Personal Health Record System (PHRS)

Info 620

Fall, 2009

Dr. Song

Sam Chenkin
Chris Grabosky
John Misczak
Summary ............................................................................................................................................. 35
Physical Design ........................................................................................................................................ 37
  ERD ........................................................................................................................................................ 37
  Database Schema .................................................................................................................................. 38
  Validation & Discussion ........................................................................................................................... 38
    Entity Relationship Diagram ................................................................................................................. 38
    Physical Database Diagram .................................................................................................................. 39
Evaluation of Analysis & Design .................................................................................................................. 40
Summary and Lessons learned ................................................................................................................... 42
  John’s Experience .................................................................................................................................. 42
  Sam’s Experience ................................................................................................................................... 43
  Chris’ Experience ................................................................................................................................... 44
Bibliography ................................................................................................................................................ 45
Appendix ..................................................................................................................................................... 46
  Division of the work among team members ........................................................................................... 46
Introduction

With the growing complexity of health care options in the United States, it is becoming clear that a new way of handling patient records is needed. In the current model, most records are stored on paper. This means that the records are hard and expensive to move to new care facilities and the records are ultimately controlled by the health care facilities rather than the patients. To resolve this, there needs to be a mechanism to digitize these records and return control to the patient.

This problem is very complex and has many possible solutions. Our team has decided to peruse one specific aspect of the electronic health care system: returning control to the patient. In the system developed and described in this document, the team has outlined the steps needed and, using UML, described the process to make a system that allows record portability.

Records are still stored at each health care facility in their health record systems. However a copy is stored on a portable device a patient can carry. This device is fully encrypted but will allow health institutions to import any or all records into their systems securely.

The need being met by the team's system is not for urgent care facilities but instead meant for when a patient switches primary physicians or sees a new specialist. In these situations, all old records can be read and imported with ease. However urgent care facilities do play a role with the device. In the event of an emergency, a hospital can look at a small portion of records stored on the device. These include emergency contact information, family history, allergies, prescriptions, and preexisting conditions. This information will prove vital in saving lives while still maintaining some level of privacy for all other records.

To outline all aspects of the system potential use cases were developed and diagrammed. These were further expanded with high level and detailed use case descriptions. These early tasks allowed the team to outline how users would interact with the system. In doing so, all actors were identified early into the development process.

The next step was to develop class diagrams. This was done with both the TCM approach as well as an iterative approach. TCM allowed the team to immediately identify potential classes and remove others from scope. Then for each class's interaction, diagrams were made and modified until the final class diagram was made. Class diagrams allowed for system sequence diagrams, more detailed system diagrams, and state diagrams.
Overview

The Problem Statement
The United States health care providers currently lack a universal and efficient way to transfer and synchronize records. While multiple proprietary systems and formats currently exist, the lack of one common format or the ability to reliably and securely transfer information between institutions, as well as the ability for a patient to digitally carry his/her own records, leads to confusion or delays. These complications limit access to critical patient health information and can have detrimental effects on patients.

Goals
The goal of the Personal Health Record System (PHRS) is to simplify and modernize the way that individuals and medical establishments manage health data. Such a revision in the way that health information is created, updated, and shared will help to improve the efficiency of the health care industry, saving both time and money for all parties involved. PHRS hopes to offer a unified common interface to health care providers and patients without forcing each party to overhaul their existing digital patient management systems and methodologies.

Context & Importance
The current healthcare industry has several different types of systems for managing patient and health data, ranging from traditional paper-and-pencil methods to electronic record keeping. However, few of these systems are interoperable, and all consider the records to be the property of the maintaining intuition rather than the patient. The Personal Health Record System solution will put health records back in the hands of the patient and allow for improved communication and flexibility between previously disengaged parties.

Scope

In-Scope
The PHRS will allow for information relevant to and carried by a particular patient to be synchronized with a doctor’s or hospital’s existing patient management system. The system will provide for secure transmission of information up to the system’s endpoints at the application level. PHRS offers a reliable and safe interchange of patient information regardless of which digital patient record tracking system the endpoints employ.

Out-Scope
The system will not be concerned with the network used for transmitting patient and health information from system endpoint to system endpoint. Moreover, PHRS will not replace a doctor’s or hospital’s existing patient management system, billing system, or other systems used to interact with insurance providers and other medical institutions.

The system will not maintain a proprietary format for storing patient data. It will instead use an internationally recognized standard. The system is not concerned with the specific format for
storing data; it is format agnostic. Instead, PHRS provides a method for exchanging and protecting standardized records.

The system shall now attempt to ascertain the validating of a participating medical facility. This verification will be accomplished outside of the system.

Requirements

Functional Requirements

- The system shall maintain basic patient information and permissions on a central server
- The system shall allow patients to create new accounts
- The system shall allow users to store critical information such as allergies and conditions on a central server
- The system shall allow certified emergency medical facilities to access critical information such as allergies and conditions as needed
- The system shall maintain full patient medical records in an encrypted format on a patient’s personal storage device
- The system shall provide a common API for health information systems to update and read records
- The system shall allow patients to specify entities with permission to access their data
- The system shall allow approved certified health providers to append information to the encrypted storage device
- The system shall allow patients to view all records stored on the device
- The system shall allow patients to update basic and insurance information on the device
- The system shall allow patients to access real-time access logs via a web browser

Data Requirements

- Patient Contact Information
- Patient Global Unique Identifier for use in external medical records systems
- Patient Emergency Information
- Patient Insurance Information
- Patient Billing Information
- Patient Medical Records
- Patient Medical Records Permissions
- Encryption Keys
- Patient Medical Record Access Records

Business Rules and Logic

- Each personal device has one patient’s information on it – that of the owner
- Only the patient may edit contact information, billing information, and insurance information
- Only certified medical facilities may append medical records to the device
- A specific system station “checkpoint” at a doctor’s office or hospital can only interface with one personal device at a time.
- The office or hospital can have multiple system checkpoints in order to interface with multiple patients’ devices concurrently.
- Medical records are only appended, never modified.
Non-Functional Requirements

Patients Can
- Update personal information on their device through a screen-based software application installed on their computers which communicates to the device over USB

Health Care Providers Can
- Access the system through a screen-based software application
- The patient health information systems used by health care providers will inform the PHRS system of changes to patient records via a common published API.
- If PHRS is failing, the current patient management system must be unaffected
- If the health care institution’s patient management system is failing, PHRS must be unaffected
- Encryption is employed on all files, transfer links, and backups

