**Project Requirements Specification**

**Course:** CS 474 Object-Oriented Design

**Project Name:** RADIS

**Team Members**

Andrew McClellan

Austin Traverse

Kelsey Hilton

**Professor:** Rob Byrd, iSchool, Abilene Christian University

**Last updated:** 23 April 2012

**Document Revision Number:** 6

|  |  |
| --- | --- |
| **Table of Contents & Status Report** | |
| **Project Component** | **% Done** |
| 1. Introduction |  |
| **Project Plan** | |
| 1. Project Vision | **100%** |
| 1. Stakeholders | **100%** |
| 1. System Actors | **100%** |
| 1. Scope of Work |  |
| **Requirements** | |
| 1. Capabilities: Functional Requirements | **100%** |
| 1. Features: Non-Functional Requirements | **50%** |
| 1. Constraints |  |
| 1. Relevant Facts and Assumptions |  |
| **API Design** | |
| 1. Design Pattern: Facade | **100%** |
| 1. Underlying APIs | **90%** |
| 1. Class Diagrams |  |
| 1. Header Files |  |
| **Use Cases** | |
| 1. System Actors |  |
| 1. Scenarios |  |
| 1. UML Diagrams |  |
| 1. Relationships |  |
| **Appendices** | |
| * 1. Glossary of Terms and Acronyms |  |
| * 1. Naming Conventions |  |
| * 1. Noun Extraction |  |
| * 1. Transitional Matrix: BPM and Use Cases |  |
| * 1. Transitional Matrix: CRUD |  |
| * 1. Data Model |  |

1. **Introduction**

In this document we describe the requirements for a Radiological Application for diagnostic Imaging and Storage, which we will here by refer to by RADIS. First we describe the system concept and define its scope. Then we provide detailed requirements derived from user stories, constraints, and client specifications. Finally we provide a data model and other information management specifications. Refer to Appendix A for a glossary of terms and acronyms used in this document.

**PROJECT PLAN**

1. **Project Vision**

For the purposes of our project we are assuming that Hendrick Medical System, the client, has a need for a Radiological Application for Diagnostic Imaging and Storage. The application is intended for use by radiology specialists as a diagnostic assistant. RADIS is a type of electronic health record (EHR) system. It will maintain a database of patient history but also have image processing capabilities to aid radiologists and physicians in making a diagnosis and treating patients. RADIS will analyze images and use doctor confirmed patient history to provide a possible diagnosis. The system will also list alternate diagnoses which will be ranked on probability of correctness.

RADIS will use past history, and standard data compiled over time from a variety of patients to compare to patient data and confirm a diagnosis or come up with an alternate diagnosis that may have otherwise been missed. An accurate diagnosis means that the patient won’t need to return when symptoms persist as a result of an inaccurate diagnosis. There will be fewer frustrated patients who are more likely to return and/or recommend Hendrick Medical System.

A noun extraction of the above paragraphs was used for inventing key objects. It can be found in APPENDIX B.

1. **Stakeholders**

Stakeholders include:

Rob Byrd, our professor, who defined the goal, guides the design, and assigns a grade.

Radiologists, the primary users of the application.

Hendrick Medical Center, the client for whom the application is being developed.

1. **System Actors**

Radiologists: the primary user of RADIS.

Patients: the secondary user of the system.

Actors represent anything that interacts with the system; they can be human or external systems. This section lists of actors and brief descriptions. Refer to Appendix B for actor cards with full details.

1. **Scope of Work**

The goal of this project is to develop a diagnostic assistant for radiologists at Hendrick Medical Center. Specific objectives have been identified with respect to the project.

1. Develop an API that incorporates image processing into an automated diagnostic help tool. The API should be open-source and reusable.
2. Identify specific type of images and their normalcy using edge-detection and histogram features.
3. Identify different diseases from a combination of patient-reported symptoms and processed image results.
4. Create a user interface to demonstrate functionality of the underlying API should follow the following business process model(BPM):

* Starts with a patient needing to be diagnosed.
* The patient explains symptoms.
* Radiologist enters symptoms into patient history.
* Radiologist determines if a scan is necessary.
* Assistant takes images of patient with medical imaging equipment.
* Image is processed and returns a result.
* A report generated from image result and patient history suggests several possible diagnoses.
* The radiologist confirms a diagnosis.
* Diagnosis is saved in the patient’s history.

For further details on the BPM, please refer to visual aid in Appendix C. The visual model represents the main business process that the system will automate.

