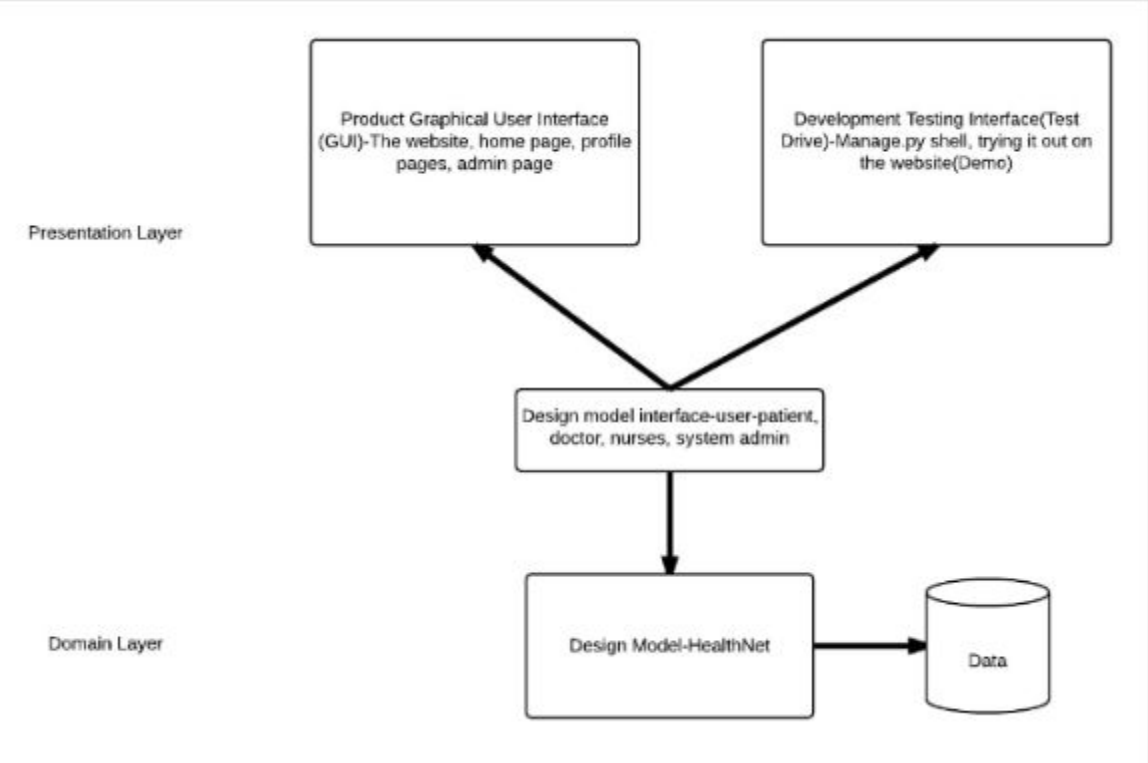
Product Design

Team 2BDetermined

Revision Number	*Revision Date*	*Summary of Changes*	*Author(s)*
0.1	09/19/2015	Created the basic design information, UML diagrams and made first revision.	Daniel Dang, Colin Fausnaught, Raasin Siddiq, William Tarr, Barry Wu
0.2	9/25/15	Fixed Design Rationale	Raasin Siddiq
0.3	10/03/15	Added the architectural model for the document to explain what the project is going to do in out point of view	Barry Wu

Architectural Model



This diagram represents the major subsystems of the product. Initially focus on the domain layer and its components before decomposing the user interface component. Note that a common interface allows both the GUI and a Command Line Interface to access the domain model in the same manner without regard to the type of presentation technique.

Components and Functions

Nouns: **Patient**, **Nurse**, **Doctor**, **System Admin**, **Appointment**, User, Test Results, Prescriptions, Medical Information, Personal Information, Message, Reminders, Notifications
Verbs: Sign up, Provide personal info, verify info, Log in, upload, reminder of appointment (user defined)?, discharge,

Appointments	Component state ● Duration of different meetings ● Boolean of whether an appointment is complete or not ● Store information of the appointment ● Handle appointment creation, deletion, and modification Component behavior ● Integrate with Calendar to plot appointments ● Used by **all users** to handle appointment creation, deletion, and modification

	● Keep certain appointments private and some shared (nurse/doctor relationship)
Logging System Activity	Component state ● Log all changes in the system ● Save the log to be reviewed later Component behavior ● All systems and components in the project will be logged by this component. ● Logs will be available to be viewed by certain users (such as System Admins)
Profiles	Component state ● Handle creation and registration of new users ● Store information for each user ● Handle different types of accounts (admins, doctors, nurses, patients) ● Handle admission/discharge from hospitals (only doctors can discharge, only nurses and doctors can admit) Component behavior ● Different account types will have different access to different profile components (i.e., basic users can't access admin panels) ● This is one of the most essential parts of the program, as it handles the storage and modification of all users' stored information.
Test Results	Component state ● Assigned to a patient ● visibility of a test result is changeable ● stores comments Component behavior ● Viewed by users on a case-by-case basis (Doctors can hide certain results from patients)

Class Diagram(s)

(attached)

Sequence Diagram(s)

(attached)

Design Rationale

9/20 - As of now, we are committed to the promises we have made in the requirements document in R1. We added test results to the components even that is slated to be in R2 but that is because we have discussed it in detail thus far. We have also discussed a messaging/notification system. The main two components we are tackling in R1 are: registering the users and specifying what type of account they are creating so that it is clear what permissions they have,

and navigating a scheduling system that will create, reschedule, and cancel appointments. These are the core elements of HealthNet. We also promised a basic logging system that the system itself will handle, a way to manage hospitals and rooms, and a way to delete or demote administrators.

10/25 - We added the following classes to our class diagram: Prescription, Message, Surgery, Test Results, Hospital. These classes are all relatively self-explanatory in function. For the simple hospital class we wanted to have it treated as an object rather than a string field because we want to be restrictive in how many hospitals there and which hospitals are in the system. We only want patients to register to hospitals that are already in the system, as HealthNet is to only have specific hospitals within its system. For the rest of the added classes we wanted to create them to more easily store data than mass-storing that data within fields but also have the classes be small and independant enough to not be burdensome. These classes are also designed to be easy to filter (i.e. you can filter a message by sender/recipient to fill in a user's inbox). We associated surgeries with appointments so we could keep track of appointments that were surgeries while still allowing for the input of surgeries into the medical history that do not have appointments (i.e. past surgeries that were not around when HealthNet was/ surgeries performed in hospitals outside of the HealthNet system). We associated Test Results, Surgeries, and Prescriptions with patients so that we could grab them when viewing patient medical info when given the patient. We updated the patient class to hold more info (height, weight, ethnicity, etc.) as per Dr.House's request. We added the following use cases to our use case diagram: use case #22 - CSV Parsing. We wanted to go through and show how we would handle an admin user attempting to populate the system with data from a .csv file as Dr.House requested that we do.