new test does in fact fail. You then update your functional code to make it pass the new tests. The fourth step is to run your tests again. If they fail you need to update your functional code and retest. Once the tests pass the next step is to start over (you may first need to refactor any duplication out of your design as needed, turning TFD into TDD).

It approximately looks like in this figure

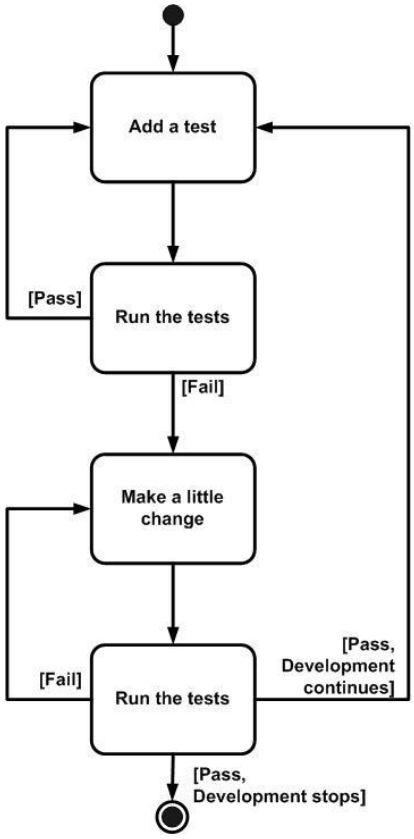

Fig.1 The Steps of test-first development

After this we used TDD (TFD + refactoring). TDD completely turns traditional development around. When you first go to implement a new feature, the first question that you ask is whether the existing design is the best design possible that enables you to implement that functionality. If so, you proceed via a TFD approach. If not, you refactor it locally to change the portion of the design affected by the new feature, enabling you to add that feature as easy as possible. As a result you will always be improving the quality of your design, thereby making it easier to work with in the future.

In this figure is represented a UML activity diagram showing how ATDD and developer TDD fit together. Ideally, you'll write a single acceptance test, then to implement the production code required to fulfill that test you'll take a developer TDD approach. This in turn requires you to iterate several times through the write a test, write production code, get it working cycle at the developer TDD level.