**REQUIREMENTS**

1. **Capabilities: Functional Requirements**

Use cases helped identify functional requirements for the development of the dA API and the corresponding user interface. Below is a list of use cases with a brief description of the requirements that were inspired by each.

UC1 describes the basic flow of an exam.

* The user should be able to input patient history.
* The user should be able to input an image.
* The system should return a result from the image processing.
* The system should interpret the image result and patient history to come up with a diagnosis.

UC2 outlines the process of adding a new patient to the Patient table in the database.

* The user should be able to select what type of image is to be processed.
* The user should be able to create a new patient object in a database.

UC3 describes the data entry process.

UC4 handles data management for the actual image processing.

* The user should be able to input new data.
* The system should be able to use the input data.
* The system should provide the user with a way to update existing data.
* Extraneous data should be destroyed.

UC 5 provides an elaborated exam flow.

* The user should be able to input patient history.
* The user should be able to input an image.
* The system should return a result from the image processing.
  + The result should be in the form: “normal or abnormal”
* The system should interpret the combination of the image result and patient history to come up with a diagnosis.
* The application should produce a report.
  + The report should include all previous history.
  + The report should include the original image
  + The report should include the processed image.
  + The result should include the result of the processed image.
* The report should include a diagnosis.
  + All diagnoses should be presented with justification.
    - The justification should list matching symptoms and matching image indications.
  + The report should include a list of alternate diagnoses.
    - The list of alternate diagnoses should be ordered by probability of correctness.

The use case diagram for these functional requirements can be found in Appendix D. Appendix E shows which use cases support which business process steps. Please also refer to Appendix F for the corresponding use cases with full details.

1. **Features: Non-Functional Requirements**

Functional requirements are important because they affect what the system does. Non-functional requirements, on the other hand, describe the capabilities of the system. Oftentimes, non-functional requirements impact the system’s cost and must therefore, be clearly communicated with the customer. As such, non-functional requirements are specific and measurable.

The non-functional requirements listed may not be implemented due to time-constraints. The purpose of recording them here is to demonstrate that we did, in fact, think about these requirements and we now know how important it is to be precise and realistic with the client.

* 1. ***Look and Feel Requirements***
     1. The application should be intuitive.
     2. The application should have no more than two screens open at the same time.
     3. The report should display both the original and the processed image.

* 1. ***Usability Requirements***
     1. The application should inform users of input errors rather than allow input errors.
     2. The application should limit the amount of symptoms to be three or less.
  2. ***Performance Requirements***
     1. It shall take no longer than five minutes to process an image.
     2. There should be no limit to the number of patients that can be entered into the database.
  3. ***Operational Requirements***
     1. The application should run in Visual Studio 2010.
     2. The application should accept jpg images only.
  4. ***Maintainability and Portability Requirements***
     1. The application should allow for frequent updates.
     2. The base cases for image diagnostics should be allowed to be updated by having easy data entry into the database.

1. **Mandated Constraints**

The application will be from the perspective of a radiologist.

The application must run in Microsoft Visual Studio 2010

No more constraints have been identified at this time.

1. **Relevant Facts and Assumptions**
   1. ***Facts***
      1. The CVIP API will only operate in Visual Studio.
      2. CVIP is written in C and must therefore be wrapped in C++ to achieve hierarchical class structuring.
   2. ***Assumptions***
      1. The computers must run a Windows platform and have Visual Studio installed.

**API Design**

1. **Design Pattern: Façade**

The goal of design patterns is to structure the API in a way that it will be easy to re-use in the future. Our diagnostic assistant (dA) API is structured around the Facade pattern. The Facade pattern breaks the system down into subsystems to reduce complexity. With the use of two base APIs and a variety of image processing techniques, subsystems exist naturally. The user interface hides the image processing and is thus the facade of the dA API.   
One consequence of the facade pattern is the number of dependencies that it creates. It is important to maintain a loosely coupled project for versatility.

The facade pattern is a good choice for the dA API development because the main goal is to simplify a complex subsystem so that the end user will be able to process an image without knowing anything at all about the logistics of processing an image.

