Medical Monitoring Device

Advisor: Dr. Orlando Hernandez

Senior Project I: Spring Report

May 2013

Steven Zilberberg

Dedication

This project was inspired by and dedicated to the life of Morris Brofsky. In November 2011, Morris, living alone in his apartment in New York, experienced a stroke leaving him unconscious on his apartment floor for three days before being discovered. A month later, December 2011, after a month of being in a comatose state, Morris passed away. If a medical device like what this project is about, it is possible that Morris could have survived his stroke attack.

Acknowledgements

I would like to thank Dr. Hernandez and Dr. BuSha for his advisement on this project and for answering questions and giving advice on different aspects of the process. I look forward to continue using them for support in the coming semester to conclude this project

Abstract

As people age, sometimes it is hard for them to maintain a normal lifestyle without assistance or there is a potential that they can be put into a life or death situation and they might not be able to call for help. For twenty years, there have been devices available to help solve this type of issue. However, due to their design, their uses are simplistic and often restrictive, which limits and even sometimes negates the function of the device. This project will introduce a new design that will revamp the ideas of these older devices by implementing new technologies such as GPS and microcomputers to increase the freedom of the user and enhance their protection. Using these new advances, a better medical monitoring device can be designed.

Table of Contents

1.0 - Team Management

Steven Zilberberg is a Computer Engineering Major and he is the only person on this project. He will endure all responsibilities from taking this project from planning, to construction and testing of all phases. This also includes updating and maintaining the documentation and website for the project progress.

2.0 - Introduction

Aging is a natural part of life and is inevitable for us all. As a person grows older and enters their senior years, age 65 years old and older, there exists the potential to put stress on the person's family. As people age, some require extra help and different types of assistance in order to maintain a normal lifestyle. This can vary but is not limited to living with their children or being put into a senior home. Because this might be undesirable, if a person can remain independent and is capable of taking care for themselves, they may choose to remain to live alone. However this can lead to many different types of problems.

1.1 - Definition

As someone ages, regardless if they can take care of themselves or not, the risk of an unexpected problem arising, natural or accidental, is real and increases everyday. As a person grows older, they can be more prone to heart attacks, strokes, blood clots, falls, broken bones, and various of other conditions. Citizens who live with their family have a good chance for those who they live with to assist them in case of an emergency. If something happens to someone who lives alone, the chance of them receiving help and even surviving the experience is not as likely.

There is currently a few devices on the market which addresses this issue but the pitfalls of these device is that the person needs to be conscious and coherent in order to activate the system. If someone becomes unconscious, this system is nullified and is no longer effective. Several devices have come into existence throughout the years and it will be discussed that all of these devices are fundamentally same because they are all inspired from a single root device with very similar functions.

1.2 - Need Identification

Senior citizens who live alone are more prone to accidents as they grow older and sometimes these accidents may only be survivable if immediate assistance is given to the individual. The current market device closes the gap partially because it addresses some different issues which are only relevant if the patron in need is conscious, otherwise, the system is nullified.

There is a need for a system to identify if patron who requires assistance regardless of the user's state of consciousness. This system can use the essence of the current commercial devices but also extend on its functionality by adding certain technologies such as GPS location and log files for medical purposes.

3.0 - Background

The traditional method of monitoring a sick or very elderly person to assure their safety is to constantly watch them by physically being in the local vicinity. This has been true for hundreds or even thousands of years. If a family had an elderly grandmother, besides sending her to an assisted living or a community home, it would be common for her to live with the family if she was unable to fully live by herself, safely. Commonly someone would always be at home such as a stay at home mom or the elderly person would not be left alone for very long.

As technology sprung forward and computers and communications systems started to become mainstream in the past thirty years, commercialized systems have come about to help give an alternative for the simple, traditional methods of monitoring the sick or elderly, which the responsibility of looking after the elderly would increase stress on the family. When first introduced, these systems were very good for what they were built for and was a large step into a helpful direction. However, many years have past and those systems, although still around are dated and inefficient by today's technological expectations.

3.1 - Current Commercial Devices

The purpose for this project is to redesign and enhance the current commercial systems on the market by replacing the leading products with a better alternative. For the remainder of this report, the current commercial system will be referred to as CCS.

The CCS is designed for senior citizens who live alone or spend a lot of time by themselves and have the potential of falling and not being able to pick themselves up or have the capacity of injuring themselves in the event of a fall. In the event of an emergency, the user activates the CCS device by the push of a button. The device is either worn as a pendent around their neck or as a bracelet on their wrist. Once activated the device communicates with a base station in the house of the user. This device communicates with the CCS's company, via landline communications, who will call the home. If nobody responds, emergency medical services (EMS) are sent to the house for assistance.

3.2 - Pitfalls of the Current Commercial System

The CCS is a very good device for senior citizens and as their company's testimonies suggest, they have helped save many lives through their system. However, this system is very dated and has not progressed or improved its functionality with time. From the time when this device has come into fruition, technology has advanced and the functional needs of this device has increased beyond the device's scope. There are many pitfalls to the device, and the largest drawbacks are explained in this section, below.

The number one restriction of the CCS is that the person needs to be conscious and relatively coherent in order to use the device. If the user is unconscious, then the system is no longer beneficial and potentially incapable of of saving someone from a life or death situation.

The CCS also has the setback of restricting the use of the device to the residence of user because it necessitates communication with a base station. Even in some cases, if the house is big enough, the user could be in a section of their home where they may not be in range of their base station which is another scenario where the CCS would be rendered useless.

The CCS device is simplistic where it doesn't take advantage of newer technologies such as microprocessing programming paired with sensors which can be used to determine if a user is in need of assistance when they aren't conscious or unable to activate their device. Taking advantage of this technology can reduce costs to the user and increase their range of freedom while getting enhanced protection.

4.0 - Design Concept

There are two main objectives for this project that forms the basis of the design of the medical monitoring device. The first and for most is functionality and it is the prime objective for this project. One reason why the original CCS devices were adequate for their time is that they worked as designed. The primary goal for this project is to get the core functionality of the device working before anything else.

The second concept is comfort and wearability. This, although not as major as the first is still crucial because if the device isn't desirable to wear, then a patron who uses the device might not wear it which will overall defeat the device. This concept is important for if this device was to be produced but since the project is mostly for functionality, comfort isn't the prime concern.

4.1 - Proposed Redesign

The medical monitoring device would ideally be a redesign of the current CCS with several enhancements in order to expand the functionality and to increase the freedom of the user. Since the CCS came to market, there have been a few successful remakes of the CCS that have added some functions to the device and expanded on the range to the base station. Although this is good for the consumer, they have branched from the same functions of the CCS and some still have the same flaws outlined above because their design path has been derived from the same root product.

This design would completely change how the CCS device operates by changing several key functions which has held back the device and other companies' redesigns. My device would operate around a small microcontroller to monitor a person through various sensors and to process the sensor's information automatically and in real time. During the data processing, if the information falls outside the bounds of normal data (such as a high/erratic heart rate which can be a sign of a heart attack), the device’s secondary functions would activate. These secondary functions would try to determine the person’s condition while acquiring their location through an onboard GSP unit. The microcontroller would also assemble a message using onboard storage information to prepare to send a report with the person’s location, current medical situation and other personal information. If a need to send the message is determined, the device executes its tertiary functions by sending out the message through its on board GSM module to predefined phone numbers possibly including emergency medical services. The advantage of this design would enable a user to be protected by a device that monitors them automatically while having a greater degree of freedom than being confined to their home for protection.

