Kelsey Hilton
Project Plan
Version 1
Last Updated: 9 April 2012
RADIS_projectPlan_01_04_09_2012.pdf

Hendrick Medical Systems has requested a Radiological Application for Diagnostic Imaging and Storage (RADIS). The application is intended for use by radiologists 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.

One reason for an application such as RADIS is to decrease the number of misdiagnoses. Algorithms to confirm diagnoses can be written using data compiled from past cases. Past cases can be analyzed for commonalities associated with a particular diagnosis. RADIS will be able to create a "base case" table/array for images. The base case can then be applied to the current case to return a "normal" or "abnormal" result. RADIS will use past history and current symptoms to 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.

The diagnostic assistant (dA) API will be built on the Computer Vision and Image Processing (CVIP) API. CVIP only runs on the professional edition of Microsoft Visual Studio which means that our dA API has the same limitation, since it depends on CVIP. There is a way to run it in a different environment but, after spending weeks trying to figure out how to use it at all, we do not have the time to figure it out. Instead we assume that the end-user has Visual Studio so that they can run the program.

Despite its restrictions, CVIP is a great development tool. It has a lot of functions built in that compute the math behind the processing but we still have to know about how the processing works so we can piece the CVIP functions together to return a result of "normal" or "abnormal". CVIP is the base of our dAprocess_image() function. It is easiest to think of this function as something that receives an image and returns a Boolean value (normal or abnormal). CVIP hides the math, dA hides the image processing.

A graphical user interface will be constructed from the Qt API and will demonstrate the underlying API's functionality as a diagnostic assistant. A basic outline of the system is as follows:

1. An existing patient comes in for an exam. The patient is assumed to need/have an image to be processed.
2. The doctor inputs the patient's information and symptoms and uploads the existing image.
3. The image is processed by dA.
4. dA returns a result "normal" or "abnormal" and adds it to the patient's symptoms.
5. The system scans a table of diagnoses and tries to find the best match for the patient by counting the number of matching characteristics i.e. symptoms.

Other features include:

- The system will also provide a of list alternate diagnoses in order of their probability of correctness (again, based on the number of matching characteristics).
- If a patient does not exist, he can be added.
- The original and processed images are available and displayed in a report.

All of that to say, functionality is designed with the doctor in mind. It is assumed that the doctor is inputting information he is verbally receiving from the patient. The patient has no interaction with the system except through the doctor. The system assumes the doctor already has an image to upload for the patient if an image is necessary. That doesn't make sense in the case of the new patient but again, this is a prototype. Another piece of software based on our API could include a section for doctors' orders to clarify where the picture is supposed to come from and indicate what the image is for. For simplicity, we are assuming that the doctor has an existing image.

In terms of planning we are working off of a combination of a waterfall and the extreme programming methods. It would have been nice to lay it all out before we began but we don't know enough about the functionality of the base APIs until we started using them. As a result we are taking a very iterative approach and being careful about versioning. If we anticipate functionality that doesn't exist, we will either write it, or if that is unrealistic, change the design of our API. In that regard we are not taking the rigid approach that the waterfall method has to offer. However, we are sticking to some of the documentation that is commonly seen in waterfall projects. Requirements are clearly stated, we have use cases and a project statement. The rest of the specifications are frequently updated which is why careful versioning is necessary and helpful.

Header files will be documented with Doxygen style comments. Functions will be thoroughly explained in the code so that it is obvious to the programmer what the function can be used for.

Test data will be collected manually and entered into text files as a substitute for databases. Normal and abnormal images will be collected from online sources. The normal images will be run through RADIS and combined to form a base case. The application will use

the base cases to compare to the input images to determine normality. Another text file will contain a list of existing patients, their IDs, their names, and any symptoms they present.