Hardware & Software

Software:
- Hospital/Doctor’s Office/Medical Establishment
  - Proprietary software system for patient management system
  - Key plugin/module that ties patient management system to PHRS
  - PHRS software
  - Windows XP/Vista/7 or Mac OS X 10.5-10.6 for personal computers inside establishment
- Patient
  - Windows XP/Vista/7 or Mac OS X 10.5-10.6 for home computer
  - PHRS software to allow home synchronization

Hardware:
- Hospital/Doctor’s Office/Medical Establishment
  - Existing computers and servers that run patient management system
  - Endpoint stations that interface with personal patient devices
- Patient
  - Handheld device that interfaces via USB to computer or station

Assumptions
- Existing patient management systems will implement the PHRS API.
- Doctor’s offices, hospitals, and other medical establishments will limit access to the devices and stations that interface with the PHRS, and will not allow unsupervised access to said devices.
- All data, forms, records, and other information used by medical establishments will be formatted according to health industry standards
Use Case Model

Actors and Their Goals

<table>
<thead>
<tr>
<th>Actor</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Patient</strong></td>
<td>• Manage Personal Account</td>
</tr>
<tr>
<td></td>
<td>• Manage data on Device</td>
</tr>
<tr>
<td></td>
<td>• Manage Permissions</td>
</tr>
<tr>
<td></td>
<td>• Manage Personal Emergency Info</td>
</tr>
<tr>
<td><strong>Emergency Medical Facility Employee</strong></td>
<td>• View emergency medical information of a patient</td>
</tr>
<tr>
<td><strong>Non-Emergency Medical Facility Employee</strong></td>
<td>• Initiate sync with in house medical records system</td>
</tr>
<tr>
<td><strong>InHouse HIS</strong></td>
<td>• Interact with health information device</td>
</tr>
</tbody>
</table>
Use case diagram
Use case description

**Update Emergency Info on Server (John)**

<table>
<thead>
<tr>
<th>USE CASE #</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>USE CASE Name</strong></td>
<td>Update Emergency Info on Server</td>
</tr>
<tr>
<td><strong>ACTOR</strong></td>
<td>Patient</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>To update the patient’s emergency information for view and use by medical establishments, doctors, emergency workers, and insurance companies.</td>
</tr>
<tr>
<td><strong>Overview and scope</strong></td>
<td>Shows the process that the patient undertakes in order to update his or her emergency information on the central server. This information is what emergency workers and medical workers and facilities use in the event of an urgent situation, when the patient is injured or able to speak for him or herself.</td>
</tr>
<tr>
<td><strong>Level</strong></td>
<td>Primary</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>Patient has created an account for use with the Personal Health Record System and has logged into the associated web service. Patient has obtained information in another medium about his or her emergency information, including drug prescriptions, allergies, family history, contact persons, and preexisting conditions.</td>
</tr>
<tr>
<td><strong>Postconditions</strong></td>
<td>The patient was informed of the completion of record additions and updates. The patient’s emergency information was added to the patient’s account and is stored on the appropriate server.</td>
</tr>
<tr>
<td><strong>Trigger</strong></td>
<td>A subscriber clicks the “Update Emergency Info” option on the appropriate PHRS web service page.</td>
</tr>
<tr>
<td><strong>Included Use Cases</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Extending Use Cases</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

**MAIN SUCCESSFUL SCENARIO**

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Patient clicks “Update Emergency Info” option on PHRS page.</td>
<td>2. System displays emergency record types to view or update.</td>
</tr>
<tr>
<td>3. Patient chooses a record type by clicking on the appropriately labeled button.</td>
<td>4. System displays any current emergency records on the screen for that record type.</td>
</tr>
<tr>
<td>5. Patient chooses option to update listing for that type of emergency record.</td>
<td>6. System displays form entry page for subscriber to enter information.</td>
</tr>
<tr>
<td>Step</td>
<td>Branching Action</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>7.</strong> Patient enters appropriate information for the selected record type.</td>
<td></td>
</tr>
<tr>
<td><strong>8.</strong> Patient clicks “Save” to finalize the record.</td>
<td><strong>9.</strong> System adds the record that has just been finalized by the patient to the existing list of records for that record type.</td>
</tr>
<tr>
<td><strong>11.</strong> Patient reviews updated listing to check that information is correct.</td>
<td><strong>10.</strong> System displays updated record listing for the selected record type.</td>
</tr>
<tr>
<td><strong>12.</strong> Patient disconnects from system.</td>
<td></td>
</tr>
</tbody>
</table>

### OTHER SUCCESSFUL SCENARIOS

<table>
<thead>
<tr>
<th>Step</th>
<th>Branching Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8a.</strong> Record cannot be saved for addition to listing because it has incorrect or incomplete information.</td>
<td>Patient reviews the information he/she entered for the record in order to find the mistake that was made and correct it.</td>
</tr>
<tr>
<td><strong>11a.</strong> Updated listing contains information that the patient recognizes as not correct.</td>
<td>Patient goes back into the form entry page for that record and changes the information to be correct.</td>
</tr>
</tbody>
</table>

### UNSUCCESSFUL SCENARIOS

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3a.</strong> Patient cannot find an appropriate emergency record type for the information he/she wishes to enter.</td>
<td>Patient contacts the system developer’s customer service department for instructions on what to do with the information that is to be entered.</td>
</tr>
<tr>
<td><strong>4a.</strong> System cannot retrieve record listing information from server due to network difficulties or server downtime.</td>
<td>Patient disconnects from system with plans to return later and update record and contacts system developer’s customer service department.</td>
</tr>
</tbody>
</table>

### Priority in scheduling

- First

### Frequency

- Often (several times a day) when the user first logs into their account in order to set up all necessary information. Infrequently (once every few months) after that.

### Other non-functional requirements

- The confirmation page for the updated record will be shown to the patient immediately after the “Save” button is clicked. The record listing should propagate to all involved servers within 24 hours of a record being updated.
View Emergency Medical Info (Sam)

<table>
<thead>
<tr>
<th>USE CASE #</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE CASE Name</td>
<td>View Emergency Info</td>
</tr>
<tr>
<td>ACTOR</td>
<td>Emergency Medical Facility</td>
</tr>
<tr>
<td>Goal</td>
<td>To view the patient’s emergency medical information from the system in the event of an emergency.</td>
</tr>
<tr>
<td>Overview and scope</td>
<td>Shows the process of viewing a patient’s emergency medical information for use by emergency workers when the patient is unconsciousness or similarly impaired. This information may be used by the facility to contact the emergency contacts listed by the patient for urgent decision making or information.</td>
</tr>
<tr>
<td>Level</td>
<td>Primary</td>
</tr>
</tbody>
</table>
| Preconditions | • Patient has created an account for use with the Personal Health Record System and updated his or her emergency contact information via the Update Emergency Info on Server use case  
• Emergency medical facility employee is associated with an Emergency Medical Facility with permissions to access Emergency Medical Records |
| Postconditions | An entry in the access log was created indicating the emergency medical facility employee’s accessed the emergency medical records of the patient |
| Trigger | An emergency medical facility employee clicks the “View Emergency Info” option on the appropriate PHRS web service page. |
| Included Use Cases | None |
| Extending Use Cases | None |