Similar to the CCS devices, this redesigned device would be made to be small, light, compact as well as easy to use. One main reason why the CCS devices are used is because it is easy to carry. One of the sensors of the system would ideal by a heart beat sensor, so an ideal, noninvasive position for this device would be on the wrist. Therefore the design will try to follow the model of a standard wristwatch. Using this and trying to keep weight down to a minimum would ensure ease and comfort of using the device.

5.0 - Specifications

The device specifications are based on trying to build a reliable and compact device that is similar to the size and weight of the CCS devices. The functions of the device can be split up into different sections; Onboard processing, GSM communication, and device sensors. Each of these parts are very important to the success of the project.

Onboard processing is the software that interacts with the hardware as well as communication between the different hardware such as the on board SD card for saving information, sending information to and from the GSM module, and handling data between the different sensors including the GPS unit. Assuming an efficient process is implanted in the final design, onboard processing of information should be relatively quick once information is transferred into the microprocessor. Below are constraint goals of the onboard processor.

Onboard Specifications	Constraint Goal
Onboard Processing time	<1 sec
SD Card Read/Write	~ 1 sec
Weight	<= 2 LBS
Size	4" x 6"

Table 5.1: Onboard Specification Constraint Goals

The onboard sensors include a heart rate sensor, an accelerometer, and a GPS module. Everything should ideally be able to collect and transfer data to the onboard microprocessor in about one second in order to reduce latency from the system and provide up to date information.

The GPS unit will have the exception of having a 5 second constraint of locating an accurate position of a person. In order to reduce the bottle neck in processing time that this will cause, the architecture of the system will be modeled as an interrupt based system. This will allow for reduced overhead which would result if the system instead using a polling strategy. The assumed latency is due to the potential of the signal not being the most ideal since the person using the device would likely be inside a building during device's usage. Below are constraint goals of the device's sensors.

Sensor Specifications	Constraint Goal
Heart Sensor response time	~ 5 sec
Accelerometer response time	~ 3 sec
GPS response time	~ 5 sec
Sending information to MCU	<1 sec
Average sensor weight	~ 0.25 LBS

Table 5.2: Sensor Specification Constraint Goals

The GSM module is the last part of the system and is the data path the the device takes in order to get the information out to an emergency contact. The GSM module should be able to communicate and transfer it's data in approximately 3 seconds. The cause this latency is normal to the fact it is communicating with cell phone towers in order to transfer data.

Sensor Specifications	Constraint Goal
GSM Module response time	~ 3 sec
GSM Module size	Less than 4" x 6"
GSM Module weight	< 1LBS

Table 5.3: GSM Module Specification Constraint Goals

Another aspect of specifications that needs to be taken into consideration is the size and overall weight of the device. Ideally the design goal is to get the product down to the size of a large watch. This will ensure that someone is able to wear and use the device daily without having to take it off. Due to the watch design, a need for comfort is also going to be taken into consideration in order for it to be wearable.

Sensor Specifications	**Constraint Goal**
Device Size	<= 4” x 6”
Device Weight	4-5 LBS
Device Style	Wrist Watch

Table 5.4: Device Specification Constraint Goals

6.0 - Hardware

The key difference between this project and the CCS devices would be the hardware of the device. The CCS devices are very dated and what this project will do is take advantage of current technology in order to enhance the device's potential and increase the user's freedom

6.1 - Hardware Concept Analysis

This redesign has a big advantage of being brought into realization twenty years after the CCS device. This allows the device to take advantage of current technologies that were either not available or too large or expensive in the past. The main devices that would be needed for this redesigned project are:

- Heart Rate Sensor
- Global System Communication System (GSM) Module
- Global Positioning System (GPS)
- Motion Accelerometer
- Secure Digital (SD) Storage Device
- Small Microcontroller

By combining these devices and driving them through a microcontroller, my design becomes possible, given the proper software to drive them. This software would be needed to be programmed from scratch and is explained abstractly in the next section.

The two main sensors are the heart rate sensor and the motion accelerometer. In conjunction, these can be used to determine the current heart rate of a person or if that person has experienced a fall. Processing the data from these devices on the microcontroller, it is possible determine if a person needs assistance. This type of functionality would required processing with a microcontroller which the logic for this heavily relies on the software implementation. Through this software, additional functions can be created such as learning the habitual movements of a person.

The use of the GSM and GPS modules is one of the things that sets this device apart from its predecessors. Using this combination, the GPS will track a person's exact location (usually accurate within 10 meters) and then sending their coordinates through GSM. Because the device would be communication through the means of an everyday cellular device instead of a base station, this means means they are almost always connected. Instead of restricting a person's protection to their home by using a base station, it gives the freedom for a person to move from their home while still being monitored, connected, and protected.

Below is a generalized diagram for how the hardware will interface with the microprocessor. Note that each of the devices may not necessarily interface directly with the microcontroller. There is some need in some cases, such as the GPS unit, to communicate with an exterior device first to make sense of the data passing back and forth with the microprocessor.

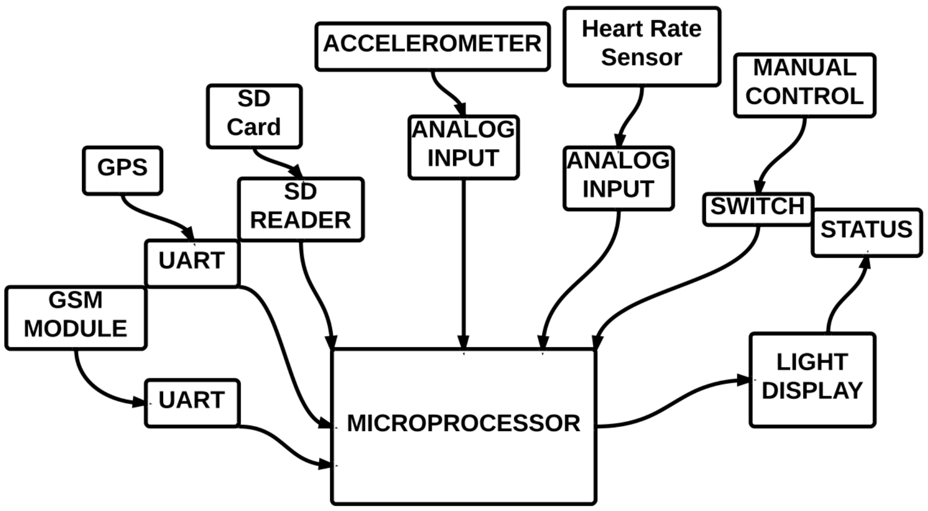


Figure 6.1: Hardware Abstraction

7.0 - Software

To interact with the hardware of this device, custom software will be written for the microcontroller. The C Programming language will be used to interface the hardware together and communicate with each peripheral. Using custom written software, it will become easy to make logical decisions of when and how to utilizes the different pieces of hardware of the device as well as to do basic data analytics.

7.1 - Software Analysis

The software aspect can be separated into three main parts; the heart rate driver, accelerometer driver, and the emergency action driver. Each of these pieces of software will be used for controlling the main functionalities of the device.

The heart rate monitor section will control how the heart rate sensor will act and it will also implement the logic to try to determine if a person is having abnormal heart activity. If the code detects abnormal data from the hardware, it will try to collect a few samples to determine if the person is having a legitimate emergency to flag the main program.

The second part, the accelerometer software, will control the logic for the motion of the device. The main purpose will be to see if the device detects a fall but will also have a purpose to see if the person if not active at a time they are normally moving around. This would be useful as a redundancy check.

