[TURKUAZ]

MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

'Text Mining On Turkish Medical Radiology Reports'

TEST SPECIFICATIONS PLAN

By

Spring, 2007

Kerem Hadımlı – 1448752

Çiğdem Okuyucu – 1448976

Esra Zeynep Abacıoğlu – 1394568

Makbule Gülçin Özsoy – 1395383

İpek Tatlı – 1395557

Table of Contents

1. Introduction

1.1. Goals and Objectives

RadioRead is a text-mining project that aims on converting free-text radiology reports to a structured format and on providing doctors and staticians efficient ways to analyze the accumulated data. RadioRead consists of 5 modules, namely GUI, Database Layer, Query Engine, Mining Engine, and Morphological Analyzer. Database Layer handles all the SQL queries to the database, Query Engine handles processing of complex queries, Mining Engine is the biggest module which handles text-mining on reports, and Morphological Analyzer is the module which wraps Zemberek Library and also uses external sources like Zargan Roche medical dictionary to combine the results.

1.2. Scope of Document

RadioRead is a big project that should have been finished in a short time. The purpose of this document is to describe the testing process done for our project. While developing our project up to now, we did testing for each module, so we will explain about the testing process that took place since the beginning of the project.

1.3. Statement of Testing Plan Scope

Testing process of our project RadioRead includes unit testing, integration testing, validation testing, performance testing, alpha and beta testing.

Unit testing: Unit testing is implemented only for Mining Engine, Morphological Analyzer and Database Layer modules. GUI was found unnecessary for unit testing, and we run out of time for unit tests for Query Engine.

Integration testing: Our project combines and integrates multiple modules, especially submodules inside Mining Engine. Combining these modules together was very important that's why we chose to integrate integration testing within the testing plan.

Validation testing: We did validation testing in two categories: requirements validation and design validation.

Performance testing: RadioRead was running very slowly at the beginning of second semester, so we decided to include performance testing.

Alpha testing: This is the first "full" testing of RadioRead after the first partial tests are complete.

Beta testing: This is the testing done after the bugs and problems found in Alpha Testing period are identified and fixed the closest testing to final release.

1.4. Major Constraints

As SBAYazılım, we had several constraints and thougths on the testing process.

1.4.1. Time

We had only a few months on second semester, which is mostly occupied by the implementation of the project. We had fairly limited amount of time to dedicate for testing. We needed to run testing processes meanwhile we were implementing the project.

1.4.2. Usefulness of Obtained Test Results / Cost Ratio

The amount of usefulness of test results per the amount of time it takes and per the amount of human resource requirement is one of the most important factors limiting the testing of the project. Because giving too much importance to test some modules can be unnecessary when we consider the test results/cost ratio since working on testing process requires both time and labor.

1.4.3. Staff

Our team consists of only 5 people, and these people are also devoted to implementing the project. With these few people, it was relatively hard to assign people to both implementation and testing, so staff was a major problem in testing.

2. Testing Plan and Strategy

2.1. Testing Plan

In our testing process, our aim is to find as many bugs and design errors as possible, as early as possible in the development lifecycle. In order to do so, we needed to plan the testing beforehand, and make sure we have different test paths for different tasks. Our testing plan extends to both endpoints in a testing environment: from unit tests to scenario tests.

2.2. Testing Strategy

Our project consisted of two main testing groups: submodules of Mining Engine module, and the bigger modules we've talked about, GUI, DB Layer, Mining Engine, Morphological Analyzer, Query Engine.

Mining Engine was designed to be independent of all other modules (except Morphological Analyzer module). Tests for this module were conducted separately. We performed unit tests and tests on data flow on the mining process. We used a bottom-up technique, we started with testing single-words (Morphological Analyzer), testing Noun Phrase Parser submodule, then switching to LocationOrMeasurementAnalyzer (the submodule that identifies a tagged-as-"Location or Measurement" sentence part as either Location or Measurement), WhatQualityAnalyzer (the submodule that divides a tagged-as-"What or Quality" sentence part to one or multiple findings with quality and "what" information). After these initial tests are complete, we were able to test SentencePartsGrouper, SentenceMiner submodules. We needed to perform tests on report extraction submodules (for converting a report first into a structured form of sections, then converting these "String" sections to sentences in our internal data format) first, before we

could do tests on FindingsSectionMiner and ResultsSectionMiner and finally ReportMiner submodules.

For the other part of the project, namely the testing of bigger modules, we were more confortable. We employed a top-down testing strategy for those, first combining the modules together, and then testing them together. We were able to do so as the bigger modules were all bound to each other, and required the other modules to work perfectly in order to work.

3. Testing Procedure

3.1. Unit Testing

We used JUnit for unit testing of our modules. Specifically, as our submodules on Mining Engine module were strictly bound together in a linear way, we used a bottom-up technique for unit tests for these submodules.

3.2. Integration Testing

As our project consists of different modules and submodules, integration testing was necessary. During the integration testing, we first made sure all the submodules of Mining Engine module work together in a uniform way. After we're assured of Mining Engine, we tested Mining Engine – Database Layer, GUI – Mining Engine, Query Engine – GUI, GUI – Database Layer parts separately. Morphological Analyzer – Mining Engine connection was already tested with Mining Engine's own tests.