<table>
<thead>
<tr>
<th>MAIN SUCCESSFUL SCENARIO</th>
<th>Actor Action</th>
<th>System Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.</td>
<td>Employee logs in with username, password, facilityID</td>
<td>14. Verifies the username and password and emergency access permissions</td>
</tr>
<tr>
<td>Step Branching Action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Employee enters patient’s name in search screen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. Searches for patients by name and displays list of possible matches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. Employee selects a patient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. System displays patient summary information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20. Employee selects emergency record type for viewing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21. System logs access and displays records</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22. Repeats steps 8 &amp; 9 as necessary before</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23. Employee requests to close window.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24. System Closes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### OTHER SUCCESSFUL SCENARIOS

<table>
<thead>
<tr>
<th>Step Branching Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a. Employee does not enter appropriate login information</td>
</tr>
<tr>
<td>User is presented with an error and redirected back to step 1</td>
</tr>
<tr>
<td>4a. Employee does not know users name</td>
</tr>
<tr>
<td>User may search by SSN and restrict by birthdate range</td>
</tr>
<tr>
<td>6a. Employee cannot find the appropriate patient or patient search list is empty</td>
</tr>
<tr>
<td>User returns to step 4 to repeat search</td>
</tr>
</tbody>
</table>

### UNSUCCESSFUL SCENARIOS

<table>
<thead>
<tr>
<th>Conditions Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2a. Employee does not have permission to view emergency records</td>
</tr>
<tr>
<td>User is presented with an error and is not able to view records.</td>
</tr>
<tr>
<td>8a. Patient has no emergency records in the system</td>
</tr>
<tr>
<td>User is presented with an error message and is able to go to step 10 or return to Step 1</td>
</tr>
</tbody>
</table>

### Priority in scheduling

First

### Frequency

Often (hundreds of times a day) as emergency cases enter Emergency Medical Facilities

### Other non-functional requirements

This use case requires a very very high uptime as patient lives may be at stake. Redundant servers and alternate access paths (ex: phone center) are required.

### Superordinates

None

### Developer

Sam Chenkin

### Creation date and last modified date

December 6, 2009
Read Records from Device (Chris)

<table>
<thead>
<tr>
<th>USE CASE #</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE CASE Name</td>
<td>Read Records From Device</td>
</tr>
<tr>
<td>ACTOR</td>
<td>In House HIS</td>
</tr>
<tr>
<td>Goal (1 phrase)</td>
<td>Allows a health information system to read any records already stored on a patient’s personal storage device.</td>
</tr>
<tr>
<td>Overview and scope</td>
<td>An in-house health information system has the ability to take any records, personal information, or emergency information that is stored on a patient’s personal storage device and read it. Reading entails pulling up the information in a read-only manner (cannot be edited) or being pulled into the in-house system. To read anything other than the emergency information or basic personal information (like contact information), the device’s encryption keys must first be acquired. This is done with the Download Encryption Keys Use Case.</td>
</tr>
<tr>
<td>Level</td>
<td>Primary</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The In House HIS has the capability of acquiring the device’s decryption key from the server</td>
</tr>
<tr>
<td>Postconditions</td>
<td>The In House HIS has imported all (or a subset of) records from the patient’s device.</td>
</tr>
<tr>
<td>Trigger</td>
<td>Plugging the patient device into the In House HIS</td>
</tr>
<tr>
<td>Included Use Cases</td>
<td>Download Encryption Keys</td>
</tr>
<tr>
<td>Extending Use Cases</td>
<td></td>
</tr>
<tr>
<td>MAIN SUCCESSFUL SCENARIO</td>
<td>Actor Action</td>
</tr>
<tr>
<td></td>
<td>1. HIS has patient device plugged into system</td>
</tr>
<tr>
<td>Step</td>
<td>Branching Action</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
</tr>
<tr>
<td>2. INCLUDE <em>Download Encryption Keys</em></td>
<td>3. System returns decryption key</td>
</tr>
<tr>
<td>4. HIS requests list of records on device</td>
<td>5. System returns list of records on device</td>
</tr>
<tr>
<td>6. HIS requests a record</td>
<td>7. System returns a record</td>
</tr>
<tr>
<td>8. HIS requests a record’s checksum</td>
<td>9. System returns a record’s checksum</td>
</tr>
<tr>
<td>10. HIS computes checksum of transferred record</td>
<td></td>
</tr>
<tr>
<td>11. HIS compares checksum of transferred record and checksum system returned of that record to make sure they match</td>
<td></td>
</tr>
<tr>
<td>12. Disconnect device</td>
<td></td>
</tr>
</tbody>
</table>

**OTHER SUCCESSFUL SCENARIOS**

<table>
<thead>
<tr>
<th>Step</th>
<th>Branching Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-11</td>
<td>Repeat these steps for as many records as wanted</td>
</tr>
</tbody>
</table>

**UNSUCCESSFUL SCENARIOS**

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>At step 3, 5, 7, 9 if system does not return anything</td>
<td>Abort the transaction, inform HIS to disconnect device and restart from step 1.</td>
</tr>
<tr>
<td>11a. Checksums differ</td>
<td>Disregard last record imported and repeat from step 6.</td>
</tr>
</tbody>
</table>

**Priority in scheduling**

- This is to be completed after emergency contact use cases are completed

**Frequency**

- Whenever a patient moves to a new doctor’s office (not urgent care facility)

**Other non-functional requirements**

- Transfer of records must be high speed and secure.