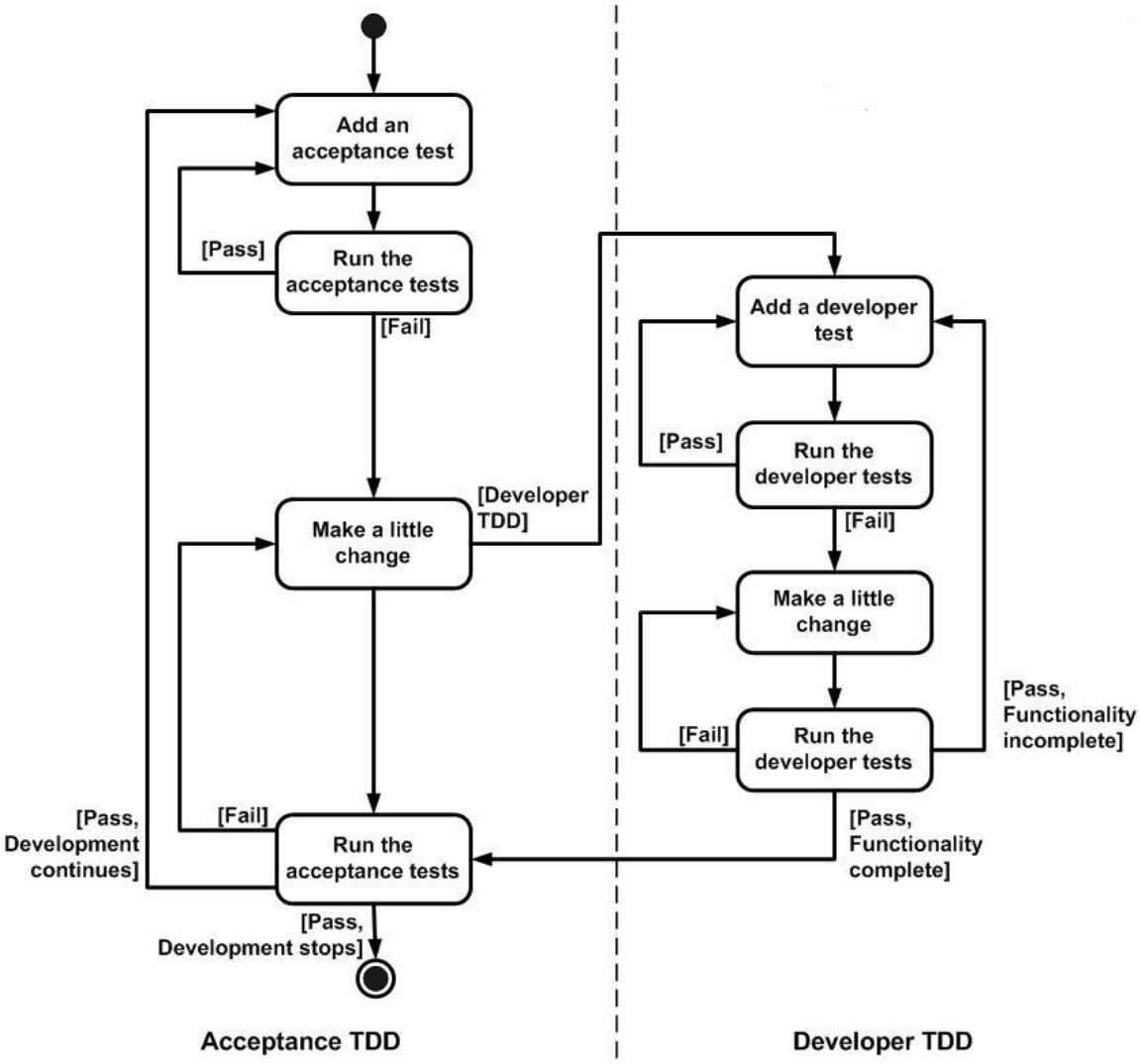

Fig.2 How acceptance TDD and developer TDD work together.

Note that Figure 2 assumes that you're doing both, although it is possible to do either one without the other.

With traditional testing a successful test finds one or more defects. It is the same with TDD; when a test fails you have made progress because you now know that you need to resolve the problem. More importantly, you have a clear measure of success when the test no longer fails. TDD increases your confidence that your system actually meets the requirements defined for it, that your system actually works and therefore you can proceed with confidence.

Well the scenario for our project itself, divided in two parts every of it consisting of three scenarios. In first one, we created the scenario for file creation, text editor and changing formats (txt to pdf). In our application this scenarios are very relevant.

Test descriptions (User story):

1) As an administrator, I want to connect to the server using sockets so that I could send to this server some data.
2) As an administrator, I want to verify if file created file exists
3) As an administrator, I want to save txt files as pdf files.

In this part we used JUnit which is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit that originated with SUnit. Figure 3 presents a UML state chart diagram for how people typically work with such tools.

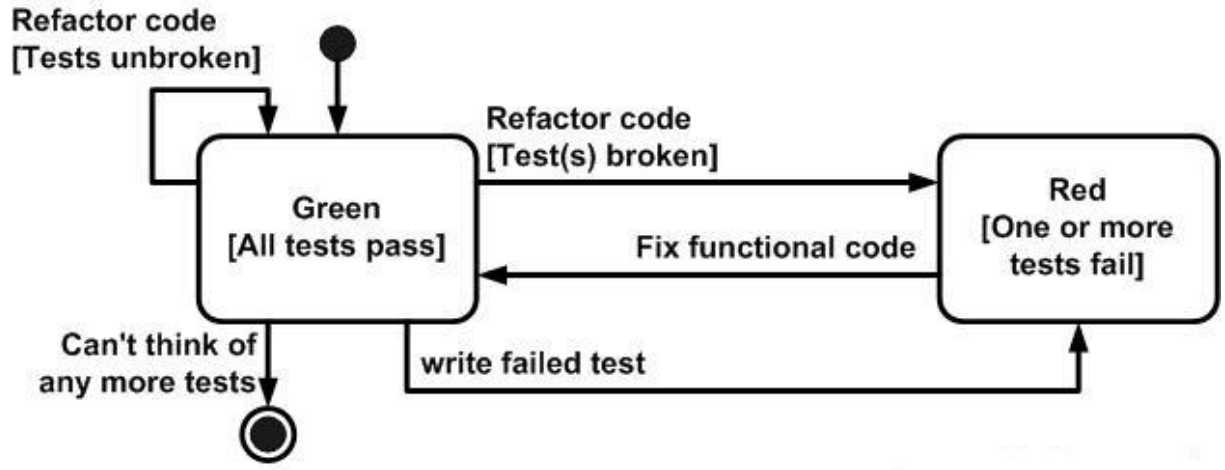

Fig.3Testing via the JUnit Framework.