The last part of code will belong to the main part of the software which will incorporate the logic of the emergency action functionality. While getting data by the resulting logic of the first two parts, this section of the code will determine if action needs to be taken. If the system needs to be activated, this code will drive the logic of the other systems to gather the needed data, organize it, and then send the alert out to the required parties.

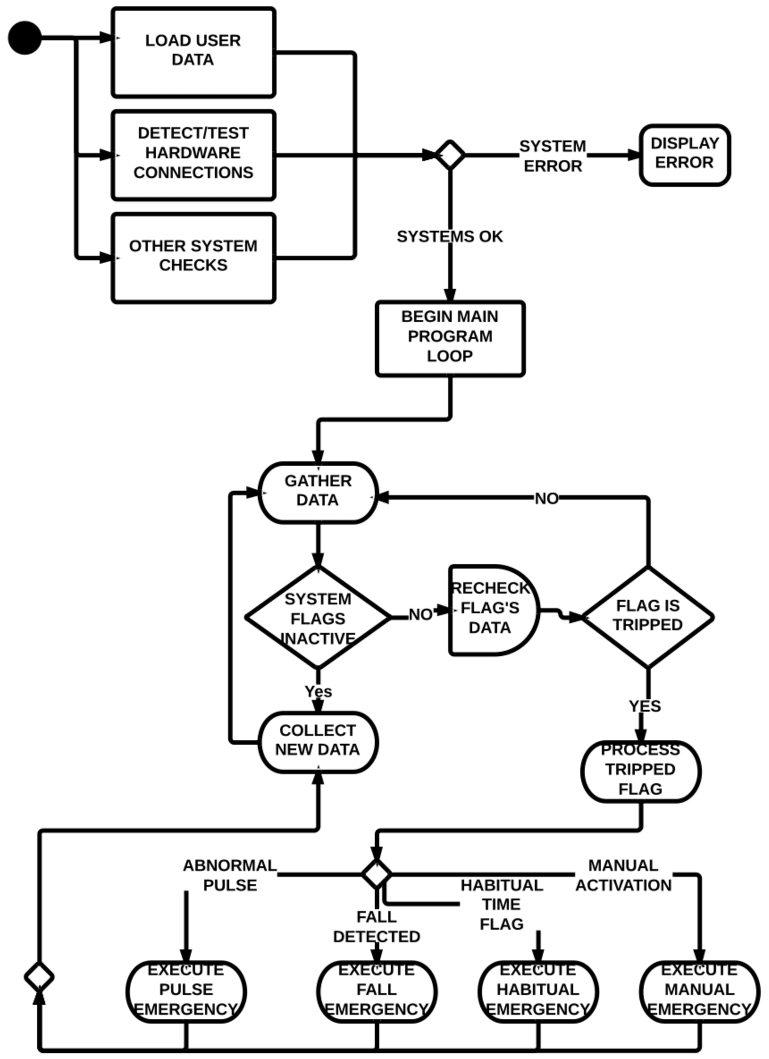


Figure 7.1: Software Abstraction

7.2 - Simulation Program

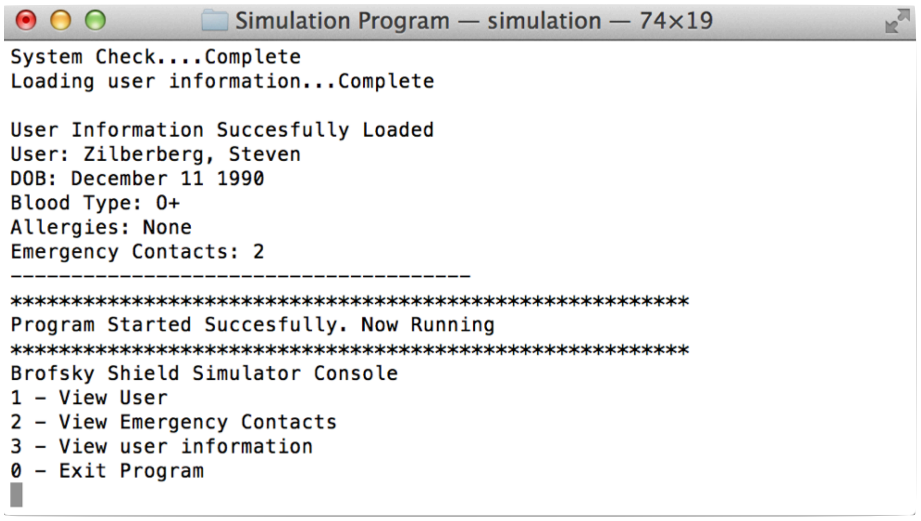
To help guide the planning phase and understand the logic of the device's software, a simulation program was constructed in the C++ programming language to encapsulate the core logic and functionality of what the device would do. The simulation is comprised of three different main files.

The main program, codeSimulation.cpp, which holds the main function of the program. This theoretically would simulate checking the hardware of the system, gets the needed reference files, loads the user's information, simulates a read of the accelerometer and heart rate sensors, and simulates a detection and action of bad data. For diagnostics as well as futher device simulation, the program utilizes multithreading to check data and also interact with the user a simple prompt for displaying information.

A class used by the program is a simple Contact class, Contact.h/Contact.cpp, which holds the information for the emergency contacts for the user. This is more of a class for organization in order to easily organize the contact information.

The second class used is PersonProfile, PersonProfile.h/PersonProfile.cpp, which is a large class which works with the main program. This class holds the information about the User and his/her contact information. It also provides basic functions of list the emergency contacts and changing the data of the user if needed. This will be useful later when the simulation will be enhanced to help further with planning.

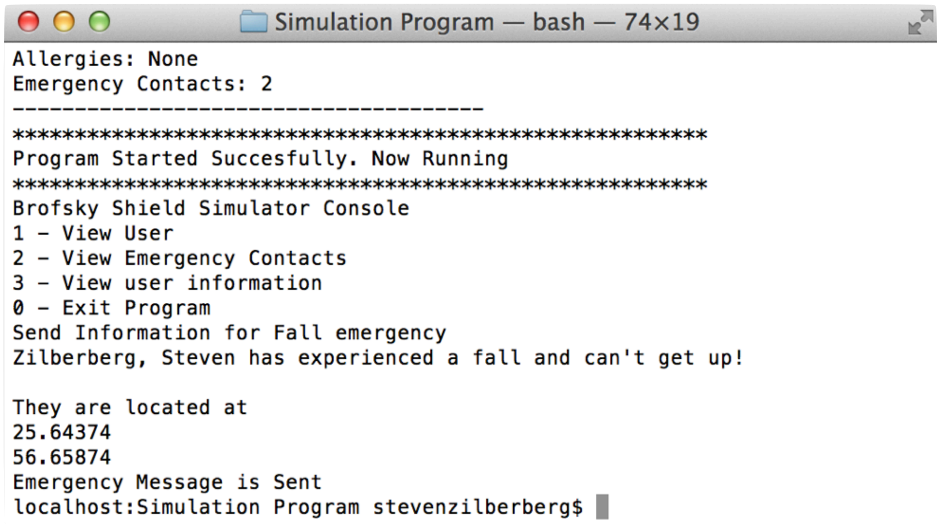
As mentioned, the purpose of this simulation is to help with the planning and analysis of the logic required for when the device is running. The current plan is to use this simulation and convert the program into C Programming which is what the software of the device will be written in. Since C++ is a superset of C Programming, the conversion should be simple to to do.



```
System Check....Complete
Loading user information...Complete

User Information Succesfully Loaded
User: Zilberberg, Steven
DOB: December 11 1990
Blood Type: O+
Allergies: None
Emergency Contacts: 2
--------------------------------------
********************************************************
Program Started Succesfully. Now Running
********************************************************
Brofsky Shield Simulator Console
1 - View User
2 - View Emergency Contacts
3 - View user information
0 - Exit Program
```