**Superordinates**
<table>
<thead>
<tr>
<th>Developer</th>
<th>Chris Grabosky</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Creation date and last modified date</strong></td>
<td>Last Modified 12/6/09</td>
</tr>
<tr>
<td><strong>Other Comments</strong></td>
<td></td>
</tr>
</tbody>
</table>
Activity diagrams

**Update Emergency Info on Server (John)**

<table>
<thead>
<tr>
<th>Patient</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click Update Emergency Info Button</td>
<td>Display Emergency Records to Date</td>
</tr>
<tr>
<td>Choose Record Type</td>
<td>Display Emergency Records for Requested Type</td>
</tr>
<tr>
<td>Enter Information</td>
<td>Changes Required?</td>
</tr>
<tr>
<td>Click Save</td>
<td>Yes</td>
</tr>
<tr>
<td>Review Record Listing</td>
<td>No</td>
</tr>
<tr>
<td>Information Correct?</td>
<td>Add Record To System</td>
</tr>
<tr>
<td>No</td>
<td>Display Updated Record Listing</td>
</tr>
<tr>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Disconnect
View Emergency Medical Info (Sam)

Emergency Medical Facility Employee

Enter Login Information

System

Verify Login Info

Invalid

Valid

Display Patient Search Window

Enter Patient Name

Search for Patient

Display Patient List

Select Patient

Patient Not Found

Patient Found

Display Patient Summary

Select Emergency Record Type

Display Emergency Record Type

Log Access

Review Record

View Another Record Type

Done

Disconnect

Patient

Patient

Not

Found

Found

May 3, 2011 :: Version 1.0 (Draft)
Read Records from Device (Chris)

In House HIS

- Plug In Device
- Choose Read Records Option
- Request Record For Download
- Display All Records

System

- System Has Key?
  - No: Download Decryption Key
  - Yes: Decrypt Storage Device

- Display Home Screen
- Display All Records
- Transfer Record
- Transfer Checksum
- Valid Transfer?
  - Yes: Transfer Another Record?
  - No: Re-transfer

Transfer Another Record?

- Disconnect Device
Discussion

The above use case diagram, along with the detailed use case descriptions and activity diagrams represent the functionality that is considered in-scope for the Personal Health Record System. The majority of the use cases for the system as found in the use case diagram fall under the Patient actor is appropriate considering that the system is intended to be used for individuals, hence the “Personal” in the title. All of the necessary functionality for the patient is captured in the use case diagram, ranging from managing one’s account to synchronizing the personal storage device. For the large part, these use cases are exclusively acted upon by the Patient actor with the lone exception of “Manage Insurance Info”, which is of course shared with the Insurance Company actor. The decision to have these use cases be exclusive to the Patient was due to the conclusion that the system would have to be accessible to the patient in his/her own home, often when he/she was alone and wanted immediate feedback and satisfaction from the system. In order to be of some value to patients, the system would have to set itself apart from traditional models and processes by giving more (and in some cases, complete) control to the patient.

The rest of the actors in the use case diagram are equally important despite not having nearly as many use cases attached to them. One of the biggest benefits of the system is providing a current version of emergency information to first responders and emergency facilities; as such, an entire use case was dedicated to the aforementioned individuals who would require emergency information in order to prepare and administer the best possible care. Alternatively, non-emergency medical facility employees would need to bring their in-house records with those on the patient’s device, which would have the most current and therefore useful set. This use case would not have the urgency of the emergency medical facility employee’s use case, but would also be highly important due to constant concerns over data integrity in the medical industry.

Lastly, the actual in-house systems that medical facilities use would have its own set of use cases for the Personal Health Record System. Since security and confidentiality is of the utmost importance when dealing with medical information, the Personal Health Record System provides a level of encryption for the personal storage device a patient uses. Therefore, any in-house system wishing to interface with the device must have the appropriate encryption keys. Since this necessary synchronous relationship can be complicated at times, an entire use case was devoted to it. Once the proper decryption keys have been acquired and put to use, the in-house system needs to be able to read records off the device for storage in the in-house system as well as add or modify records on the device after the patient’s current visit is complete.

The included detailed use case descriptions and activity diagrams for the chosen primary use cases represent a broad sampling of use cases, as the team felt it would be best to include as many actors as possible when picking what use cases to focus on. Furthermore, they represent some of the more important use cases for the system; namely, updating emergency info (as done by the patient), viewing emergency info (as done by an emergency medical facility employee), and reading records from the device (as done by an in-house system). While not completely exhaustive of all actors or even use cases,
these diagrams and models provide a snapshot of just how complex the functionality of the system will end up being, which enabled the team to better prepare for the ensuing class and interaction diagrams.
## Class Model

**TCM table**

<table>
<thead>
<tr>
<th>Nouns</th>
<th>Class Elimination Rules Applied (Step 2)</th>
<th>Class Categories Applied (Step 3)</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Meta-language (CER6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual</td>
<td>Redundant (CER1), Vague (CER3) – Use User Account</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medical Establishment</td>
<td>Redundant (CER1) – use Health Care Provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Meta-language (CER6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface</td>
<td>Irrelevant Classes (CER2)</td>
<td>Organizations (CC4)</td>
<td>X</td>
</tr>
<tr>
<td>Health Care Provider</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patient</td>
<td>Redundant (CER1) – use User Account</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allergy</td>
<td>Concepts with Properties (CC7), Things in a container (CC12)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Drug</td>
<td>Physical Things with value (CC3), Things in a container (CC12)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Family History</td>
<td>Concepts with Properties (CC7), Things in a container (CC12)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Emergency Contact</td>
<td>Roles of People (CC1), Things in a container (CC12)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>Concepts with Properties (CC7), Things in a container (CC12)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Patient Management System</td>
<td>Irrelevant (CER2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User-Entered Emergency Record</td>
<td></td>
<td>Containers of other things (CC11)</td>
<td>X</td>
</tr>
<tr>
<td>Patient Record</td>
<td>Rename “Patient Medical Record”</td>
<td>Concepts with Properties (CC7), Containers of other things (CC11)</td>
<td>X</td>
</tr>
<tr>
<td>Information</td>
<td>Meta-language (CER6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hospital</td>
<td>Redundant (CER1) – use Health Care Provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>Meta-language (CER6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insurance Provider</td>
<td>Rename “Insurance Company”</td>
<td>Organizations (CC4)</td>
<td>X</td>
</tr>
<tr>
<td>Medical Institution</td>
<td>Redundant (CER1) – use Health Care Provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>Vague (CER3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Class Definitions

- **Device**: The device class stores basic information about a device. A device stores the actual patient records.
- **Access Log**: The Access Log class stores records of access by a healthcare provider. The actual medical records are only stored on the device, but the access of these medical records is stored in our system.
• **User-Entered Emergency Medical Records**: Users enter limited emergency medical info which is stored on the server and can be accessed by an emergency medical facility. Each type of record is a dedicated sub-class of this super-class. All are the same type of data and are associated with the patient, but each have their own fields.

• **Permission**: Defines the permission each health care provider has to different types of a patient’s records.

• **Patient Medical Records**: Patient medical records are a standardized format stored on the device. The actual format is outside of the scope of this system.

**Association Definitions**

- **Provider Account::Health Care Provider**: Provider accounts are associated with a medical provider facility. This relationship is used to determine which medical records an account has access to.

- **Health Care Provider::Permission**: Health Care Providers have permissions for patient records. This relationship is used to determine which patients and patient record types a health care provider is allowed to have. These permissions are set by the patient.