1. **Underlying API**

The actual image processing application is built on the existing Computer Vision Image Processing(CVIP) API. CVIP is an open source API that provides clients with a variety of image processing capabilities. It is written in plain C which does not have classes built into the language. Since the hierarchy created by classes is an important concept in object-oriented design, wrapper functions are required to replicate the structures.   
    We use the following CVIP functions in our API:

void **display\_image**(const char\*image\_name, IMAGE\_FORMAT format)

<image\_name> name of the image file

<format> the image format

void **view\_image**(Image \*inputImage, char \*imageName)

<inputImage> pointer to the input image

<imageName> character string as the image name in the display window

double **\*hist\_feature**(image \*originalImage, \*labeledImage, int r, int c)

<originalImage> pointer to the original image

<labeledImage> pointer to the labeled image

<r> row coordinate of a point on the labeled image

<c> column coordinate of a point on the labeled image

Returns 5 histogram feautres: mean, standard deviation, skew, energy and entropy

Image **\*read\_Image**(char \*filename, int showmessages)

<filename> - pointer to an character string containing the file name

<showmessages> - show messages

unsigned **GetNoOfCols\_Image**(Image\* image)

<image> pointer to an image

unsigned **GetNoOfRows\_Image**(Image\* image)

<image> pointer to an image

Image \***duplicate\_Image(**const Image \*a)

<a> pointer to an image

Image **\* label**( const Image \*imageP )

<imageP> - pointer to an Image

Image **\*threshold\_segment**(Image \*inputImage, unsigned int

threshval, CVIP\_BOOLEAN thresh\_inbyte)

<inputImage> - pointer to Image structure

<threshval> - threshold value

<thresh\_inbyte>

- CVIP\_NO apply threshval directly to image data;

- CVIP\_YES threshval is CVIP\_BYTE range; remap to

image data range before thresholding.

1. **Data Entities and Relationships**

Several entities have been identified in the requirements. The attributes of each entity are listed below.

Patient: contains a record of each patient, which includes the patientID, patientName, symptoms, and diagnosis, among other attributes.

Diagnoses: contains a list of diagnoses represented by a list of common symptoms.

Image: contains histogram features, and edge-detection results of a base image.

Patient-Diagnoses: One diagnosis may have many patients but a patient may only have one diagnosis.

The relationships listed in this section are illustrated in the class diagram in Appendix G.

The interaction between the use cases described in the Functional Requirements above and the database tables listed in this section is represented in the matrix in Appendix H.

**Use Cases**

1. **Actor Cards**

|  |  |  |
| --- | --- | --- |
| ***Actor Specification*** | | |
| **Actor Name:** Image Processor | | |
| **Type:** Secondary | **Personality:** Boring | **Abstract:** Yes |
| **Role Description:**  The image processor is a system rather than a person. It is responsible for running various tests on a given image and returning a result. | | |
| **Actor Goals:**   * Perform edge-detection * Compare with base data * Return a result | | |
| **Use Cases Involved with:** | | |

|  |  |  |
| --- | --- | --- |
| ***Actor Specification*** | | |
| **Actor Name:** Dr. Oz | | |
| **Type:** Primary | **Personality:** Extroverted | **Abstract:** Yes |
| **Role Description:**  Dr. Oz is a radiologist who works at Hendrick Medical Center. He is the user with the most interaction with the system. | | |
| **Actor Goals:**   * User input * Confirms final diagnosis | | |
| **Use Cases Involved with:**   * Every use case involves Dr. Oz. He is the primary actor because he is the main user. | | |

|  |  |  |
| --- | --- | --- |
| ***Actor Specification*** | | |
| **Actor Name:** Jane Doe | | |
| **Type:** Secondary | **Personality:** Introvert | **Abstract:** Yes |
| **Role Description:**  Jane Doe is a patient who is visiting the radiologist because she is ill. She interacts with the system through the radiologist. She provides all of the necessary input data. | | |
| **Actor Goals:** | | |
| **Use Cases Involved with:** | | |

1. **Scenarios**

|  |  |
| --- | --- |
| **Use Case ID** | UC1 |
| **Use Case** | Simplified Exam |
| **Elaboration** | Basic flow of an exam |
| **Actors** | Patient, Physician |
| **Description** | A patient comes into the office for an exam. The physician enters patient data into the diagnostic assistant application and it returns a probable diagnosis. The physician confirms the diagnosis. The diagnosis is saved in patient’s history. The physician treats the patient. |
| **Priority** | High |
| **Non-Functional Requirements** | (only those associated with this use case, if any) |
| **Assumptions** |  |
| **Source** |  |

|  |  |
| --- | --- |
| **Use Case ID** | UC2 |
| **Use Case** | Add Patient |
| **Elaboration** | Base use case |
| **Actors** | Physician, Patient |
| **Description** | Add patient to database |
| **Pre-conditions** | The patient is a new patient |
| **Flow of Events** | 1. Click add patient button 2. Enter first name, last name, date of birth, physician 3. Automatically assign patient ID number 4. Enter family history 5. Enter past history 6. Enter present medical problems 7. Enter allergies 8. Enter current medications 9. Save patient to database and return to main |
| **Post-conditions** | Patient history is stored in database |
| **Alternative Flows** |  |
| **Priority** | High |
| **Non-Functional Requirements** | (only those associated with this use case, if any) |
| **Assumptions** | No one has the same name. |
| **Outstanding Issues** | Given time constraints there will not be an actual database |