Figure 7.1: Screen shot of simulator



```
Allergies: None
Emergency Contacts: 2
--------------------------------------
********************************************************
Program Started Succesfully. Now Running
********************************************************
Brofsky Shield Simulator Console
1 - View User
2 - View Emergency Contacts
3 - View user information
0 - Exit Program
Send Information for Fall emergency
Zilberberg, Steven has experienced a fall and can't get up!

They are located at
25.64374
56.65874
Emergency Message is Sent
localhost:Simulation Program stevenzilberberg$
```

Figure 7.2: Screen shot of a simulated emergency

8.0 - Conclusion

The medical monitoring device is designed to to better the lives of those who are elderly or sick. The primary target user is someone who is elderly who might not have a very active lifestyle and live either alone or spend large amounts of time away from others. There are products on the market that have existed that have helped save lives of people but due to advancements of technology and how people live today, those devices are now dated and my project is something new and is a device that could fill the technological gap. This project by taking advantage of technologies that are available today that wouldn't have been able to be used ten or twenty years ago.

Currently, a simulation project has been constructed and is continuing to be refined in order to understand the flow and logic of the overall system. The simulation program shows the proof of concept of the logic and after being perfected, will be translated into C Programming which is what will be used in the software of the device. The program itself is simple and the challenge will be integrating it with the hardware that has yet to be acquired. Once acquired, the challenge will be getting everything to work accurately and quickly. Initially, testing will take place before any integration is done in order to understand the usage of each of the devices.

Over the summer break, in the early weeks, a list will be confirmed of what is needed for the project and the hardware will be ordered. During the waiting period, documentation will be updated as well as the online documentation such as the website for the project. The goal for by the end of the summer is to have a rough prototype which will hopefully be refined in order to get a better quality product during the spring semester.

9.0 - Reference

1) Luštrek, Mitja, Hristijan Gjoreski, Simon Kozina, Božidara Cvetkoviü, Violeta Mirchevska, and Matjaž Gams. *Detecting Falls with Location Sensors and Accelerometers*. Tech. N.p.: n.p., n.d. Print.

2) "Cplusplus.com - The C Resources Network." *Cplusplus.com - The C Resources Network*. N.p., n.d. Web. 29 Apr. 2013.

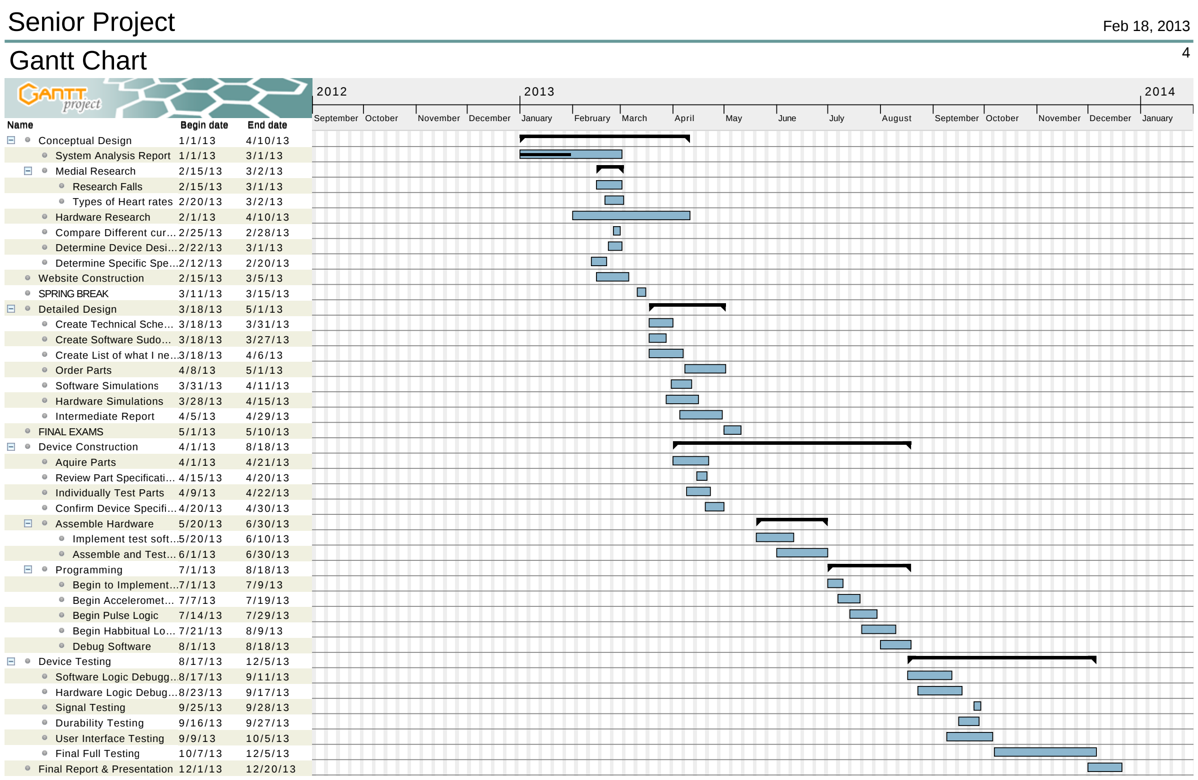
3) "Microchip Technology" *Microchip Technology Inc*. N.p., n.d. Web. 29 Apr. 2013.

4) "SparkFun Electronics" *SparkFun Electronics*. N.p., n.d. Web. 29 Apr. 2013.

Appendix A: Biographies

Steven Zilberberg is a Senior student at the College of New Jersey studying Computer Engineering. For the past four consecutive years, he has been a tutor in the college's Tutoring Center for all Computer Science levels. In addition he has interned for two consecutive summer, with this coming summer being the third, as a software engineer for Teletronics Technology Corporation in Newtown, PA. He has a very strong programming background from his outside experiences.

Appendix B: Gantt Chart



Senior Project

Feb 18, 2013

Gantt Chart

4

Name	Begin date	End date
Conceptual Design	1/1/13	4/10/13
System Analysis Report	1/1/13	3/1/13
Medial Research	2/15/13	3/2/13
Research Falls	2/15/13	3/1/13
Types of Heart rates	2/20/13	3/2/13
Hardware Research	2/1/13	4/10/13
Compare Different cur…	2/25/13	2/28/13
Determine Device Desi…	2/22/13	3/1/13
Determine Specific Spe…	2/12/13	2/20/13
Website Construction	2/15/13	3/5/13
SPRING BREAK	3/11/13	3/15/13
Detailed Design	3/18/13	5/1/13
Create Technical Sche…	3/18/13	3/31/13
Create Software Sudo…	3/18/13	3/27/13
Create List of what I ne…	3/18/13	4/6/13
Order Parts	4/8/13	5/1/13
Software Simulations	3/31/13	4/11/13
Hardware Simulations	3/28/13	4/15/13
Intermediate Report	4/5/13	4/29/13
FINAL EXAMS	5/1/13	5/10/13
Device Construction	4/1/13	8/18/13
Aquire Parts	4/1/13	4/21/13
Review Part Specificati…	4/15/13	4/20/13
Individually Test Parts	4/9/13	4/22/13
Confirm Device Specifi…	4/20/13	4/30/13
Assemble Hardware	5/20/13	6/30/13
Implement test soft…	5/20/13	6/10/13
Assemble and Test…	6/1/13	6/30/13
Programming	7/1/13	8/18/13
Begin to Implement…	7/1/13	7/9/13
Begin Acceleromet…	7/7/13	7/19/13
Begin Pulse Logic	7/14/13	7/29/13
Begin Habbitual Lo…	7/21/13	8/9/13
Debug Software	8/1/13	8/18/13
Device Testing	8/17/13	12/5/13
Software Logic Debugg…	8/17/13	9/11/13
Hardware Logic Debug…	8/23/13	9/17/13
Signal Testing	9/25/13	9/28/13
Durability Testing	9/16/13	9/27/13
User Interface Testing	9/9/13	10/5/13
Final Full Testing	10/7/13	12/5/13
Final Report & Presentation	12/1/13	12/20/13

Appendix C: Estimated Budget

Note: None of the parts have been ordered yet and therefore the following is a theoretical budget.