- **Permission::Patient Account**: Patients have permissions that they define. These permissions determine which medical facilities can see which parts of their medical records.

- **Patient Account::Device**: Patients have multiple devices (one at a time). These devices actually store their medical records. We keep track of them to manage encryption keys and access logs.

- **Device::Access Log**: Allows for the logging of accesses to patient medical records.

**Discussion**

To properly identify all classes and their interactions, the team utilized a modified TCM approach. In this approach, a narrative was created that explained the purpose of the system and all elements that were considered in scope. Then, using noun identification, a series of potential classes was identified. These potential classes were then either qualified or disqualified as classes using TCM Class Elimination Rules (CERs) or class categories. The results of this work can be found in the TCM table and all identified classes were carried into the Class Diagrams.

The first Class Diagram, representing Stage 1 of our iteration, was created without doing the TCM approach. The result of this is a large number of classes with overly-complicated relationships and little cohesion. That’s not to say that the first diagram was a waste of time. In fact, it was quite the opposite. As the team adopted the TCM approach, many of the classes identified in the Stage 1 diagram were carried into the TCM table and later to the Stage 2 diagram. This meant that the team was on the right track to developing the final class diagram. It showed that relationships could be simplified, classes could be reduced, and more analysis was needed.

The work from the TCM table and the analysis of the problems in the Stage 1 diagram was carried into the Stage 2 diagram. Many superfluous classes were removed and the amorphous cloud of connections was simplified. At this stage, the team was concentrating on the finalized set of classes and any need-to-
know relationships. Identifying cardinality would come at a later stage when all classes and connections were identified and finalized.

At Stage 2, the diagram became much more streamlined and more representative of the problem domain. However it was neither ideal nor complete. This meant there was a need for Stage 3. Here, generalizations began to be implemented. An “Emergency Medical Record” class was identified to extrapolate commonalities between all emergency records like allergies, family history, and contact information. Patient Medical Records, which were previously left out as their storage was somewhat out of scope, were introduced as a separate database with a connection to the Patient Account. User Account also became a generalized class for Patient Account and Provider Account as both would need access to the system. The Encryption class was also made redundant. Since each device only had one encryption key, the key could be made a property of the device rather than a separate class. Lastly, some connections were reordered as more analysis was done on how classes would interact. Most notably here is the Permission class. Two types of connections were added here to represent the limited access that health care providers had as well as the permissions which the patient had created.

Stage 3 represented the nearly finalized design. As a result, Stage 4 was simply adding cardinality to all connections previously established. There were also some modifications to properties within the classes, most notably the Permission class. The Domain property did not reflect what was really being stored within the class so it was changed to Medical Record Type. This is meant to represent contextual information, like which providers get access to information all of the time as opposed to emergency situations, as well as content related information, like which records are available to be seen or synchronized. This better represented the implementation of the class than Domain could.

The team felt that the Class Diagram was finalized at this point. However there was discussion of whether the recently identified Patient Medical Records database should be within the diagram. While it certainly held records associated to a patient, it was more associated with a specific device. Records are stored on a device in a format not developed by or for PHRS. Since the format was specifically out of scope and the records are not created by a patient (they are created by healthcare providers and synchronized onto a device), the Stage 5 diagram was created. In this diagram, the system was split into two different problem domains. Where the team spent most of the previous time and effort, the Core System Domain, the team placed the updated Stage 4 Class Diagram. Then a new problem domain, the Device Domain, was created. This domain stores the Patient Medical Records. While the storage format is still out of scope, leaving this entirely out of the diagram seemed incomplete. Since it is stored within the device itself and not on any PHRS system, there are no direct associations between the problem domains.

Utilizing a modified TCM approach and a continually evolving diagram, the team was finally able to create the finalized UML Class Diagram in Stage 5. A sixth stage was attempted with the idea of putting header classes representing each emergency record type (allergy, drug, family history, emergency contact, and condition), however the team ultimately decided to not pursue this. Further analysis of why can be found in the design class discussion section.
System Design

System Sequence Diagrams

Update Emergency Info on Server (John)

1: Verify Identity (PatientID, Username, Password)
   displayHomeScreen()

Loop [For each Record Type]
   2: selectRecordType(type)
      displayRecordListing()

Loop [For each New or Modified Record]
   3: updateRecordDetails(recordID, fields...)
      displayRecordDetails()
   4: confirmDetails()
      displayRecordListing()

12: disconnectFromSystem
View Emergency Medical Info (Sam)

::Emergency Medical Facility

1: VerifyIdentity(FacilityID, Username, Password, HardwareTokenKey)
   displayEmergencyOptions()

2: SearchForPatient(FirstName, LastName, Address, DOB, SSN)
   showPatientResults()
   displayEmergencyInfoSummary()

3: requestPatient(searchResultID)
   displayEmergencyInfoSummary()

Loop [As Needed]

4: requestDetailedInfo(type)
   displayDetailedInfo()
   displayEmergencyInfoSummary()

5: requestPatientOverview()
   displayEmergencyInfoSummary()

6: disconnectFromSystem
Read Records from Device (Chris)

1: initiateDecryption(decryptionKey)
   confirmKey()

2: selectReadRecordsOption()
   displayAllRecords()

[loop [for each record]]
3: requestRecordFromDevice(recordId)
   transferRecordDetails(recordId)

[loop [for each record]]
4: validateImportedRecords(recordId)
   returnStoredRecordsChecksumForImportedRecord(recordId)

5: disconnectFromDevice
Read Records from Device (Chris)

1. Top Package in House HIS
2. Connect Device
3. Request Decryption Key
   - requestDecryptionKey(hardwareID)
4. Decryption Key
   - inflateDecryption(decryptionKey)
5. Check if Key is Valid
   - Was A Valid Key
6. Select Records Option
   - selectReadRecordsOption()
7. Loop For Each Record
   - requestRecordFromDevice(recordID)
8. Get Record
   - getRecord(recordID)
9. Validate Imported Record
   - validateImportedRecord(recordID)
10. Get Record Checksum
    - getRecordChecksum(recordID)
11. Destroy Record
    - Destroy(recordID)
12. Disconnect Device

Destroy()
**Design Class Diagram**

**Insurance Company**
- Name
- Address
- Insurance Policies :: Collection
  - findCompany(str searchValue)
  - listActivePatients()

**Patient Insurance Policy**
- Group ID
- Plan ID
- START DATE
- END DATE
- Patient Account
- Insurance Company
  - getInsuranceCo()
  - getPatient()