|  |  |
| --- | --- |
| **Use Case ID** | UC3 |
| **Use Case** | Data entry |
| **Elaboration** | Base use case |
| **Actors** | Physician, Patient |
| **Description** | Describe reason for visit and enter data necessary for diagnosis |
| **Pre-conditions** | The patient exists in the database. |
| **Flow of Events** | 1. Find Patient in database 2. Enter date of exam 3. Enter name of physician 4. Enter symptoms 5. Upload image |
| **Post-conditions** | The reason for visit is recorded |
| **Priority** | High |
| **Non-Functional Requirements** | (only those associated with this use case, if any) |
| **Assumptions** | Symptoms will match exactly with those stored in database. Every entry is required except the image. |

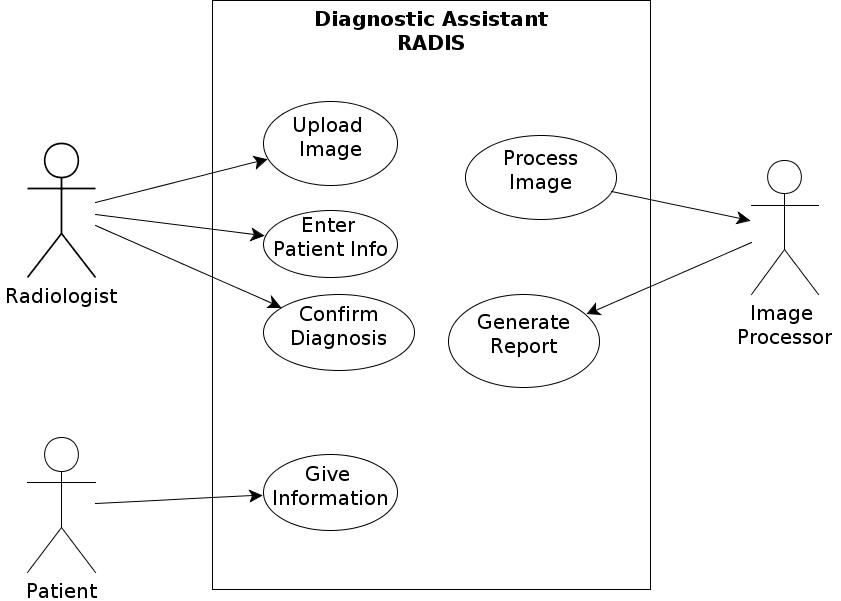
|  |  |
| --- | --- |
| **Use Case ID** | UC4 |
| **Use Case** | Exam Form |
| **Elaboration** | Base use case |
| **Actors** | Physician, Patient |
| **Description** | Describe reason for visit prepare for diagnosis |
| **Pre-conditions** | The patient exists in the database. |
| **Flow of Events** | 1. Find Patient in database 2. Enter first name, last name, date of birth, physician 3. Automatically assign patient ID number 4. Enter family history 5. Enter past history 6. Enter present medical problems 7. Enter allergies 8. Enter current medications 9. Save patient to database and return to main |
| **Post-conditions** | Patient history is stored in database |
| **Alternative Flows** |  |
| **Priority** | High |
| **Non-Functional Requirements** | (only those associated with this use case, if any) |
| **Assumptions** | No one has the same name. Capitalization matters. |

|  |  |
| --- | --- |
| **Use Case ID** | UC4 |
| **Use Case** | Image Processing |
| **Elaboration** | Base use case |
| **Actors** | Physician, Computer |
| **Description** | This use case handles standard data management events for image processing |
| **Pre-conditions** | A patient history and image exist |
| **Flow of Events** | **Create Data**  C1. Physician uploads image  C2. The original image is saved  C3. The image is copied  C3. The copied image is normalized(grayscale, number of pixels, contrast)  C4. Preform histogram features, and edge detection  **Read Data**  R1. Compare numerical values to values that are considered standard values for the image type. These standard values exist in a database.  R2. If the values are similar, return “normal” as a symptom  R2.1 Else return “abnormal” as a symptom  R3. Compare the patient’s symptoms, which now include the result of the image processing, and compare to the diagnosis database. Return the diagnosis with the highest number of matching symptoms.  R4. Return a list of alternate diagnoses ordered by the number of matching symptoms.  R5. Physician confirms or enters his own diagnosis  **Update Data**  U1. Save the original and processed image to patient history.  U2. Save diagnosis to patient history.  **Delete Data**  D1. Delete any intermediate copies of the processed image. |
| **Post-conditions** | An image is processed and saved in patient history |
| **Priority** | High |
| **Assumptions** | An existing API contains image processing functions |