A JUnit test fixture is a Java object. With older versions of JUnit, fixtures had to inherit from junit.framework.TestCase, but new tests using JUnit 4 should not do this. Test methods

must be annotated by the @Test annotation. If the situation requires it,[3] it is also possible to define a method to execute before (or after) each (or all) of the test methods with the @Before (or @After) and @BeforeClass (or @AfterClass) annotations.

In the second one we try to implement some scenarios using the meaning of QRcode.

Test descriptions (QRcode)

1) Encrypt data in a html, table.

Scenario: Author wants to encrypt some data into a QRCode image with high quality. With the source code the author can modify the source code and can introduce any data in it. Then Author have just to save the file, and open it in any known browser which support JavaScript.

2) Encrypt data in a html, canvas.

Scenario: Author wants to encrypt some data into a QRCode image with medium quality. With given source code author can modify the source code, can introduce any data in it, save the file, and open it in any browser which support JavaScript.

3) Decrypting the data encrypted in the QRCode.

Requirements:

1) A device which runs on Android OS.

2) QR Droid - A free application from

Scenario: Author wants to decrypt the data encrypted in the QR Code image. Author Runs QR Droid on his device, focus on the Printed QR Cod, scans it, and application recognize it as a valid QR Code.At the end shows author the encrypted data.

Generally speaking, there are two basics rules. First, you should write new business code only when an automated test has failed. Second, you should eliminate any duplication that you find. How these two simple rules generate complex individual and group behavior:

- You develop organically, with the running code providing feedback between decisions.
- You write your own tests because you can't wait 20 times per day for someone else to write them for you.
- Your development environment must provide rapid response to small changes (e.g you need a fast compiler and regression test suite).

- Your designs must consist of highly cohesive, loosely coupled components (e.g. your design is highly normalized) to make testing easier (this also makes evolution and maintenance of your system easier too).

For developers, the implication is that they need to learn how to write effective unit tests.

- Run fast (they have short setups, run times, and break downs).
- Run in isolation (you should be able to reorder them).
- Use data that makes them easy to read and to understand.
- Use real data (e.g. copies of production data) when they need to.
- Represent one step towards your overall goal.

Task 3

1. Code Inspection

Code inspection is a largly used technique that have to be used by every coder on this planet. You have to check your code for other people to understand your code, although when you watch someone's code you see fig1.; anyway you have to use some code inspection tools.

Types of code inspection:

- Over-the-shoulder – One developer looks over the author's shoulder as the latter walks through the code.
- Email pass-around – Source code management system emails code to reviewers automatically after checkin is made.
- Pair Programming – Two authors develop code together at the same workstation, such is common in Extreme Programming.
- Tool-assisted code review – Authors and reviewers use specialized tools designed for peer code review.

Being a team with Mr. Bibilici V. we Inspected the code in Pair Programming manner. In my opinion this is one of the most productive and fun technique, as when people have two different points of views, they are coming to an unexpected solution, that works better than

the way every developer initially proposed. Using this technique we achieved and optimized code, and I guess the best solution we could get.

While reviewing, the we also considered the strategic direction of the work, coming up with ideas for improvements and likely future problems to address. This frees the driver to focus all of his/her attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide. So this was awesome, because we were making each other's lief much easier. First I was very sceptic about this but later I understood that this is an awesome idea, and who invented it was really a mad guy.

2. Refactoring

Refactoring is usually motivated by noticing a code smell. For example the method at hand may be very long, or it may be a near duplicate of another nearby method. Once recognized, such problems can be addressed by *refactoring* the source code, or transforming it into a new form that behaves the same as before but that no longer "smells". For a long routine, one or more smaller subroutines can be extracted; or for duplicate routines, the duplication can be removed and replaced with one shared function. Failure to perform refactoring can result in accumulating technical debt.

There are two general categories of benefits to the activity of refactoring.