**Patient Account**
- SSN
- Birthdate
- User-Entered Emergency Records :: Collection
  - addInsurancePolicy()
  - getAllInsurancePolicies()
  - getCurrentInsurancePolicies()
  - addDevice(int HWID)
  - deactivateDevice(int HWID)
  - findPatient(int SSN)
  - listPatientPermissions()
  - addPatientPermission(int providerID, Permission permissionInfo)
  - removePatientPermission(int permissionId)
  - getDevice()
  - addEmergencyRecord(Record record)
  - removeEmergencyRecord(int recordID)
  - listEmergencyRecords(str RecordType)
  - updateEmergencyRecords(recordid, Record record)

**User Account**
- Username
- Web Password
- Name
- Email
  - createAccount(username, password)
  - modifyPassword(str newPassword)
  - modifyEmail(str newEmail)
  - verifyPassword(username, str password)

**Device**
- HW ID
- Active Date
- Expiration Date
- Encryption Key
  - getAccessLogEntries(datetime startDate, datetime endDate)

**Provider Account**
- Organization Name
  - Access Logs :: Collection
  - Health Care Provider
    - getAccessLogEntries(datetime startDate, datetime endDate)
    - addAccessLogEntry(LogEntry entry)
    - checkEmergencyAccessStatus()
    - getfacilityID()

**User-Entered Emergency Records**
- Patient Date Added
  - getDetails()

**User-Entered Emergency Records**
- Patient Date Added
  - getDetails()

**Emergency Contact**
- Relation
- Phone
- Phone
- Address
- Notes

**Allergy**
- Name
- Dose
- Notes

**Drug**
- Name
- Dose
- Notes

**Family History**
- Relation
- Condition
- Notes

**Condition**
- Name
- Date Diagnosed
- Notes

**Access Log**
- Date
- Access Type

**Access to records Limited by Permissions**

**Permission**
- Medical Record Type
- Permission
- Date Granted
  - getDetails()
Validation & Discussion
The included system sequence diagrams, sequence diagrams, and design class diagram for the Personal Health Record System model the various ways that different classes and objects in the system interact and interoperate with one another.

Sequence Diagrams

**Update Emergency Info on Server Explanation**
The “Update Emergency Info on Server” use case sequence diagram depicts the underpinnings of the system’s relationship with the patient actor. The patient first logs into the system and has his/her identity verified by the information tied to his/her account. Once verified, the system displays a list of services to the patient, who then chooses to modify the emergency records associated with the account.

The system pulls up all preexisting emergency records for all the different record types, allowing the patient to drill down and select the exact record type they wish to update. Once selected, the system displays all of the records stored for that record type, with the option to update the existing records or add a new one. Once the patient has updated a particular record, the system will display a confirmation page to the patient in order to ensure that any information entered was indeed what the patient wanted to include. Once satisfied, the patient confirms and, if he/she so chooses, can repeat the process for as many records and record types as need be. If the patient has finished updating all of the records he/she desires, he/she cannot disconnect, thus killing all of the system objects that were being used to keep track of the user’s proceedings.

**View Emergency Info Explanation**
In the View Emergency Medical Info detailed system use case we can see the process the system uses to access and display these records.

The first step is checking to see if a user has permissions to access emergency records in general. This is done by asking the health care provider for all of its records which in turn requires asking each permission for its details (in a loop).

Next the system asks for a list of patients matching a first and last name from a generic patient account object. Once the user selects the appropriate user the system gets that particular patient object from the system.

This object is then asked for all of its emergency records of a particular type for as many record types as the user would like. Getting the detailed emergency record requires looping through each record of a particular type and asking each for its details. The User Entered Emergency Record superclass is used to represent each of the individual sub-classes, since any of them could be accessed in this loop.

Finally, a new Access Log object is created and added to the logged in user account.
This detailed sequence diagram does not show all return messages, but it shows the important one. It also illustrates the complexity of our system, and the need to follow several associations to get the needed information.

**Read Records from Device**
The purpose of the "Read Records from Device" use case is to allow a user to connect a patient's personal records device and either selectively import individual records or import all records into the health care facility's in house HIS. This is performed when a patient’s personal device is plugged into the system. The system can handle one device at a time.

Before this use case is handled, someone must first log into the in house HIS. After this, the device is connected. Assuming the device's decryption key is inside the system (if not, it is requested online -- see the activity diagram for more details), the device is decrypted. Then the HIS selects from the list of records which to import.

Each record that was selected is imported one at a time. Following the record, the HIS requests a checksum of the file that was stored on the device. The HIS then computes a checksum of the file it received and compares the two checksums. This step is not shown on the diagram because it is handled by the HIS itself, not the system the team has developed. As a result, it is out of scope. However the system the team has developed has this functionality to ensure the record was transmitted into the HIS without error.

Assuming the record was transferred correctly, the record is imported into the HIS. These steps continue until all selected records have been imported. Upon completion of the transfer, the device is disconnected.

**Design Class Diagram**
The design class diagram, while seemingly complex at first sight, is quite close to the system’s analysis class diagram. All of the same classes are there, and thus, the same logic and cardinality between the classes is applied. However, reference attributes and operations were added to many of the classes for the purpose of making the diagram more abstraction friendly when it comes time to actually construct the system. The cornerstone of the design class diagram is the patient account, which not only has reference set attributes for the User-Entered Emergency Records, Devices, Permissions, and Patient Insurance Policy classes, but also operations that deal with those same classes that allow it to populate and maintain the aforementioned attributes.

The justification behind this is relatively straightforward; while the system will be dealing with one patient account at a time, that single account can contain a wealth of insurance policies, emergency records, access and usage permissions, or devices that are permitted to synchronize and operate with it. Furthermore, the navigability between Patient Account and User-Entered Emergency Records and Permissions is a one-way one because the emergency records and permissions objects don’t store any information about a particular patient in them, but the patient account requires the emergency record and permission information in order to be completely functional for all actors’ use cases. Likewise, the navigability between Patient Account and the Patient Insurance Policy and Device classes is two-way, as
both sides of the associations between Patient Account and either of those two classes have need of some information stored on the other side of the association.

Additionally, the inclusion of reference attributes and navigability allows for several classes to obtain information about other classes in ways that would not be possible otherwise. For example, the Provider Account class is not directly associated with the Permission class, but still receives permission collection information wrapped inside a Health Care Provider object, which serves as one of Provider Account’s reference attributes. Since the association between Provider Account and Health Care Provider has two-way navigability, Health Care Provider also receives a collection of provider accounts on its side of the association, which will include all attributes contained in a Provider Account object, such as a collection of access logs. By utilizing two-way navigability and any involved reference attributes in this design class diagram, our team allowed such information to flow back and forth between classes, thereby granting us more effective management of the involved classes and their associations.

Summary
While designing and compiling our sequence diagrams and design class diagram, the team was faced with two distinct alternatives that altered the way the rest of the system would function. The introduction of operations and additional attributes in the form of references and reference sets caused our team to debate whether a group of new classes should be added to our design class diagram that would act as intermediaries between the record types (Allergy, Drug, Family History, etc.) and the class “User-Entered Emergency Records”. These proposed new classes, tentatively given the title of “Record Headers” for each Record type, would act as containers for the actual record type classes. Including this set of classes in the sequence diagrams and eventual design class diagram was the initial choice of the group, and the appropriate diagrams were drawn up and discussed internally.

However, it was eventually decided that the inclusion of these header classes and objects provided little to no value that could not be addressed through the use of reference set attributes and appropriate operations and methods in the existing classes and objects. Moreover, the header classes and entity objects complicated one’s understanding and analysis of the sequence diagrams and design class diagram, destroying any value they may have brought to the table. Thus, the team made the decision to remove the header classes and entity objects from the sequence diagrams and design class diagram, and to instead insert appropriate reference attributes and operations where needed to compensate for their functionality.

The team chose the selected use cases (Update Emergency Info on Server, View Emergency Medical Info, Read Records from Device) to represent in the sequence diagrams because they embodied activities that could be repeated often by an actor in a single session; therefore, they had to be captured and designed accurately in order to avoid any oversights that would cause problems with the system, whether internally or externally facing. Each sequence diagram and accompanying system sequence diagram contain a loop that captures the need for a repeated action, whether it be reading multiple records from the device, updating multiple records, or viewing multiple records in an emergency environment. By creating interaction diagrams of these use cases, the team was able to identify and
correct any interaction and functionality problems which assisted in further analysis and design work on the system.
Physical Design

ERD
Validation & Discussion

Entity Relationship Diagram
The entity relationship diagram was reversed engineered from our class diagram. It was intended to be a more easily understood representation of our system – targeted at non-technical individuals. To do this we made several concessions:
• Patients and Medical Practitioners are represented as entirely different entities, rather than subclasses of a common class. From a user point of view these two types of users have little in common.
• All emergency medical records are represented together, with different records displayed as attributes of the Emergency Medical Records entity. This greatly simplifies the diagram.
• The access log is shown as an attribute of the relationship between Patient Medical Records and Medical Practitioners. We felt this was a more intuitive representation than making it its own entity.

The biggest difference between our Entity Relationship Diagram and our Class Diagram is the inclusion of the Patient Medical Records entity in the ERD. In reality, our system does not store medical records – they are stored on a device controlled by the patient. However, showing the records as part of the ERD makes it easier to communicate the system to end users.

We believe that this Entity Relationship Diagram would be a powerful tool for explaining how the system is structured to the uninitiated. If we had created it before the class diagram, it also would have been a useful starting point for creating that document. Indeed, the ERD closely resembles early iterations of our class diagram.

Physical Database Diagram
Our physical database diagram is almost structurally identical to our class diagram. For the most part we just took each of our classes and made it a table, adding attributes as necessary. This was possible because we do not have any many-to-many relationships in our class diagram. This precludes the need for intermediate tables. We also do not have any classes that don’t require data storage (temporary classes, etc).

The only structural change required is to create tables for the super- and sub-classes. This means that we created a table for each super class (Account and Patient Entered Emergency Medical Record) and a table for each of the sub classes (Patient Account, Provider Account, and each emergency medical record classes). To do this we created the same primary key for both the super- and sub-classes. The appropriate record type can be determined by performing a left-join between the super- and sub-class tables.

Attributes were fairly easy to determine. Most are pulled directly from the class diagram. A few attributes in the class diagram are composite attributes and needed to be broken down into multiple database columns (ex: Address becomes Address1, Address2, City, State, Zip). We also added primary and foreign keys to the tables. Association attributes helped guide the foreign key creation.

We did have some difficulty keeping track of changes – changes in the class diagram necessitate changes to this physical database diagram, and vice-versa. Using an integrated program like Rational Rose would likely have mitigated this issue.
Evaluation of Analysis & Design
The idea of electronic health care records has gotten a lot of press in recent years. With President Obama's initiative to get a modern electronic health record system, it makes sense that the patients should have the ultimate control over their records. Because of all the opportunity, our project domain is centered on this concept.

However this is not an easy situation. After all, if it was a simple problem, it would have been solved by now and not have a five year timeline issued by the President. There is an inordinate amount of complexity involved including the issues of privacy, security, reliability, ease of use, complexity, and more. Because of this, our project had a narrow scope to encompass the most important issues involved with a personal health care record system.

UML served well to help to limit the scope of the problem. Using Use Case Diagrams the team was able to find what scenarios are most important to each group of users and concentrate on those through the development process. Because this was the first diagram completed, and because scope was so crucial, the team debated greatly here. To do this the team utilized group collaboration tools. Microsoft LiveMeeting and Microsoft SharedView were used to share screens, notes, and workplaces to allow for collaboration. Skype was also used to verbal and written communications. Email also proved useful for updating the group when not immediate response was needed.

The Use Case diagram was done using Microsoft Visio. This tool was selected because the team was already used to it through other professional and academic work. With Visio 2007 and 2010, the UML diagraming capabilities have become quite good. This, when combined with the capability to track revision history and add notes, allowed for communication and collaboration both synchronously with LiveMeeting and SharedView and asynchronously through email. Microsoft Word was used in a similar fashion to do all of the analysis and documentation.

The team found the beginning of the project much more difficult and consuming than the rest. This is likely caused by becoming more accustomed to UML as well as the creation and adoption of methodologies and workflows. These workflows include the creation of a road map to project completion, a timeline for deliverables for the team, splitting work to be done individually, and virtual meetings on a regular basis to assemble parts and check in. Once this formula was created, the project really came to life. This approach was taken for the rest of the UML diagrams that were completed.

Many of the UML techniques that were described to aide in development (like TCM) were a great basis for the project. It allows for an easy way to get started on aspects of the system. It became the building blocks that the other diagrams and processes relied on.

Despite the success in working together, some things would have been done if there was more time. The biggest issue that was minimized was the lack of external reference. While the team had done some research on the problem domain, some additional work could have been completed. This might include interviews with health care professionals and patients to see what they would like in a personal health record system. More research might have also been done on proprietary systems in place currently.
While the system the team proposed interacts with proprietary systems, due to a lack of time our system interacts via an API. Basically, the proprietary systems are a black-box. If the team had more time, more of this could be straightened out.

Working for the project in a team was really beneficial to all of us as we learned a lot by practice and from each other. In the end, we hope that we cleared all major issues that the system could possibly face and provide alternative solutions wherever necessary.
Summary and Lessons learned
As the above diagrams and discussion details, the Personal Health Record System is a complex system. The number of associations needed is significantly greater than we had originally anticipated. This is especially well illustrated by our full sequence diagrams, in which we navigate through a number of entities before getting to the information we want.

Although the complexity of the system as a whole was greater than anticipated, we did manage to limit the scope by not defining the actual storage format of patient information. This data is overwhelming complex, and creating a structure would require far more subject domain expertise than we have. Simply avoiding the issue and referencing a standard format gave us the ability to focus on the system itself.

Indeed, the only medical information we are storing is basic emergency medical information. This information needed to be accessible at all times, which meant we couldn’t rely on keeping this data on the personal storage device. Modeling this information proved complicated. In particular, we had some difficulty determining whether we needed header or collection classes for each type of emergency medical information. These classes would have enforced some kind of structure and process over creating these records, but in the end it was determined that it would unnecessarily complicate the retrieval of the records themselves.

Over all, this project was both a lesson in UML – particularly the ramifications of heavily associated data – and a lesson in group work. The group work portion was probably the most useful. We learned to collaborate remotely using a number of tools and to manage a large project over time.

John’s Experience
While my experience on this project was a mostly positive one, there were a few problems and instances that represented an area of improvement for any similar follow-up projects or tasks. First, I found that working collaboratively on the diagrams and components of the project was much easier to do in person than over the Internet using Skype, Microsoft Live Meeting, or other similar collaborative software. While the latter way of working with teammates is definitely more convenient, the former method allows for more frequent informal discussion as well as an easier time in communicating exactly what you have in mind with regards to a particular diagram or model. For these reasons, I would like to attempt to have the majority of the group meetings in-person for the next project of this nature that I take part in.

Additionally, because this particular project had so many parts that depended on each other, I found that more planning is required than other projects before you even sit down to work on a single diagram, model, or paragraph. Our group eventually decided to construct and follow a project roadmap, which laid out what group members should be doing what tasks at what point of the project. This plan ensured that the group would always have the pre-requisites necessary to move forward onto the next phase when the time came, whether it be a diagram or area of discussion. One part of the project where the roadmap was particularly helpful was that of the sequence diagrams and design class diagram; since the two are very much intertwined, the roadmap gave the group a deadline for completing their
sequence diagrams. This situation allowed an appropriate amount of time to work on the design class diagram whilst also guaranteeing that the group would be able to consult the sequence diagrams during the design class diagram development process. Without such planning, the group would very likely have been less organized throughout the development of the project.

Another lesson I learned while working on this project was the need for constant and immediate documentation of everything discussed or thought of. A few times while planning the sequence and class diagrams, I would come up with an idea to incorporate, even when I was not working on the project at the time. Early on in the project, I would often fail to write down this idea, instead trusting my memory to provide that particular detail at the later date and time when I would need it. Unfortunately, this was not such a great idea, as I would often forget what I wanted to include or how I wanted to include it. Thus, I began documenting every little idea and suggestion I had for the project; while many did not find themselves into the final product, the group that did certainly made it better in the end.

Building on the previous statement, the last lesson I learned while working on this project was that you don’t have to be formally in “working on project” mode to productively think about the various diagrams, models, and other components of the project. Often, I would find that some of my best ideas about a particular component or problem would come during unusual times, like driving in the car or sitting in another class. While some projects for other classes can be completely done in exclusive, mandated “project time” windows, I soon found out that this project required much more natural yet persistent thinking and deliberation in order to overcome all the included obstacles.

**Sam’s Experience**

My experience on this project was a positive one, both from formal education and group-work perspectives. The project was a great education about UML. In the undergraduate version of this course I learned the basics of the content, but this class gave me a deeper understanding of the processes and artifacts. In particular, I learned more about the associations between classes and how to show them both in class diagrams and in full sequence diagrams.

I also found the group work to be an enlightening experience. During the beginning of this project I had the good fortune to meet with my group members in person. Towards the end, however, we were not in the same area. I have never had to complete a project of this scale with individuals who are geographically dispersed and found it to be a significant challenge. Fortunately I had access to a Microsoft Live Meeting account and we were able to use this tool to collaborate remotely. Without this tool I do not think this project would have been possible.

Another difficulty was keeping track of versions of documents. As we progressed through the design process we frequently had to return to earlier diagrams and modify them to reflect new realizations. This was especially difficult because each team member was working on their own personal diagrams (like sequence diagrams) which each required modifying a number of shared earlier documents (like the class diagrams). Ensuring that each member understood the changes made to these diagrams proved to be a significant issue.
In the end we adopted a number of standard project management techniques. During the last week of the project we had almost daily project status meetings using Skype, a Voice over IP application. During some of these status meetings we used collaborative tools to share our work. We also adopted a project manager (John Misczak) who kept track of tasks and timelines and let us know what needed to be done.

In the end, I believe that this is the one project at the iSchool that most resembles a real-world design process.

**Chris' Experience**

In doing this project I learned some important concepts. Methodologies like TCM made it possible to easily do things that previously took much more time and effort. I was able to identify important concepts, classes, and potential problems much earlier in development. This is quite useful as I could better plan and be more prepared when the time comes to begin writing code.

These analysis techniques not only help at the beginning and planning stages. Instead I certainly see how certain diagrams have their part throughout the development life cycle. Most importantly, I see where many diagrams are useful years after the system is completed. UML diagrams serve to almost document how things were accomplished and allow a new developer to get into the mindset of the original developers when they designed the system. The power of UML is certainly apparent here as this self-documenting code could save countless hours in modifying and maintaining the system in the future. Because maintenance is one of the highest costs of software, this is indispensable. Also because the main cost in the development cycle branches from complexity and poor planning, the UML diagrams will prove useful here as well.

The team work also proved beneficial. We utilized some tools and techniques to get this job accomplished that I have never used before or not in this context. Email served great as an asynchronous communications tool. Skype was great when we needed to look at things together and communicate orally. Microsoft Live Meeting and SharedView helped us collaborate on documents almost as if we were at the same location and computer. Lastly, the revision tracking tools in Microsoft Office made it easy to see how our work changed and progressed. The tools, techniques, and experience in working with a group (and working on individual aspects within the team) will prove indispensable in the future.
Bibliography


Appendix

Division of the work among team members

All team members contributed equally on this project. For the chosen primary use cases, each member was responsible for constructing his use case’s detailed use case description, activity diagram system sequence diagram, and sequence diagram. All team members reviewed each other’s diagrams and provided feedback when necessary. The larger diagrams (use case diagram, class diagram, design class diagram) were done in tandem with all members present, whether it be in person or over Microsoft Live Meeting and/or Skype. Additionally, all team members contributed to the discussion sections in equal amounts.