|  |  |
| --- | --- |
| **Use Case ID** | UC5 |
| **Use Case** | In-depth exam |
| **Elaboration** | Elaborated use case |
| **Actors** | Physician, Patient |
| **Description** | The physician goes through the exam recording symptoms and adding images to be processed. The information is then analyzed and results are displayed in a report. |
| **Pre-conditions** | A patient comes into the office for an exam |
| **Flow of Events** | 1. Enter Patient name 2. If patient does not exist in database    1. Add new patient 3. Enter date of Exam 4. Enter reason for visit 5. Enter symptoms 6. If an image is needed for diagnosis    1. Upload image    2. Save and copy original image    3. Process image copy    4. Save processed image    5. Return image result as a symptom 7. Click generate report button to find a diagnosis 8. Display report with original and processed images as well as a diagnosis and a list of alternatives 9. Have physician confirm diagnosis 10. Save diagnosis and images to patient history |
| **Post-conditions** | Patient is diagnosed and the diagnosis and images are saved in patient history. |
| **Alternative Flows** | 1. An image may not be necessary depending on the reason for visit in which case the upload should be ignored.  2. A physician may not agree with any of the provided diagnoses in which case he should be allowed to enter his own. |
| **Priority** | High |
| **Non-Functional Requirements** | (only those associated with this use case, if any) |
| **Assumptions** | Necessary images already exist. |

1. **UML Diagrams**

Arrows from an actor to a use case indicate that the actor triggers the use case, whereas arrows from the system to an actor indicate that the system triggers the use case.



**APPENDIX A  
Glossary of Terms and Acronyms**

CVIP: Computer Vision Image Processing, the foundational API for this project.

CRUD: C, R, U or D, depending on whether a use case creates, reads updates and/or deletes data.

Histogram:

BPM: Business Process Model, a modeling method to describe the scope of the work to be done by the development team.

Edge-detection:

**APPENDIX B**

**Naming Conventions**

Radiologist:

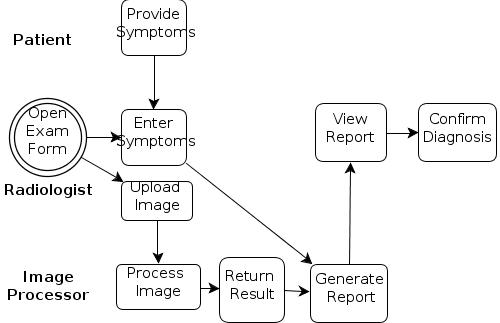
Database: The usage of the term database throughout this project refers to a modifiable text document rather than an actual database.

Image:

Whatever…

All classes will have the format rad\_<description>: ex. rad\_Image  
We will follow the camelCase style within the classes:ex. patientHistory  
All c libraries follow camelCase style as well.

**APPENDIX C  
Scope of Work: Business Process Model**

****

**APPENDIX E  
Transitional Matrix: BPM and Use Cases**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **BPM Steps** | **Use Case** | | | | | |
| **UC 1** | **UC 2** | **UC 3** | **UC 4** | **UC 5** |  |
| Enter Patient Information | X | X | X | X | X |  |
| Upload Image | X |  | X | X | X |  |
| Save and Copy Image |  |  |  |  | X |  |
| Process Image |  |  |  |  | X |  |
| Display Report |  |  |  |  | X |  |
| Confirm Diagnosis | X |  |  |  | X |  |

**APPENDIX F**

**Transitional Matrix: CRUD (Create, Read, Update and Delete)**

This matrix lists one column for every use case and a row for every database table. The cells contain the letters C, R, U or D, depending on whether the use case creates, reads, updates and/or deletes records in the corresponding table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Use Case** | | | | | |
| **UC 1** | **UC 2** | **UC 3** | **UC 4** | **UC 5** |  |
| Patient | C,U | C | C | C,R,U | C,U |  |
| Image | C |  | C |  | C,R,U |  |
|  |  |  |  |  |  |  |

**APPENDIX H  
Class Diagrams**