1. *Maintainability*. It is easier to fix bugs because the source code is easy to read and the intent of its author is easy to grasp. This might be achieved by reducing large monolithic routines into a set of individually concise, well-named, single-purpose methods. It might be achieved by moving a method to a more appropriate class, or by removing misleading comments.
2. *Extensibility*. It is easier to extend the capabilities of the application if it uses recognizable design patterns, and it provides some flexibility where none before may have existed.

The next step in refactoring would be to make the code more clear, for someone else who see this code to understand it, it should have more sugestive names. Maybe the code will more heavily in size but it will be more readable. Also adding comments to code help to improve the quality of code. The white spaces between lines also counts. Because it separates the code visualy in blocks of code, also separating the functionalities and compartiments of functions.

Refactoring have improved code quality, made it way more readable. The comments added to the code made it really clear and easy to read. I guess anybody could understand this code.

This refactoring achieved some goals of *Maintainability,* and some of *Extensibility.*

3. Architectural Patterns

An **architectural pattern** is a standard design in the field of software architecture. The concept of an architectural pattern has a broader scope than the concept of design pattern. The architectural patterns address various issues in software engineering, such as computer hardware performance limitations, high availability and minimization of a business risk. Some architectural patterns have been implemented within software frameworks.

3.1 ETL (Extract, Transform and Load)

Extract, Transform and Load (**ETL**) refers to a process in database usage and especially in data warehousing that involves:

- Extracting data from outside sources
- Transforming it to fit operational needs (which can include quality levels)
- Loading it into the end target (database, more specifically, operational data store, data mart or data warehouse)

As an example we will use a database with QRCode Images, in which will be included the names of all students from a certain group. So every student will have their own personalized QRImage, and this image will be printed on the test so that test will not be replaced. This feature will improve the security.

We need to create the application this way that QRCodes will be generated from an external database which will contain the names of all students. Every field in the table will correspond to a certain student.

The application will communicate with the precreated Database, and will generate automatically the images. The only need in teacher's activity will be to fullfill the database with necccessary names, and their group, but once, the database will be created it won't have the necessity to alter it.

So as we can see our application correspond to the principle of SOA.

- It extracts the data from a database.
- Application transform all the data into operational data and its needs.
- All the data is transferred to the end target application layer, and the data doesn't need to be refactored.

3.2 MFT (Managed File Transfer)

Managed file transfer (MFT) refers to software solutions that facilitate the secure transfer of data, in flight and at rest, from one computer to another through a network (e.g., the Internet). MFT solutions are often built to support the FTP network protocol. However, the term specifically describes solutions that remedy the disadvantages associated with FTP.

Typically, MFT offers a higher level of security and control than FTP. Features include reporting (e.g., notification of successful file transfers), non-repudiation, auditability, global visibility, automation of file transfer-related activities and processes, end-to-end security, and performance metrics/monitoring.

MFT applications are available as both licensed software packages and SaaS solutions. Some are specially designed for enterprise use while others are for sale to individual consumers. A few enterprise-focused SaaS MFT providers also manage the additions of new trading partners, which can free up a lot of IT resources.

In our application MFT will be implemented on printing complete tests.

Tests can be generated on any computer, and transferred to a local printer in the university. This way application is really flexible and universal.

Our application will have an extraordinary feature. Largely used last years in software parctice and science, cloud storage. The application will automatically upload all the test to a local UTM server where the tests could be verified by authorized persons. For example the Dean can access any time the files to see the tests and verify if the tests correspond to the standards. The files will be transferred over the server over File Transfer Protocol, or HTTPS.

If the professor has a computer home with acces to internet he can easily make the test at home in comfort with a cup of coffee or tea, without bothering about some problems that the files could be lost.

The goals of this architectural pattern are:

- The application support multiple file transfer protocols (FTP, HTTPS)
- Securely transfer files over public and private networks using encrypted file transfer protocols. HTTPS and FTP were designed to be secured initially so we don’t need any other further development.
- All the generated data will be stored on the utm server.
- HTTPS and FTP has built-in detection and handling of failed file transfers. At the end application shows the log in which there are all the results.

3.3 EAI(Enterprise Application Integration)

Enterprise application integration (EAI) is the use of software and computer systems architectural principles to integrate a set of enterprise computer applications.

At the level of our application we can integrate it by sharing it to every teacher in the university.