In the fall the semester, the budget will be more certain as other parts will be decided upon.

Part	Cost	Quantity	Total
PIC18F43K20 Microcontroller	$40	1	$40
ADXL362 SEN-11446 Triple Axis Accelerometer	$14.95	1	$14.95
PMB-648 GPS	$40	1	$40
SM5100B GSM Evaluation Board	$59.95	1	$59.95
Heart Sensor	$30	1	$30
SD Card - 1GB	$5	1	$5
UART Communicator	$15	2	$30
Total			$219.90

Appendix D: Realistic Constraints & Standards Form

SCHOOL OF ENGINEERING

The College of New Jersey

Realistic Constraints and Engineering Standards

PROJECT:______________The Brofskey Shield____________________________________

Team Leader:________________Steven Zilberberg__________________________________

Checklist Completed: __________Dr. Hernandez_______________ ________3/27/13___________
Technical Advisor Date

DIRECTIONS:
This checklist should be completed by the team leader and signed by thetechnical advisor.

In the self-check column, place a C if covered in body of report, a N if not covered in body of report but is covered in the remarks column, and a N/A if not applicable (in all cases, C, N, or N/a, include a justification in the remarks column).

	Self-Check C	N	N/A	Remarks
Economic	X			The device, due to the technology inside of it will be slightly costly compared to other systems, but still affordable to the average person.
Environmental			X	The device will have negligible impacts to the environment
Social		X		This is not to be meant as a social device. It will have the appearance of a normal watch to reduce social self-consciousness.
Political		X		Political issues that are possible are in the events of false negatives. This would have to be backed by many different constraints and agreements
Ethical			X	There isn't an ethical issue viewable to me
Health and Safety			X	While constructing the device, proper precautions handling equipment will be used although this is a low powered device and there will not carry a large risk of safety.
Manufacturability	X			Manufacturing and repairability will be touched on in the report. Repair should be simple because there are no moving parts and initial manufacturing is simple.
Sustainability	X			Easily sustainable because there is no much wear and tear
Standards	X			There needs to be high standards to prevent lawsuits and to maximize device functionality.

Appendix E: Realistic Constraints

Economic

There is a cost in the construction of The Medical Monitoring device. The total cost of parts is estimated to be around $219.90 as of the end of the Fall Semester.

Environmental

There are negligible impacts on the environment so long as the batteries and the other parts are recycled properly.

Social

This is a medical device and not meant to be a social device. The user of the device should be the only user.

Political

The political aspect of The Medical Monitoring device is only applicable if the device was to be mass produced.

Ethical

The ethical component of the device would be the standards the constructor/designer would need to follow. If the manufacturer cut corners that could result in a loss of life if the device was to be mass produced

Health and Safety

While constructing the device, proper precautions handling equipment will be used although this is a low powered device and there will not carry a large risk of safety.

Manufacturability

The cost of the The Medical Monitoring device was kept as low as possible and materials were used that could easily be manufactured and assembled.

Sustainability

The goal of the device is to be able to sustain itself for at least a week before needing a new power source

Standards

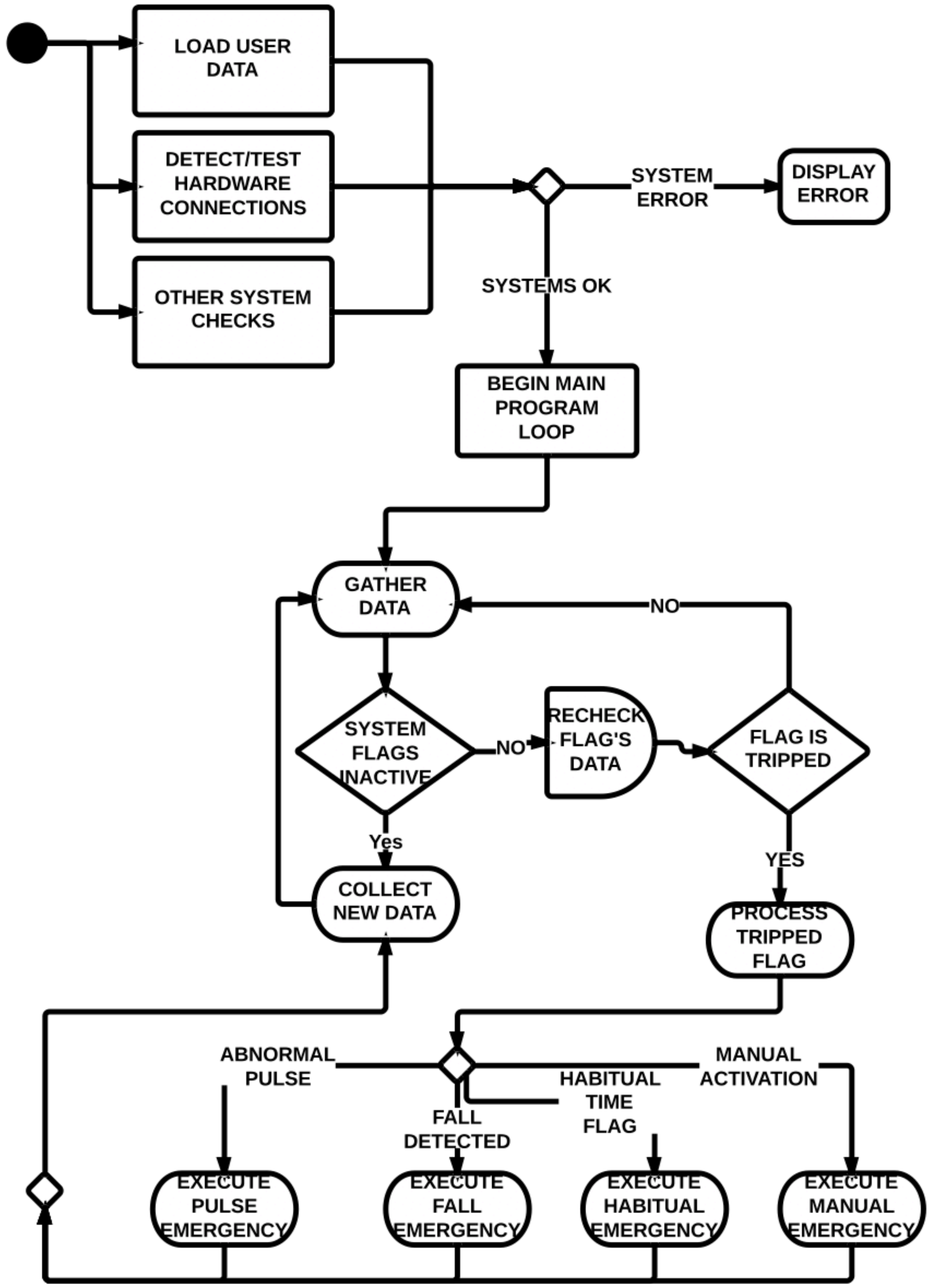
As construction commences and further documentation progresses, IEEE standards will be kept with the devices construction.

Appendix F: Engineering Standards

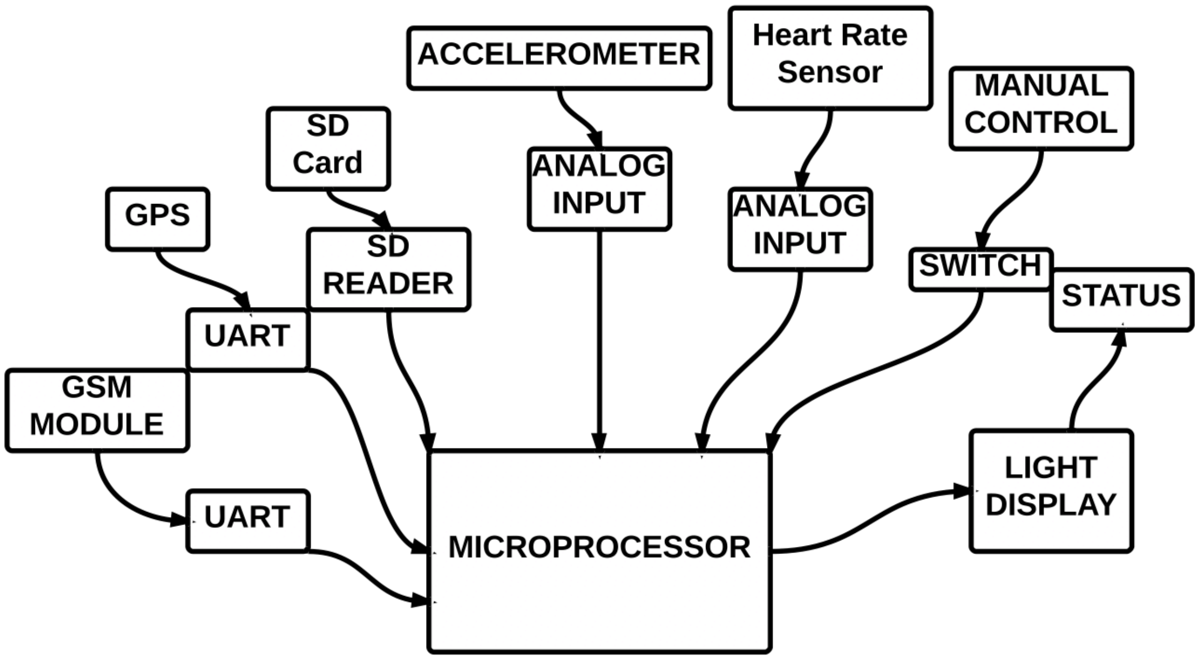
As of now, there are no standards the need to be followed since no physical components of the device have been purchased/constructed

Appendix G: Figures List

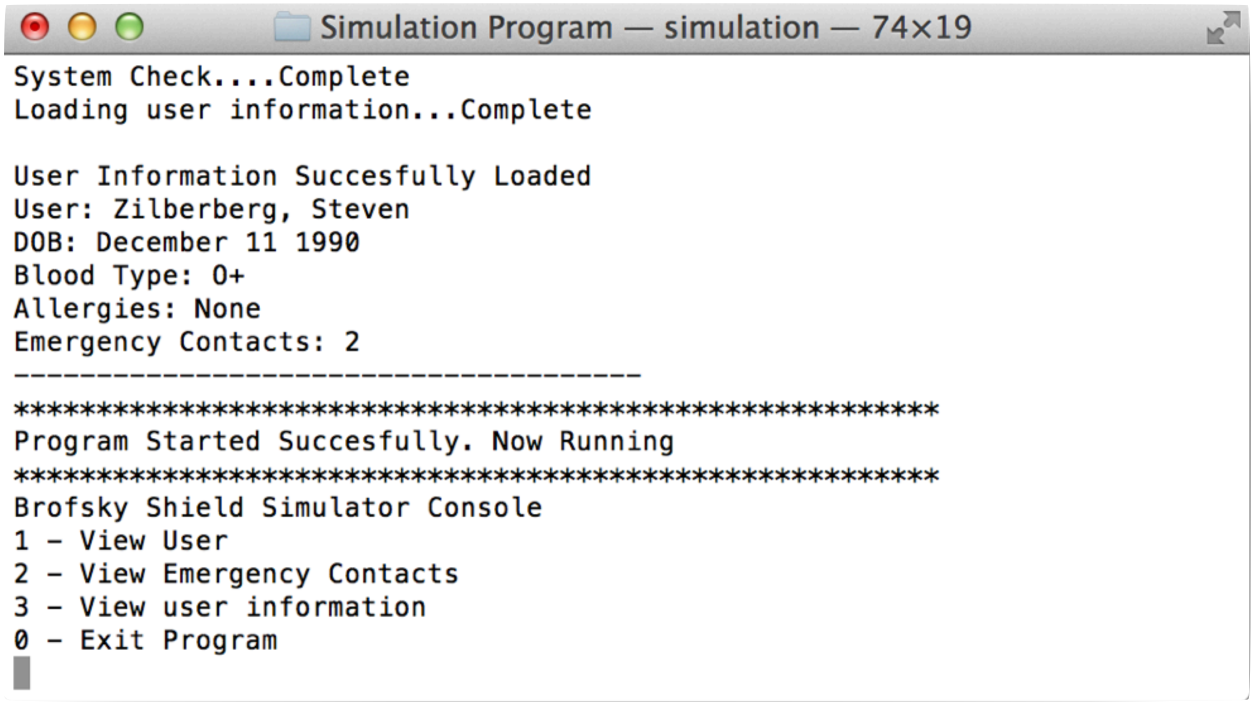
G-1 : Software Abstraction



G-2: Hardware Abstraction



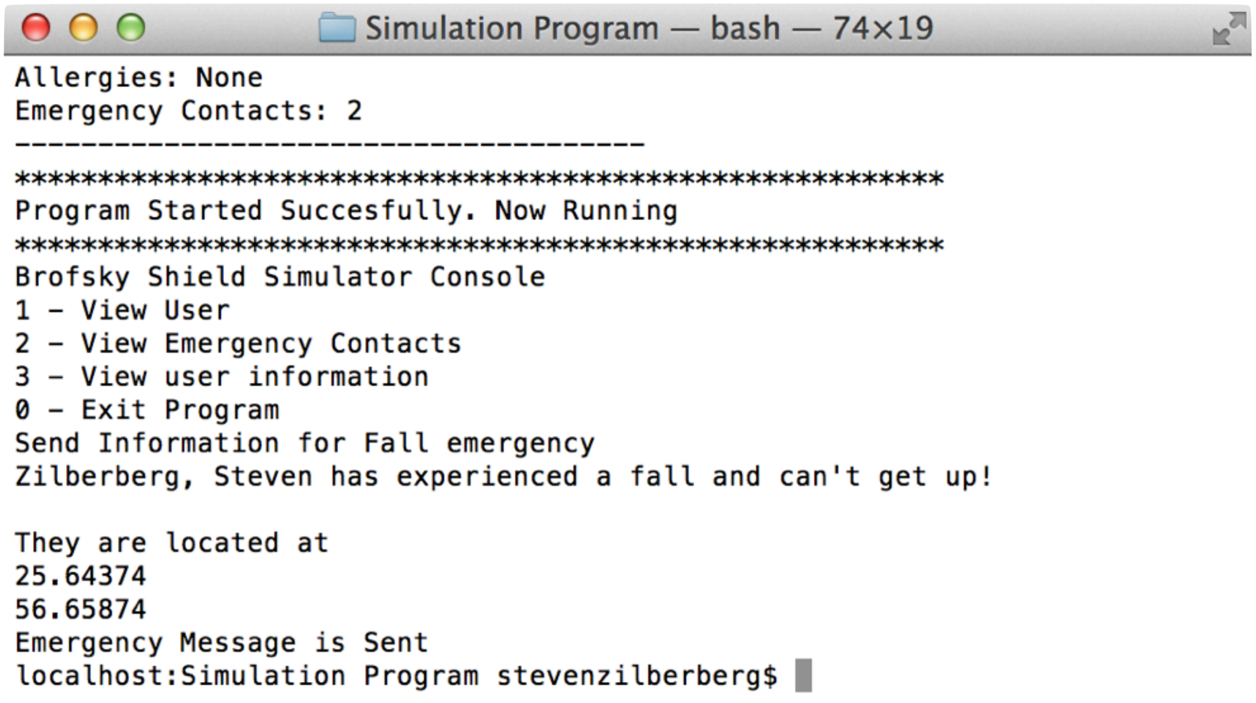
G-3: Simulation Screenshot



```
System Check....Complete
Loading user information...Complete

User Information Succesfully Loaded
User: Zilberberg, Steven
DOB: December 11 1990
Blood Type: O+
Allergies: None
Emergency Contacts: 2
--------------------------------------
********************************************************
Program Started Succesfully. Now Running
********************************************************
Brofsky Shield Simulator Console
1 - View User
2 - View Emergency Contacts
3 - View user information
0 - Exit Program
```

G-4: Simulation Screenshot Simulated Emergency



```
Allergies: None
Emergency Contacts: 2
--------------------------------------
********************************************************
Program Started Succesfully. Now Running
********************************************************
Brofsky Shield Simulator Console
1 - View User
2 - View Emergency Contacts
3 - View user information
0 - Exit Program
Send Information for Fall emergency
Zilberberg, Steven has experienced a fall and can't get up!

They are located at
25.64374
56.65874
Emergency Message is Sent
localhost:Simulation Program stevenzilberberg$
```

Appendix H: Source Code

Simulation: Main Function - codeSimulation.cpp

```
//Senior Project Logic Code Simulation
//Steven Zilberberg

#include <iostream>
#include <pthread.h>
#include <unistd.h>
#include <fstream>
#include "PersonProfile.h"
using namespace std;

//Constants
const int kNumberOfUserItems = 10;
const string kUserProfileFile = "UserData.txt";
const string kHeartDataFile = "HeartData.txt";
const string kAccelDataFile = "AccelData.txt";

        //Error Codes
const int kError_NoErrors = 0;
const int kError_AccelerometerRead = 1;
const int kError_HeartRateRead = 2;

//System Check
bool systemCheck();
bool checkGSMSensor();
bool checkGPSSensor();
bool checkHeartRateSensor();
bool checkFiles();

//User Information
bool loadUserInformation(PersonProfile&);
void displayUser(PersonProfile);
void getUserLocation();

//Get Data Functions
int getData();
bool readAccelerometer();
bool readHeartRate();

//File I/O
bool check(string file);

//Housekeeping Functions
void flush();
void waitForEnter();

//MultithreadFunctions
void * programUserMenu(void* passedPerson);
PersonProfile newPerson = PersonProfile();
```

```
int main()
{

	bool systemOK = false, userLoaded = false, programRunning = false;


	cout << "System Check....";
	systemOK = systemCheck();
	if(systemOK)
		cout << "Complete" << endl;
	else
		cout << "Fail" << endl;

	if(systemOK)
	{
		cout << "Loading user information...";
		userLoaded = loadUserInformation(newPerson);
		if(userLoaded)
			cout << "Complete" << endl << endl;
		else
			cout << "Fail" << endl << endl;
	}

	if(systemOK && userLoaded)
	{
		cout << "User Information Succesfully Loaded" << endl;
		displayUser(newPerson);
		cout << "------------------------------------------" << endl;

		programRunning = true;
		cout << "**************************************************" << endl;
		cout << "Program Started Succesfully. Now Running" << endl;
		cout << "**************************************************" << endl;

		pthread_t menuThread;
		pthread_create( &menuThread, NULL, programUserMenu, &newPerson);

		while(programRunning)
		{
			sleep(1);
			int errorFlag = getData();
			if(errorFlag != kError_NoErrors)
				programRunning = false;

			//NEED to look into multithreading...DONE :D
			switch (errorFlag)
			{
				case kError_NoErrors:
				{
					//No Errors
					break;
				}
				case kError_AccelerometerRead:
					{
						cout << "Send Information for Fall emergency" << endl;
						cout << newPerson.getName() << " has experienced a fall and 
can't get up!\n" << endl;
						cout << "They are located at ";
						getUserLocation();
						cout << endl;
						exit(kError_AccelerometerRead);
						break;
					}
					case kError_HeartRateRead:
					{
						cout << "Send Information for heart rate emergency" << endl;
						cout << newPerson.getName() << " is experiencing a heart 
related emergency. Current BPM: 28\n";
						cout << "They are located at ";
						getUserLocation();
						cout << endl;
						exit(kError_HeartRateRead);
						break;
					}
				}
			}
	}
	else
	{
		programRunning = false;
		cout << "******************************************************" << endl;
		cout << "Error. Program Terminated" << endl;
		cout << "******************************************************" << endl;
	}
}

void * programUserMenu(void* passedPerson)
{
	int option;
	while(true)
	{
		cout << "Brofsky Shield Simulator Console" << endl;
		cout << "1 - View User" << endl;
		cout << "2 - View Emergency Contacts" << endl;
		cout << "3 - View user information" << endl;
		cout << "0 - Exit Program" << endl;
		cin >> option;

		switch (option)
		{
			case 0:
			{
				cout << "Program now Terminated" << endl;
				exit(kError_NoErrors);
				break;
			}
			case 1:
```

```
                                {
                                        cout << "User: " << newPerson.getName() << endl;
                                        waitForEnter();
                                        break;
                                }
                                case 2:
                                {
                                        newPerson.viewAllContacts();
                                        waitForEnter();
                                        break;
                                }
                                case 3:
                                {
                                        cout << newPerson.toString() << endl;
                                        waitForEnter();
                                        break;
                                }
                                default:
                                {
                                        break;
                                }
                        }
                }
}

bool systemCheck()
{
        bool gsmOK = checkGSMSensor();
        bool gpsOK = checkGPSSensor();
        bool heartOK = checkHeartRateSensor();
        bool filesFound = checkFiles();
        if(gsmOK && gpsOK && heartOK && filesFound)
                return true;
        else
                return false;
}

bool checkGSMSensor()
{
        return true;
}

bool checkGPSSensor()
{
        return true;
}

bool checkHeartRateSensor()
{
        return true;
}

bool checkFiles()
{
	bool userData = check(kUserProfileFile);
	bool accelData = check(kAccelDataFile);
	bool heartData = check(kHeartDataFile);

	if(userData && accelData && heartData)
		return true;
	else
		return false;
}

bool loadUserInformation(PersonProfile& person)
{
	bool userProfileFound = check(kUserProfileFile);

	if(userProfileFound)
	{
		ifstream inFile(kUserProfileFile.c_str());
		string token;
		inFile >> token;				//get first name
		person.setFirstName(token);
		inFile >> token;				//Get last name
		person.setLastName(token);
		getline(inFile, token);			//Dummy Pull
		getline(inFile, token);			//Get birthday
		person.setBirthdayString(token);
		inFile >> token;				//get blood type
		person.setBloodType(token);
		getline(inFile, token);			//Dummy Pull
		getline(inFile, token);			//Get Allergies
		person.setAllergies(token);
		while(getline(inFile, token))	//Get Emergency Contacts
		{
			person.addNewContact(Contact(token));
		}
		inFile.close();

		return true;
	}
	else
	{
		return false;
	}
}

void displayUser(PersonProfile person)
{
	cout << "User: " << person.getName() << endl;
	cout << "DOB: " << person.getBirthday() << endl;
	cout << "Blood Type: " << person.getBloodType() << endl;
	cout << "Allergies: " << person.getAllergies() << endl;
	cout << "Emergency Contacts: " << person.getNumberOfContacts() << endl;
}

bool check(string file)
```

```
{
	ifstream inFile(file.c_str());
	if(inFile)
	{
		return true;
	}
	else
	{
		cout << file << " Not Found" << endl;
		return false;
	}
}

int getData()
{
	bool accelerometerRead = readAccelerometer();
	bool heartRateRead = readHeartRate();

	if(!accelerometerRead)
		return kError_AccelerometerRead;
	if(!heartRateRead)
		return kError_HeartRateRead;
	return 0;
}

bool readAccelerometer()
{
	check(kAccelDataFile);
	ifstream inFile(kAccelDataFile.c_str());
	int val = -1;
	inFile >> val;
	if(val == 0)
		return true;
	else
		return false;
}

bool readHeartRate()
{
	check(kHeartDataFile);
	ifstream inFile(kHeartDataFile.c_str());
	int val = -1;
	inFile >> val;
	if(val == 0)
		return true;
	else
		return false;
}

void getUserLocation()
{
	cout << "\n25.64374\n56.65874";
}
```

```
void flush()
{
	int ch;
	while ((ch = cin.get()) != '\n' && ch != EOF);
}

void waitForEnter()
{
	string dummy;
	flush();
	getline(cin, dummy);
}
```

Simulation: Contact Header - Contact.h

```
//Senior Project Logic Code Simulation
//Contact Class Header
//Steven Zilberberg

#include <iostream>
using namespace std;

class Contact
{
private:
	string relation;
	string phoneNumber;
	string firstName;
	string lastName;
public:
	Contact();
	Contact(string contactData);
	Contact(string newRelation, string newPhone, string newFirst, string newLast);

	void setRelation(string newRelation);
	void setPhoneNumber(string newPhone);
	void setFirstName(string newFirst);
	void setLastName(string newLast);

	string getRelation();
	string getPhoneNumber();
	string getFirstName();
	string getLastName();

	string getContact();
};
```

Simulation: Contact Implementation - Contact.cpp

```
//Senior Project Logic Code Simulation
//Contact Class Cpp
//Steven Zilberberg

#include "Contact.h"

Contact::Contact()
{
	relation = "";
	phoneNumber = "";
	firstName = "";
	lastName = "";
}

Contact::Contact(string data)
{
	relation = data.substr(0,data.find(' '));
	data = data.substr(data.find(' ') + 1);
	firstName = data.substr(0,data.find(' '));
	data = data.substr(data.find(' ') + 1);
	lastName = data.substr(0,data.find(' '));
	data = data.substr(data.find(' ') + 1);
	phoneNumber = data.substr(0,data.find(' '));
	data = data.substr(data.find(' ') + 1);
}

Contact::Contact(string newRelation, string newPhone, string newFirst, string newLast)
{
	relation = newRelation;
	phoneNumber = newPhone;
	firstName = newFirst;
	lastName = newLast;
}

void Contact::setRelation(string newRelation)
{
	relation = newRelation;
}

void Contact::setPhoneNumber(string newPhone)
{
	phoneNumber = newPhone;
}

void Contact::setFirstName(string newFirst)
{
	firstName = newFirst;
}

void Contact::setLastName(string newLast)
{
	lastName = newLast;
}

string Contact::getRelation()
{
	return relation;
}

string Contact::getPhoneNumber()
{
	return phoneNumber;
}

string Contact::getFirstName()
{
	return firstName;
}

string Contact::getLastName()
{
	return lastName;
}

string Contact::getContact()
{
	string output = "";
	output += getLastName() + ", " + getFirstName() + "\n";
	output += getRelation() + "\n";
	output += getPhoneNumber() + "\n";

	return output;
}
```

Simulation: Person Header - Person.h

```
//Senior Project Logic Code Simulation
//PersonProfile Class Header
//Steven Zilberberg

#include <iostream>
#include "Contact.h"
using namespace std;
```

```
const int kNumberOfContacts = 10;

class PersonProfile
{
	private:
			string firstName;
			string lastName;

			string birthday;

			string bloodType;
			string allergies;

			Contact listOfContacts[kNumberOfContacts];
			int contactIndex;
	public:
			PersonProfile();

			void setFirstName(string newFirst);
			void setLastName(string newLast);
			void setBirthdayString(string newBirthday);
			void setBloodType(string newBlood);
			void setAllergies(string newAllergies);

			string getName();
			string getBirthday();
			string getBloodType();
			string getAllergies();

			string toString();

			bool addNewContact(Contact newContact);
			bool hasContacts();
			int  getNumberOfContacts();
			Contact getContactAtIndex(int index);
			void viewAllContacts();

};
```

Simulation: Person Implementation - Person.cpp

```
//Senior Project Logic Code Simulation
//PersonProfile Class Cpp
//Steven Zilberberg

#include "PersonProfile.h"

PersonProfile::PersonProfile()
{
	contactIndex = -1;
}
```

```
void PersonProfile::setFirstName(string newFirst)
{
	firstName = newFirst;
}

void PersonProfile::setLastName(string newLast)
{
	lastName = newLast;
}

void PersonProfile::setBirthdayString(string newBirthday)
{
	birthday = newBirthday;
}

void PersonProfile::setBloodType(string newBlood)
{
	bloodType = newBlood;
}

void PersonProfile::setAllergies(string newAllergies)
{
	allergies = newAllergies;
}

string PersonProfile::getName()
{
	return (lastName + ", " + firstName);
}

string PersonProfile::getBirthday()
{
	return birthday;
}

string PersonProfile::getBloodType()
{
	return bloodType;
}

string PersonProfile::getAllergies()
{
	return allergies;
}

string PersonProfile::toString()
{
	string output = "";

	output += getName() + "\n";
	output += getBirthday() + "\n";
	output += "Blood Type: " + getBloodType() + "\n";
	output += "Allergies: " + getAllergies() + "\n";
	return output;
}

bool PersonProfile::addNewContact(Contact newContact)
{
	if(contactIndex < kNumberOfContacts - 1)
	{
			contactIndex++;
			listOfContacts[contactIndex] = newContact;
			return true;
	}
	else
	{
			return false;
	}
}

bool PersonProfile::hasContacts()
{
	if(contactIndex >= 0)
			return true;
	else
			return false;
	//return contactIndex >= 0 ? true : false;
}

int  PersonProfile::getNumberOfContacts()
{
	return contactIndex + 1;
}

Contact PersonProfile::getContactAtIndex(int index)
{
	return listOfContacts[index];
}

void PersonProfile::viewAllContacts()
{
	if(contactIndex == -1)
			return;
	for(int c = 0; c <= contactIndex; c++)
	{
			Contact current = getContactAtIndex(c);
			cout << current.getContact() << endl;
	}
}
```