3.3. Validation Testing

Validation testing is the process that validates the conformance of the implementation versus design documents and customer requirements. We needed to perform validation testing

as our success was bound to correct implementation of our proposed algorithms in our design documents, and also the satisfaction of our customer.

3.3.1. Algorithm Validation

We needed to perform Whitebox testing for implemented algorithms and techniques within the project. In whitebox testing, the internals of a module are tested step-by-step, with the information of the exact data paths within the project. We tried to make sure every algorithm devised in design papers were implemented correctly, and were assured they work within some error ratio.

3.3.2. Customer Requirements Validation

We performed Blackbox testing for requirements validation. In blackbox testing, we only tested if the project gives desired outputs for specific inputs. We were mostly concerned with GUI and Mining Engine. We manually placed text mining reports, and watched the outcomes, and analyzed the results to see if they performed within an acceptable error ratio.

3.4. Performance Testing

RadioRead project's mining process was extremely slow at the start of second semester. We logged time statistics on a set of reports, and on critical submodules in Mining Engine module, then we improved the relevant submodules, optimizing the time they take to process their input.

3.5. Alpha – Beta Testing

We conducted 2 main testing periods, Alpha and Beta. Both are done by team members. In alpha testing period, every member chose a subset of radiology dataset given (usually 30-40 reports) and conducted tests on those data. In beta testing period, every

member chose some random reports, and tested using those. In both alpha and beta tested, every aspect of the project is tested, pretending the team is using the project in a live environment.

4. Scenarios

4.1. Analyze Report Scenario

After the GUI is shown, the user proceeds to Analyze Reports tab. From there, he loads a report from a subset of given dataset. He checks if the report is shown completely. He then enters patient information, clicks on Analyze Report. At this step, he checks if meaningful "AskDoctor" questions are asked regarding the unknown phrases in the report. After RadioRead informs the user that report is complete, the user checks the "All Reports" tab, then clicks on "Show Findings of Selected Report" button. He opens a copy of the report text in another window, then moving over each sentence, he tries to find the extracted findings from that sentence, and check if they're meaningful.

This scenario is applied for different reports.

4.2. List Reports Scenario

In this scenario, the user checks whether if the GUI – Database Layer integration is correct. The user first opens List Reports tab, from there he opens a report's text, then navigates to that report's findings. He should be able to see all the findings, and select each of them to see the detailed information on the right side of the window.

This scenario is for testing if the navigation codes and database interaction are implemented correctly.

4.3. Find Patient / Find Report Scenario

The user first goes to the List Reports tab to find a patient/report to search for. He notes down patient name, age, report date, and some extracted findings from the report. Then, the user opens the Find Patient tab, and enters some details of the patient to see if the system

would be able to find that specific patient. The user enters different subset of the patient (and his report) to see if he can find that patient with that information. The same steps are applied for Find Report tab.

This scenario basically tests the query engine.

4.4. Find Patient Count Scenario

In this scenario, the user first applies the same steps in Find Patient Scenario, but counts the number of patients returned. Then he enters the same details to Find Patient Count tab, and tries to see if the same number of patients is returned. This test is repeated for a few times, using different search details each time.

4.5. Patient Ratio Scenario

The user goes to the Find Patient Ratio tab in Statistical Queries tab. Here, she specifies two patient subgroups, each having the same details except for age field. She lets one of the patient group's age fields unbounded, where as the other within 18-35 years range. Then she tests whether a meaningful ratio will be popped up.

4.6. Measurement Graph Scenario

The user goes to the Measurement Graph tab in Statistical Queries tab. Here, he specifies a non-bounded selection on findings (so it matches all the findings ever extracted from reports), leaves the column count as 4, and clicks on calculate. He should see the number of findings that lie within 4 different measurement ranges.

5. Testing Tools and Environment

We have used the following tools and environment for testing process of RadioRead:

5.1. Eclipse

We've used eclipse for compilation and debugging in the first phases of the project development cycle.

5.2. NetBeans

We switched to NetBeans after we noticed the needs for preparing a graphical user interface and debugging it. Although most team members switched to NetBeans, we sometimes still used Eclipse, especially when working on department computers.

5.3. JUnit

We used JUnit Framework for unit tests of our independent modules.

6. Testing Resources and Staffing

The task distribution for testing process to the team members can be shown as follows:

Çiğdem Okuyucu: Text-Mining Engine, Morphologic Analyzer, GUI Engine,Data Engine

Esra Zeynep Abacıoğlu: Text-Mining Engine, Morphologic Analyzer, GUI Engine, Query Engine

İpek Tatlı: Text-Mining Engine, GUI Engine, Data Engine, Query Engine

Kerem Hadımlı: Morphologic Analyzer, GUI Engine, Data Engine, Query Engine

Makbule Gülçin Özsoy: Text-Mining Engine, Morphologic Analyzer, Data Engine, Query Engine

7. Testing Schedule

Our testing schedule is below:

Test Specification Planning	*06.05.2008*	*10.05.2008*
Unit Tests and Integration Tests	*30.03.2008*	*15.05.2008*
Validation Tests	*15.05.2008*	*20.05.2008*
Performance and Security Tests	*20.05.2008*	*25.05.2008*
Alpha Testing	*27.05.2008*	*31.05.2008*
Beta Testing	*01.06.2008*	*05.06.2008*
Bug Tracing, Detection and Correction	*01.05.2008*	*12.06.2008*