Regarding the cloud space, every teacher can have some limited space on the server, for example 5GB, this will be sufficient. All the data is kept there and there is no need to have radnom trash on your computer. The server can set that data older than 18month to be automatically deleted. And free space is automatically gained.

3.4 ESB (Enterprise Service Bus)

An enterprise service bus (ESB) is a software architecture model used for designing and implementing the interaction and communication between mutually interacting software applications in service-oriented architecture (SOA). As a software architecture model for distributed computing it is a specialty variant of the more general client server software architecture model and promotes agility and flexibility with regards to communication and interaction between applications. Its primary use is in enterprise application integration (EAI) of heterogeneous and complex landscapes.

In our SOA application there will be implemented some ESB duties will need to be specified. So, as interacting software applications will serve the main device itself with basic application and the print device or the server will be placed the information.
Some of the duties duties of an ESB will be :

- Monitor and control routing of message exchange between services
- Resolve contention between communicating service components
- Control deployment and versioning of services
- Cater for commodity services like event handling, data transformation and mapping, message and event queuing and sequencing, security or exception handling, protocol conversion and enforcing proper quality of communication service.

ESB will have to transform the message into a format that the application can interpret. A software “adapter” fulfills the task of effecting these transformations (analogously to a physical adapter).

Task 4

Steps to achieve the goal of the project

1. Creating the system vision.

   This step include in itself one of the most important steps of the application. Here are described Business Context; Business Oportunities; Problem Description.

   The product: Important documentation for future development.

   Accepting Criterias: Well formed Business Context; Business Oportunities; Problem Description.

   Nr. Of hours needed: 10-27.

   Start-End Date: 01.02-03.02.

2. Describing the Stakeholders.

   This step should include all the main stakeholders and it has to contain the description of responsibilities of each stakeholder. Usually it is represented in a table, so it is easier to read.

   The product: A important criteria for management of the team.

   Accepting Criterias: All the stake holders are included, and they have a well formed description, and all the responsabilities are well managed.

   Nr. Of hours needed: 9-18.

   Start-End Date: 01.02-02.02.

3. Documenting the functional requirements of the system.

   The functional requirements of the selected system define the functionality the system is to provide. Based on these requirements we will later derive the use case scenarios.

   The product: Documentation for achieving the main goal.

   Accepting Criterias: All Functional requirements are written and described. There should be no other necessity to add something.

   Nr. Of hours needed: 18 -27.

   Start-End Date: 01.02-03.02.

4. Non-functional requirements of the system.

The non-functional requirement of the system is a constraint imposed on it, which has an architectural value.

The product: Documentation for extended possibilities of the application.

Accepting Criterias: All Non-Functional requirements are written and described. There should be no other necessity to add something.

Nr. Of hours needed: 18-27.

Start-End Date: 03.02-06.02.

5. Architectural sketch

As we got some data from the previous steps, we have to design a sketch of the system's main components. Each of the components should be characterized by their responsibility.

The product: Visual preview of the system. Half explanation of the application

Accepting Criterias: The sketch have prety clear architecture, and it allready suppose what the main architecture will look like.

Nr. Of hours needed: 5-10.

Start-End Date: 06.02-06.02.

6. Functional Blocks

Some of the blocks have related responsibilities. They should be grouped in *functional building blocks*, which will define the modularity of the system. Therefore, the sketch will now change - it will also include the grouping of similar blocks.

The product: Documentation about every module of the application.

Accepting Criterias: The functional block are well defined and there is no need to regroup them later.

Nr. Of hours needed: 18-36.

Start-End Date: 06.02-08.02.

7. Selecting an architecture

We have to select an architecture, style of application. The are many styles but we have to select the one which corresponds to our application architecture.

The product: A complex preview of the system all in all.

Accepting Criterias: The selected architecture should satisfy all the necessities of the application.

Nr. Of hours needed: 1-5.

Start-End Date: 09.02-09.02.

8. Architectural Principles

We have to describe how our sketch comply the architectural principles.

The product: The advantages and disadvantages of selected archtitecture

Accepting Criterias: The architecture should correspond to the architectural principles.

Nr. Of hours needed: 25-30.

Start-End Date: 10.02-13.02.

9. Final architectural sketch

In this step we design a final architecture of the application. This is the main architecture, that's why all application work flow is based on one single diagram, well which is developed on lower level.

The product: The final preview of the system work flow.

Accepting Criterias: Well designed sketch of the architecture.

Nr. Of hours needed: 10-20.

Start-End Date: 13.02-16.02.

10. Write some test description

We have to describe all the features of our application and write down the scenarios most common scenarios for every feature.

The product: Test case for each feature.

Accepting Criterias: Test cases are describing really well the comportament of the modules/application.

Nr. Of hours needed: 20-30.

Start-End Date: 17.02-22.02.

11. Tests

In this part we predict a test case for every feature, and try to imagine how the application will behave in certain situations.

The product: Concrete Test cases for all the features, and this is great.

Accepting Criterias: Tests are representing the real behaviour of the application.

Nr. Of hours needed: 50-70.

Start-End Date: 23.02-2.03.

12. Actual Code

At this step we create our child, we create the application, we inspire life in our compiler.

The product: Actual application.

Accepting Criterias: A working version of the application.

Nr. Of hours needed: 440-480.

Start-End Date: 03.03-21.04.

13. Code Inspection

At this step we analyze the code we've previously written and correct all anomalies there were behind the process. We create version 0.2 of our child, we rewrite the fate of our creature.

The product: A lot of bugs.

Accepting Criterias: Well analyzed code, and optimized solutions.

Nr. Of hours needed: 20-27.

Start-End Date: 22.04-25.04.

14. Refactoring

Now we create version 0.3 of our application, and if there will be no improvements of the application this will be the final version application.

The product: Final version of the product.

Accepting Criterias: Everything should work as smooth as possible, code should be optimized as much as possible.

Nr. Of hours needed: 30-40.

Start-End Date: 26.04-01.05.

Conclusion

This domain is a huge one, but at the same time it offers you a general view of building a system and the steps you should to accomplish, objectives you should to achieve in order to gain a good informational system. It aims to familiarize us with information technology of large technical system analysis based on international standards. It also offers you the possibility to learn the principles of building informational and functional models, and what exactly tools you need in order to reach your goal. Methods of system analysis and modeling, allows solving the following problems:

• providing the required functionality of the system and adapt to varying operating conditions;

• design objects made in the system;

• implementing programs and interfaces (screen forms, reports) that will provide viewing results interpellations to databases;

• taking into consideration of environmental and technology of the project, as: network topology, configuration of technical resources, the architecture used.

In our course thesis we tried to implement all this principles. In our application named "Quiz tool kit" we tried to follow each step of main objectives of building an informational system. In the beginning, we describe the main reason of building such a system and provide some reasonable arguments. Also we provide the necessarily, functional and non-functional requirements which is very important "additional information" to the system and have included a lot of features for future implementing. We used JUnit test for eclipse to generate some of possible scenarios. And the most difficult part was code inspection and refactoring from the perspectives of Qrcode generator scenarios. As Server Oriented Architecture we choose the architectural patterns and applied it to our system describing the way they interact with application.

Bibliography

Books

1. Fowler, Martin (1999), *Improving the Design of Existing Code,* Object Technology International INC.
2. Beck, Kent (2000), *Test Driven Development: by example,* Pearson Education, Sep 1
3. Astels, David (2003) *Test Driven development: a practical guide*, Prentice Hall PTR,
4. Link, Johannes (2003) *Unit Testing in Java: How Tests Drive the Code,* Morgan Kaufmann, Jan 10
5. Massol, Vincent, Husted, Ted (2004) *Junit in Action,* Manning, Jun 28
6. Hunt, Andrew, Thomas, David (2003) *Pragmatic unit testing in Java with JUnit,* Pragmatic Bookshelf, Sep 8
7. Cédric Beust, Hani Suleiman (2008) *Next Generation Java Testing: TestNG and Advanced Concepts,* Addison-Wesley

Links

1. Code refactoring
   http://en.wikipedia.org/wiki/Code_refactoring
2. Test Driven Development
   http://www.agiledata.org/essays/tdd.html
3. Refactoring, principles
   ftp://po.istu.ru/public/docs/other/_Unsorted/new/Software@2520Development/Refactoring%20-%20Improving%20the%20Design%20of%20Existing%20Code%20(2002,%20Addison%20Wesley).pdf
4. Code inspection and refactoring
   http://codereview.stackexchange.com/questions/7300/refactoring-of-my-code
5. Reason of refactoring
   http://sourcemaking.com/refactoring/when-should-you-refactor
6. TDD
   http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd