McMaster University

Verification and Validation

for

NANA

Version 1.0

Prepared by

Group Name: Group 3

Junaid
Nicholas
Azzam
Josh
Ahmad
Abhishek

Instructor:

Course: MECHTRON 4TB6

Table of Contents

Revisions

Version	Date	Authors	Description of Revision
0	Feb 21, 2012	Entire Group	Initial Version of Verification and Validation document
0.1	Mar 2, 2012	Entire Group	Updated document with testing results
1	Mar 23, 2012	Entire Group	Updated document with new functions and testing results

1. Introduction

This project is aimed at developing an automated system to tend to the needs of the elderly and sick. In today's society, many retirement and nursing homes employ personnel, trained to provide health care for patients. However, in the modern world of technological advancement, it is only natural to imagine an automated system that would aid in the healthcare and well-being of those who cannot look after themselves. The goal of this project and hence the final product, is to aid the aging population in living a comfortable lifestyle, to monitor their health especially when no one is around to do it for them.

1.1 Purpose

This document outlines the verification and validation methods for the Home Monitoring and Response System for Older/Ill people. For information regarding the system requirements of our design refer back to Software Requirements Specification for Elderly Care System. The document will provide analysis of the interfaces required for this particular system to operate with sufficiency and ease. This document will be used during all successive steps of the development cycle to show the product functionality and architecture.

1.2 Problems in the World

As in many developed countries around the globe, Canada's population is aging. In 2010, about 14% (4.8 million) of Canadians were seniors (those age 65 and older).[1] By 2036, this proportion will rise to about 25% (10.4 million).[2] In 2011, the first members of the largest birth cohort in Canada's recent history—the baby boom generation—turned age 65.[3] As a result, the aging of Canada's population has accelerated.[2, 3]

While Canada's population is still relatively young compared with that of many other developed nations,[3, 4] its accelerated aging has raised some alarm bells. Members of the media and the general public have expressed concerns that Canada's health care system will be unable to meet the growing health care needs of an aging population and have

called into question the overall sustainability of the system as a result.[5] Research, however, suggests that population aging has contributed only modestly to increases in health expenditure to date.[6–8] Instead, the evidence points to factors beyond population aging, such as general population growth and general health service utilization patterns, that will likely require Canada's health care system to change and adapt.[7–9] Still, the costs and resource needs of health and social care for seniors should not be underestimated. About 45% of provincial and territorial governments' health care expenditure in 2009 was spent on seniors,[10] yet this group accounts for only 14% of the population. Seniors are more frequent users of several sectors of Canada's health care system [11] and utilize the system in different ways and with different intensity than other age groups.

From this research and statistics we see a growing demand for helping our aging population. Our solution satisfies that demand by helping the elderly live comfortably and longer through a monitoring service. The current market solutions are not feasible as they only contain single components like fall detection. Our solution gives way to a full system that detects fall detection, heart rate, location sensing, and environmental controls. It has the power to make elderly living more comfortable and secure.

1.3 Background Research

Refer to Background Research of the *System Requirements Specification for Elderly Care System* for a description of the required behavior expected from the system.

1.4 Product Scope

The focus of this document will be on specifying the design of hardware and software that will be used to complete the project. The hardware includes the physical chest wear, a processing unit, and monitoring station to incorporate certain sensors to satisfy our previous document, Software Requirements Specification for Elderly Care System.

1.5 Intended Audience and Overview of Document

This document is intended for the developers of the home monitoring and response system as well as the users. It is also intended for the course instructor and teaching assistants of the course Software/Mechatronics Engineering 4G06/4TB6. The rest of this document outlines the overall description, specific requirements and non-functional requirements of the system. This entire document is written for those who are concerned about the technical aspects and details of the system. Those who are less concerned about the detail and only the general usage of such a system should resort to reading only the Introduction and Overall Description, parts 1 and 2 of the documents, respectively.

1.6 Definitions, Acronyms and Abbreviations

Term	Definition
System	In the context of this document, system is a short form and refers to the home monitoring and response system for older/ill people
Sensors	A converter that measures a physical quantity and converts it into a signal which can be read by an electronic instrument
Operational Maintenance Teams	Personnel and staff members which relay the signals triggered by the monitoring system to the appropriate source
Trigger	An activation of a signal or response to be generated
ChestWear	A physical device worn by the user to detect and monitor heart rates and pulses
Base Station	A physical device to be installed in the homes of the user to assist in picking up and relaying signals from the chest piece to the operational maintenance teams
Heart Rate	The number of heart beats per unit of time, typically expresses as beats per minute
Beats Per Minute (BPM)	A measure for the heart rate

Signal	A function that conveys information about the behavior or attributes of some phenomenon
TBD	More definitions to come as the project progresses

Table 1: Variables and Definitions

1.7 Variables

Variable	Definition	Type
m_heartRate	Measured value of the heart rate sensor	BPM
in_heartRate	Output of heart rate sensor	V/BPM
m_accelerometerST, m_accelerometer	Inputs to the accelerometer	Degrees/%/g
Vout_X	Output to the accelerometer, to measure g(acceleration) on the x axis, datasheets provide mv/g sensitivity	mV / V
Vout_Y	Output to the accelerometer, to measure g(acceleration) on the y axis, datasheets provide mv/g sensitivity	mV/ V
m_emergPush	Button pushed or not	Button up or down
in_emergPush	Send high voltage (5V) if pushed 0 if not, input to microcontroller	V
m_doorSensor	Input sensor to any entry/exit doors detecting if chest piece within home	V
in_doorSensor	Output signal to the base station to indicate user is in /out of the home	V
out_microdata	Input to Bluetooth, formats inputs from the sensors into a transferable form.	data
out_bluetooth	Input to base station, packets sent out to the base station	Data, id, security
c_bluetoothOn	Controls if the Bluetooth is on or off	V
in_monitoring	Output from base station to monitoring, signals what	Situation, ID,

	situation occurred through processed results from the sensors of the chest piece.	security formed in packets
out_call911	Monitoring calls 911 alerting them of an emergency. Critical information regarding the event should be given such as the name, location, age, pertinent medical history, and the situation and urgency of the situation reported.	Identification(User's Name, Location, Age), Situation
out_callHome	Monitoring calls the user's home to ensure the client is okay or to ensure the client is actually in need of medical attention. Provides information that the sensors can't easily detect.	Questions
out_callTech	Calls technicians to fix problems with the base station or chest piece.	Product ID, Location
out_emergPush	Input to base station, packets sent out to base station	V

Table 2: Variables

1.8 References

IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998.

1. Statistics Canada, population estimates 1971–20 1. 10 (Ottawa, Ont.: Statistics Canada, 2010).

2. Statistics Canada, *Population Projections for Canada, Provinces and Territories: 2009 to 2036* (Ottawa, Ont.: Statistics Canada, 2010).

3. L. Martel and E. C. Malenfant, *Portrait of the Canadian Population in 2006, by Age and Sex, 2006 Census* (Ottawa, Ont.: Statistics Canada, 2007).

4. Organisation for Economic Co-operation and Development, *OECD Factbook 2010: Economic, Environmental and Social Statistics* (Paris, France: OECD, 2010).

5. Canadian Health Services Research Foundation, *Sustainability of Canada's Healthcare System. Commission on the Future of Health Care in Canada* (Ottawa, Ont.: CHSRF, 2002).

6. R. G. Evans et al., “Apocalypse No: Population Aging and the Future of Health Care Systems,” presented at SEDAP Conference on Population Aging, the Health Care System, and the Economy in Burlington, Ontario, on April 27, 2001.

7. Canadian Institute for Health Information, *Health Care Cost Drivers: The Facts*. (Ottawa, Ont.: CIHI, 2011).

8. A. Constant et al., *Research Synthesis on Cost Drivers in the Health Sector and Proposed Policy Options* (Ottawa, Ont.: Canadian Health Services Research Foundation, 2011).

9. Canadian Health Services Research Foundation, *Better With Age: Health Systems Planning for the Aging Population—Synthesis Report* (Ottawa, Ont.: CHSRF, 2011).

10. Canadian Institute for Health Information, *National Health Expenditure Trends*, 1975 to 2011 (Ottawa, Ont.: CIHI, 2011).

11. M. Rotermann, “Seniors’ Health Care Use,” *Health Reports* 16, Suppl. (2006): pp. 33–45, Statistics Canada catalogue no. 82-003.

12. Analog Dialogue. Detecting Human Falls with a 3-Axis Digital Accelerometer. Retrieved from the World Wide Web: http://www.analog.com/library/analogdialogue/archives/43-07/fall_detector.html

13. SparkFun: ADXL335, *three-axis accelerometer.* Retrieved from the World Wide Web, on December 2, 2012 from: http://elinux.org/SparkFun:_ADXL335,_three-axis_accelerometer#Inputs_and_Outputs

14. Seeed Wiki. Piezo Sensor - LDT1-028K Lead Attachments. Retrieved from the World Wide Web, on December 5, 2012 from: http://www.seeedstudio.com/wiki/Piezo_Sensor_-_LDT1-028K_Lead_Attachments

15. Arduino. Arduino Mega 2560. Retrieved from the World Wide Web, on December 5, 2012 from: http://arduino.cc/en/Main/ArduinoBoardMega2560

16. “ADXL345: bildr blog”, Retrieved from the World Wide Web: March 2, 2013, from http://bildr.org/2011/03/adxl345-arduino/

2. Hardware Prototypes

Prototype 1 (Figure 2.1) focused on the basics of getting serial input from various sensors. Our initial sensors were a pushbutton, HRMI, and Accelerometer(AD335), LCD Screen. The Screen was to output results of X, Y, Z of the accelerometer. The accelerometer has since been replaced.

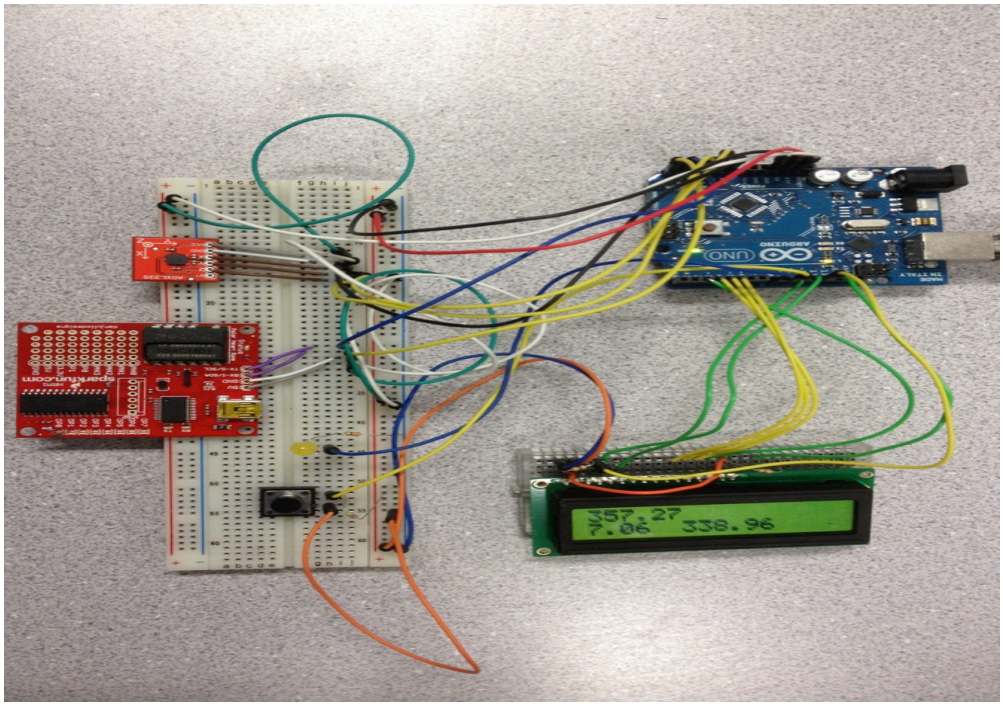


Figure 2.1: Prototype 1

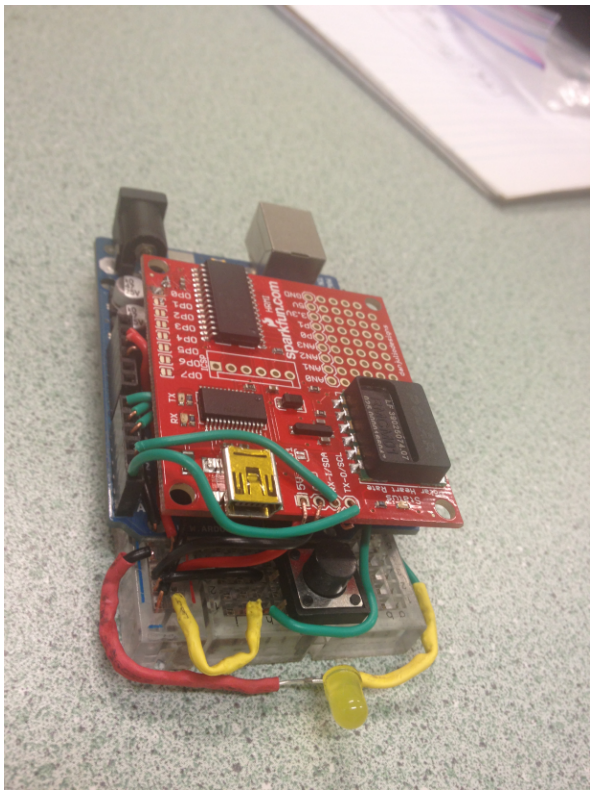
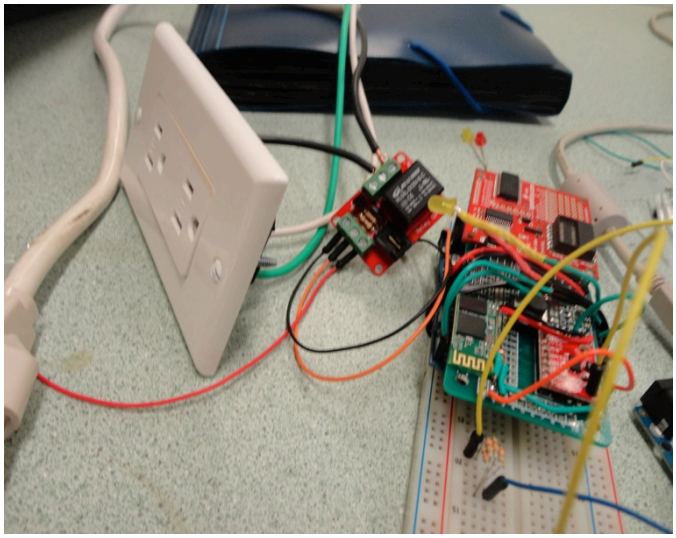


Figure 2.2: Prototype 2

Prototype 2 (Figure 2.2) focused on the condesnsing the wiring and space of the system. All functionalty from prototype#1 are intact.



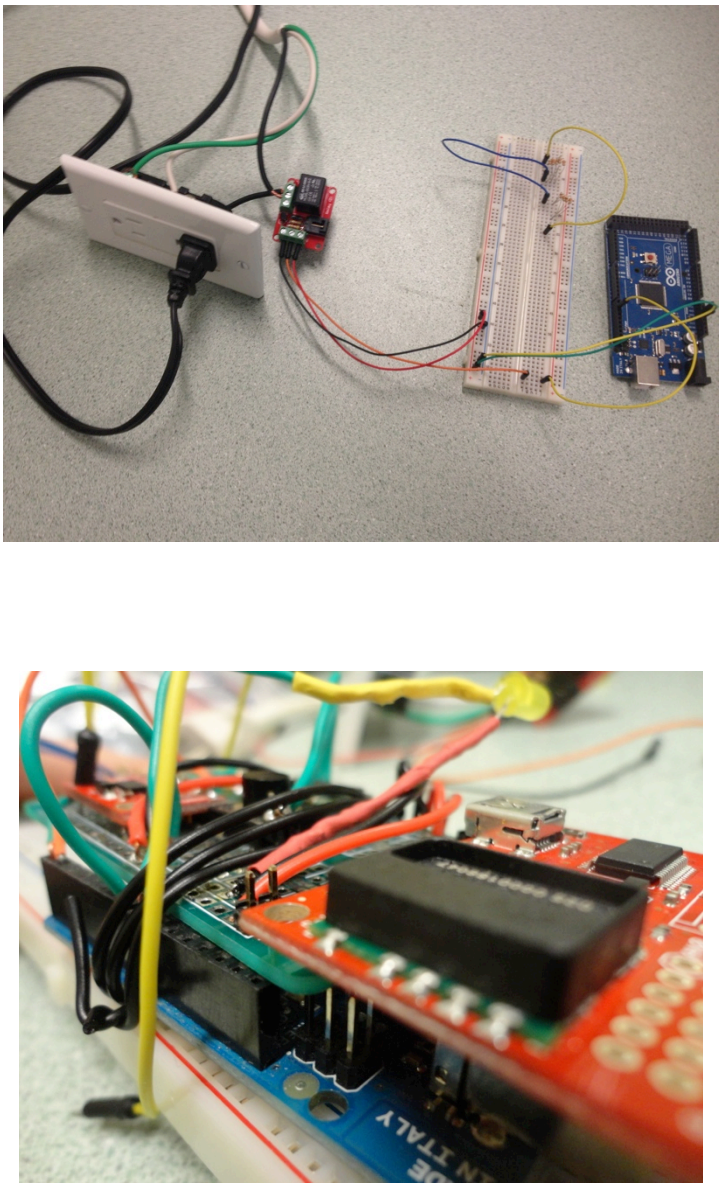


Figure 2.3: Prototype 3 (Current Phase)

Prototype 3 (Figure 2.3) comprises all the sensors, bluetooth, and pushbutton soldered directly to a pcb shield. Relay system is fully functional with our current system.

3. Hardware Testing

3.1 Method of Testing

Each hardware is tested using the box approach. This method of testing describes the point of view that we take when designing test cases. Using white box testing or black box testing, it uses external descriptions of the hardware, including specifications, requirements, and designs to derive test cases. The functionalities can be examined without any knowledge of the internal implementation.

The aim of this testing is to test the functionalities of hardware according to applicable requirements.

3.2 Heart Sensing

In order to allow Heart Sensing within our application, the system must be able to connect to HRMI (Heart Rate Monitor Interface) using I2C protocol. It must also setup the type of algorithm to use. The current functions below allow for heart sensing performing accurately within our system.

The system currently is reading the data inputs of heart sensing. However, we stated in our goals that processing of heart situations must be included. The processing of the heart will be completed at a later stage of the project.

3.2.1 Unit Testing

3.2.1.1 Black Box Specifications – Expected Conditions

Heart Sensing on the hardware side requires us to analyze all the functions necessary to read values from the Polar Heart Rate Monitor Interface. The bottom functions setHeartMonitor, getHeartRate, writeRegister, hrmiGetData will be tested.

```
//---------------------------------------------------------------------
// Setup Heart Monitor Function
//---------------------------------------------------------------------

void setupHeartMonitor(int type){
       Wire.begin();//setup the heartrate monitor
       writeRegister(HRMI_I2C_ADDR, 0x53, type);
       // Configure the HRMI with the requested algorithm mode
}
```

Expected Conditions through setupHeartMonitor(int type)

Input Variables			Output Response
int type	HRMI_HR_ALG 0		Raw Sample Setup
	HRMI_HR_ALG 1		Average Sample Setup
writeRegister(p1, p2, p3)	writeRegister(p1,,)	HRMI_I2C_ADDR = 127	HRMI configured with I2C
	WriteRegister(,p2,)	HEX Command value = 0x53	SetMode of heartRate Algorithm
	WriteRegister(,,p3)	type = 0 or 1	Raw Setup/Average Setup

```
//---------------------------------------------------------------------
// Acquire Heart Rate Function
//---------------------------------------------------------------------

int getHeartRate(){
        //get and return heart rate
        //returns 0 if we couldn't get the heart rate
        byte i2cRspArray[3]; // I2C response array
        i2cRspArray [2] = 0;
        int noBeat = 0;

        writeRegister(HRMI_I2C_ADDR,  0x47, 0x1); // Request a set of heart rate values

        if (hrmiGetData(127, 3, i2cRspArray)) {
                noBeat = ((i2cRspArray[0] & (1 << 2-1)) != 0);
                if (noBeat == 1) return 0;
                        return i2cRspArray[2];
        }
        else{
                return 0;
        }
}
```

Expected Conditions through getHeartRate()

Input Variables			Output Response
hrmiGetData(p1,p2,p3)	hrmiGetData(p1,,)	HRMI_I2C_ADDR = 127	Raw Sample Setup
	hrmiGetData(,p2,)	numBytes < 0	No request made
		numBytes >= 0	Request will be made
	hrmiGetData(,,p3)	dataArray = NULL	No dataArray used
		dataArray != NULL	DataArray will be filled
writeRegister(p1,p2,p3)	writeRegister(p1,,)	HRMI_I2C_ADDR = 127	HRMI configured with I2C
	WriteRegister(,p2,)	HEX Command value = 0x47	SetMode of heartRate Algorithm
	WriteRegister(,,P3)	type = 0 or 1	Raw Setup/Average Setup
i2cRspArray[3]	hrmiGetData(p1,p2,p3) = TRUE	noBeat = 1	Pulse 0 read
		noBeat = 0	Pulse Value read
	hrmiGetData(p1,p2,p3) = FALSE		Pulse 0 read

```
//---------------------------------------------------------------------
// Write Register Function
//---------------------------------------------------------------------

void writeRegister(int deviceAddress, byte address, byte val) {
        //I2C command to send data to a specific address on the device
        Wire.beginTransmission(deviceAddress); // start transmission to device
        Wire.write(address);       // send register address
        Wire.write(val);           // send value to write
        Wire.endTransmission();    // end transmission
}
```

Expected Conditions through writeRegister()

Input Variables		Output Response
int deviceAddress	Specified address to communicate to I2C HRMI_I2C_ADDR = 127	Wire.beginTransmission successful
	Unspecified Address not related to HRMI	Wire.beginTransmission Unsuccessful
byte address	HEX Command value = 0x47	Wire.write(address) Successful
	Unspecified Command value from HRMI	Wire.write(address) Unsuccessful
byte val	HRMI_HR_ALG 0	Raw Sample Setup
	HRMI_HR_ALG 1	Average Sample Setup

```
//---------------------------------------------------------------------
// HRMI getData Function
//---------------------------------------------------------------------

boolean hrmiGetData(byte addr, byte numBytes, byte* dataArray){
        //Get data from heart rate monitor and fill dataArray byte
        //Returns true if it was able to get it, false if not
        Wire.requestFrom(addr, numBytes);
        if (Wire.available()) {
                for (int i=0; i<numBytes; i++){
                        dataArray[i] = Wire.read();
                }
                return true;
        }
        else{
                return false;
        }
}
```

Expected Conditions through getData()

Input Variables		Output Response
byte addr	HRMI_I2C_ADDR = 127	HRMI configured with I2C
byte numBytes	numBytes < 0	No request
	numBytes >= 0	requestFromWire
byte* dataArray	dataArray = NULL	No dataArray to Use
	dataArray != NULL	dataArray to be filled
Wire.Available	Wire.Available = TRUE	dataArray[i] = Wire.read()
	Wire.Available = FALSE	!dataArray[i] = Wire.read()

3.2.1.2 Black Box Specifications – Expected Conditions Methods

3.2.1.2.1 Test Wired I2C Connection

The method for testing the I2C connection is to verify the connection can send data serially through the microcontroller to the computer. We chose this method because it would help with Bluetooth testing later in the project.

Input Response	Expected Response	Output Response	Pass/Fail
Open wire connection with Wire.begin and setup with writeRegister	Connection between Ardunio and C# Successful	Successful Connection	Pass

3.2.1.2.2 Check The Setup for setHeartMonitor

The method for testing setup is to bring the hear receiver close to the transmitter and see if it configures correctly to give units beats/min. We chose this method because it would help determine if setup was portrayed accurately.

Input Response	Expected Response	Output Response	Pass/Fail
Program the Arduino with correct references to the HRMI and bring the transmitter close to HRMI	Readings in Serial Monitor in B/M units	Successful – Beats/min accurately displayed on Interface	Pass

3.2.1.2.3 Testing Steady Pulse

The method for testing this parameter is for the user to breathe constantly for 30 sec and get correct readings within +/- 5 heartbeats/min. We chose this method because it would help determine if sensor is consistent.

Input Response	Expected Response	Output Response	Pass/Fail
User is breathing at constant pace	Constant average HB/min	Successful – Beats/min displayed within +/-5 range	Pass
User is breathing at constant pace while moving.	Constant average HB/min	Successful – Beats/min displayed within +/-5 range	Pass

3.2.1.2.4 Testing Increasing Pulse

The method for testing this parameter is for the user to breathe at a growing rate for 30 sec and get correct readings within +/- 5 heartbeats/min. We chose this method because it would help determine if sensor is consistent.

Input Response	Expected Response	Output Response	Pass/Fail
User is breathing at growing pace	Constant increase HB/min	Successful - Beats/min displayed within +/-5 range	Pass
User is breathing at growing pace while moving.	Constant increase HB/min	Successful - Beats/min displayed within +/-5 range	Pass

3.2.1.2.5 Testing Decreasing Pulse

The method for testing this parameter is for the user to breathe at a decelerating rate for 30 sec and get correct readings within +/- 5 heartbeats/min. We chose this method because it would help determine if sensor is consistent.

Input Response	Expected Response	Output Response	Pass/Fail
User is breathing at decreasing pace	Constant decrease HB/min	Successful - Beats/min displayed within +/-5 range	Pass
User is breathing at decreasing pace while moving.	Constant decrease HB/min	Successful - Beats/min displayed within +/-5 range	Pass

3.2.1.3 Black Box Specifications – Boundary Conditions Methods

3.2.1.3.1 Heart Receiver about 1cm within Transmitter

The method for testing this parameter is to wear the device in close proximity. We chose this method because it would help determine the connectivity for results for HB/min.

Input Response	Expected Response	Output Response	Pass/Fail
User is within 1cm of transmitter	Sensor is working	Successful - Beats/min accurately displayed on Interface	Pass

3.2.1.3.2 Heart Receiver about 75cm within Transmitter

The method for testing this parameter is to wear the device in its limits. We chose this method because it would help determine the connectivity issues when at limits.

Input Response	Expected Response	Output Response	Pass/Fail
User is within 75cm of transmitter	Sensor is working	Successful - Beats/min accurately displayed on Interface	Pass

3.2.1.3.3 Transmitter moving within 10-75cm of the Heart Receiver

The method for testing this parameter is to wear the device in its limits. We chose this method because it would help determine the connectivity issues when at limits.

Input Response	Expected Response	Output Response	Pass/Fail
User is within 10-75cm of transmitter	Sensor is working	Successful - Beats/min accurately displayed on Interface	Pass

3.2.1.4 Black Box Specifications – Off Nominal Conditions Methods

3.2.1.4.1 Transmitter 1m away from Heart Receiver

The method for testing this parameter is to wear the device out of its limits. We chose this method because it would help determine the connectivity issues when beyond limits.

Input Response	Expected Response	Output Response	Pass/Fail
User is beyond 1m of transmitter	Sensor is not working	Successful - Beats/min accurately displayed on Interface	Pass

3.2.1.4.2 Heart Receiver attached, but no pulse detected

The method for testing this parameter is to wear the device and get a response that is zero. We chose this method because it would help us respond to emergency situations.

3.2.1.5 Black Box Specifications – Varying Order Methods

3.2.1.5.1 Vary the order of inputs for the Boundary Conditions

With constant varying of the inputs above, we found as a group that there is no impact on the results. The heart sensor behaves correctly and as it should.

3.2.1.5.2 Vary the order of inputs for the Off Nominal Conditions

With constant varying of the inputs above, we found as a group that there is no impact on the results. The heart sensor behaves correctly and as it should.

3.2.1.6 White Box Specifications – Path Completeness

For Heart Sensing, the total number of output responses are 28 which are from the functions setHeartMonitor, getHeartRate, writeRegister, and hrmiGetData.

- setHeartMonitor – 5 output responses
- getHeartrate – 10 output responses
- writeRegister – 6 output responses
- hrmiGetData – 7 output responses

The path of the Heart Sensor is as follows. The transmitter receives a raw data from the user through the Polar Heart Transmitter. Then, the Polar Receiver receives data through magnetic fields to store in certain addresses. We setup the addresses to read the data in Ardunio and observe the data for analyzing.

3.3 Accelerometer

The ADXL-335 accelerometer chip should be able to accurately report angle orientations as well as fall detection. The implementation of the accelerometer should be able to employ techniques that trigger only positive ‘fall detections’.

3.3.1 Unit Testing

3.3.1.1 Individual Hardware Testing

3.3.1.1.1 Black Box Specifications – Expected Conditions

Sensing orientation via the ADXL 345 chip requires initialization of coordinates in the X, Y and Z direction.

```
//--------------------------------------------------------------------------------------
// Read Accelerometer Function
//--------------------------------------------------------------------------------------
#include <Wire.h>
#include <ADXL345.h>
```

```
ADXL345 adxl; //variable adxl is an instance of the ADXL345 library

void setup(){
  Serial.begin(9600);
  adxl.powerOn();
  //set activity/ inactivity thresholds (0-255)
  adxl.setActivityThreshold(75); //62.5mg per increment
  adxl.setInactivityThreshold(75); //62.5mg per increment
  adxl.setTimeInactivity(10); // how many seconds of no activity is inactive?
  //look of activity movement on this axes - 1 == on; 0 == off
  adxl.setActivityX(1);
  adxl.setActivityY(1);
  adxl.setActivityZ(1);
  //look of inactivity movement on this axes - 1 == on; 0 == off
  adxl.setInactivityX(1);
  adxl.setInactivityY(1);
  adxl.setInactivityZ(1);
  //look of tap movement on this axes - 1 == on; 0 == off
  adxl.setTapDetectionOnX(0);
  adxl.setTapDetectionOnY(0);
  adxl.setTapDetectionOnZ(1);
  //set values for what is a tap, and what is a double tap (0-255)
  adxl.setTapThreshold(50); //62.5mg per increment
  adxl.setTapDuration(15); //625µs per increment
  adxl.setDoubleTapLatency(80); //1.25ms per increment
  adxl.setDoubleTapWindow(200); //1.25ms per increment
  //set values for what is considered freefall (0-255)
  adxl.setFreeFallThreshold(5); //(5 - 9) recommended - 62.5mg per increment
  adxl.setFreeFallDuration(20); //(20 - 70) recommended - 5ms per increment
  //setting all interupts to take place on int pin 1
  //I had issues with int pin 2, was unable to reset it
  adxl.setInterruptMapping( ADXL345_INT_SINGLE_TAP_BIT,   ADXL345_INT1_PIN );
  adxl.setInterruptMapping( ADXL345_INT_DOUBLE_TAP_BIT,   ADXL345_INT1_PIN );
  adxl.setInterruptMapping( ADXL345_INT_FREE_FALL_BIT,    ADXL345_INT1_PIN );
  adxl.setInterruptMapping( ADXL345_INT_ACTIVITY_BIT,     ADXL345_INT1_PIN );
  adxl.setInterruptMapping( ADXL345_INT_INACTIVITY_BIT,   ADXL345_INT1_PIN );
  //register interupt actions - 1 == on; 0 == off
  adxl.setInterrupt( ADXL345_INT_SINGLE_TAP_BIT, 1);
  adxl.setInterrupt( ADXL345_INT_DOUBLE_TAP_BIT, 1);
  adxl.setInterrupt( ADXL345_INT_FREE_FALL_BIT,  1);
  adxl.setInterrupt( ADXL345_INT_ACTIVITY_BIT,   1);
  adxl.setInterrupt( ADXL345_INT_INACTIVITY_BIT, 1);
}

void loop(){
  int x,y,z;
  adxl.readAccel(&x, &y, &z);
  byte interrupts = adxl.getInterruptSource();
  Serial.print("A");
  if(adxl.triggered(interrupts, ADXL345_FREE_FALL)){
    Serial.print("2");
  }
  else if(adxl.triggered(interrupts, ADXL345_SINGLE_TAP)){
    Serial.print("1");
```

```
  }
  else{
     Serial.print("0");
  }
  Serial.print("X");
  Serial.print(x);
  Serial.print("Y");
  Serial.print(y);
  Serial.print("Z");
  Serial.print(z);
}
```

Expected Conditions through readAccel()

Input Response	Expected Response	Output Response	Pass/Fail
INT X	REG X Initialized	REG X Initialized	Pass
INT Y	REG Y Initialized	REG Y Initialized	Pass
INT Z	REG Z Initialized	REG Z Initialized	Pass

3.3.1.1.2 Test Cases – Active High/Active Low

A scheme to set the interrupts on the Arduino micro-controller for detection of different activities such as free-fall and taps

- Active High: LED Illuminated
- Active Low: No Behaviour

```
//-------------------------------------------------------------------------------------
// Set Interrupt Level Bit Function
//-------------------------------------------------------------------------------------
void ADXL345::setInterruptLevelBit(bool interruptLevelBit)
{
    setRegisterBit(ADXL345_DATA_FORMAT, 5, interruptLevelBit);
}
```

Expected Conditions through setInterruptLevelBit()

Input Response	Expected Response	Output Response	Pass/Fail
INTERRUPT SET TO 0	ACTIVE HIGH	ACTIVE HIGH	Pass
INTERUPT SET TO 1	ACTIVE LOW	ACTIVE LOW	Pass

3.3.1.1.3 Black Box Specifications – Boundary Conditions Methods

- b is between 0 and 255 gives a Interrupt for an Active High or Low
- b is 0 gives an inadmissible

```
//-------------------------------------------------------------------------------------
// Get Tap Threshold Bit Function
//-------------------------------------------------------------------------------------
int ADXL345::getTapThreshold()
{
    byte _b;
    readFrom(ADXL345_THRESH_TAP, 1, &_b);
    return int (_b);
}
```

Expected Conditions through getTapThreshold()

Input Response	Expected Response	Output Response	Pass/Fail
b $\in$ (0, 255)	INTERRUPT DETECTED (HIGH OR LOW)	INTERRUPT DETECTED (HIGH OR LOW)	Pass
b == 0	UNDEFINED	UNDEFINED	Pass

3.3.1.1.4 Test Cases – Interrupt/Undefined

- Undefined: Terminal does not print activity correctly
- Interrupt Set: Free fall is initialized and ready to be reported (acceleration detected)

```
//-----------------------------------------------------------------------------------------
// Set Free Fall Threshold Function
//-----------------------------------------------------------------------------------------
void ADXL345::setFreeFallThreshold(int freeFallThreshold)
{
  freeFallThreshold = min(max(freeFallThreshold,0),255);
  byte _b = byte (freeFallThreshold);
  writeTo(ADXL345_THRESH_FF, _b);
}
```

Expected Conditions through setFreeFallThreshold()

Input Response	Expected Response	Output Response	Pass/Fail
INTERRUPT SET TO 0	UNDEFINED	UNDEFINED	Pass
INTERRUPT SET TO 1	INTERRUPT SET	INTERRUPT SET	Pass

3.3.1.1.5 Test Cases – Free Fall/Undefined

- Free fall reported to terminal when acceleration detected

```
//-----------------------------------------------------------------------------------------
// Set Free Fall Duration Function
//-----------------------------------------------------------------------------------------
void ADXL345::setFreeFallDuration(int freeFallDuration)
{
        freeFallDuration = constrain(freeFallDuration,0,255);
        byte _b = byte (freeFallDuration);
        writeTo(ADXL345_TIME_FF, _b);
}
```

Expected Conditions through setFreeFallDuration()

Input Response	Expected Response	Output Response	Pass/Fail
freeFallDuration > THRES_FF	UNDEFINED	UNDEFINED	Pass
freeFallDuration <= THRES_FF	Free-fall Detected	Free-fall Detected	Pass

3.4 Emergency Button

When Emergency Button is pushed (Active High), LED is illuminated in response. No illumination should be detected when the Emergency is not pushed (Active Low)

3.4.1 Unit Testing

3.4.1.1 Individual Hardware Testing

3.4.1.1.1 Black Box Specifications – Expected Conditions

Below is the function to set a particular LED to detect whether the Emergency Button is pushed.

```
//-------------------------------------------------------------------------------------------
// Get Emergency Button Function
//-------------------------------------------------------------------------------------------
void getEmergencyButton (int buttonStat)
{
  if (buttonStat == 1){
     digitalWrite(ledPin, HIGH);
     buttonStat = 0; //set back to zero
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Expected Conditions through getEmergencyButton()

Input Response	Expected Response	Output Response	Pass/Fail
buttonStat == 1	ACTIVE HIGH	ACTIVE HIGH	Pass
!(buttonStat==1)	ACTIVE LOW	ACTIVE LOW	Pass

4. Software Testing

4.1 Method of Testing

Each software is tested using the box approach. This method of testing describes the point of view that we take when designing test cases. Using white box testing or black box testing, it uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. The functionalities can be examined without any knowledge of the internal implementation.

The aim of this testing is to test the functionalities of software according to applicable requirements.

4.2 Serial Communication

Serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. The device sends data to the Monitoring Station (C# interface) via serial communication. The Arduino chip collects data from the Heart Sensor, Accelerometer, RFID tag, the Emergency button and relays it to the Monitoring Interface.

The system is currently transmitting the heart rate (bpm), Accelerometer readings (x,y,z), Location with the help of RFID tag (Bathroom, Kitchen etc.) and a high/low when the emergency button is pushed.

4.2.1 Unit Testing

4.2.1.1 Individual Software Testing

4.2.1.1.1 Black Box Specifications – Expected Conditions

The table below shows the variables being sent from the Arduino board to the Interface in C#. All the values are received at the software end as expected.

```
//--------------------------------------------------------------------------------
// Packet Function
//--------------------------------------------------------------------------------
void sendPacket(int heartRate)
{
    int p_data[6];

    p_data[0] = mySensVals[0];
    p_data[1] = mySensVals[1];
    p_data[2] = mySensVals[2];
    p_data[3] = buttonStat;
    p_data[4] = heartRate;
```

```
    Serial.print("X");
    Serial.print(p_data[0], DEC);
    Serial.print("Y");
    Serial.print(p_data[1], DEC);
    Serial.print("Z");
    Serial.print(p_data[2], DEC);
    Serial.print("H");
    Serial.print(p_data[4], DEC);
    Serial.print("B");
    Serial.print(p_data[3], DEC);
}
```

Expected Conditions through sendPacket()

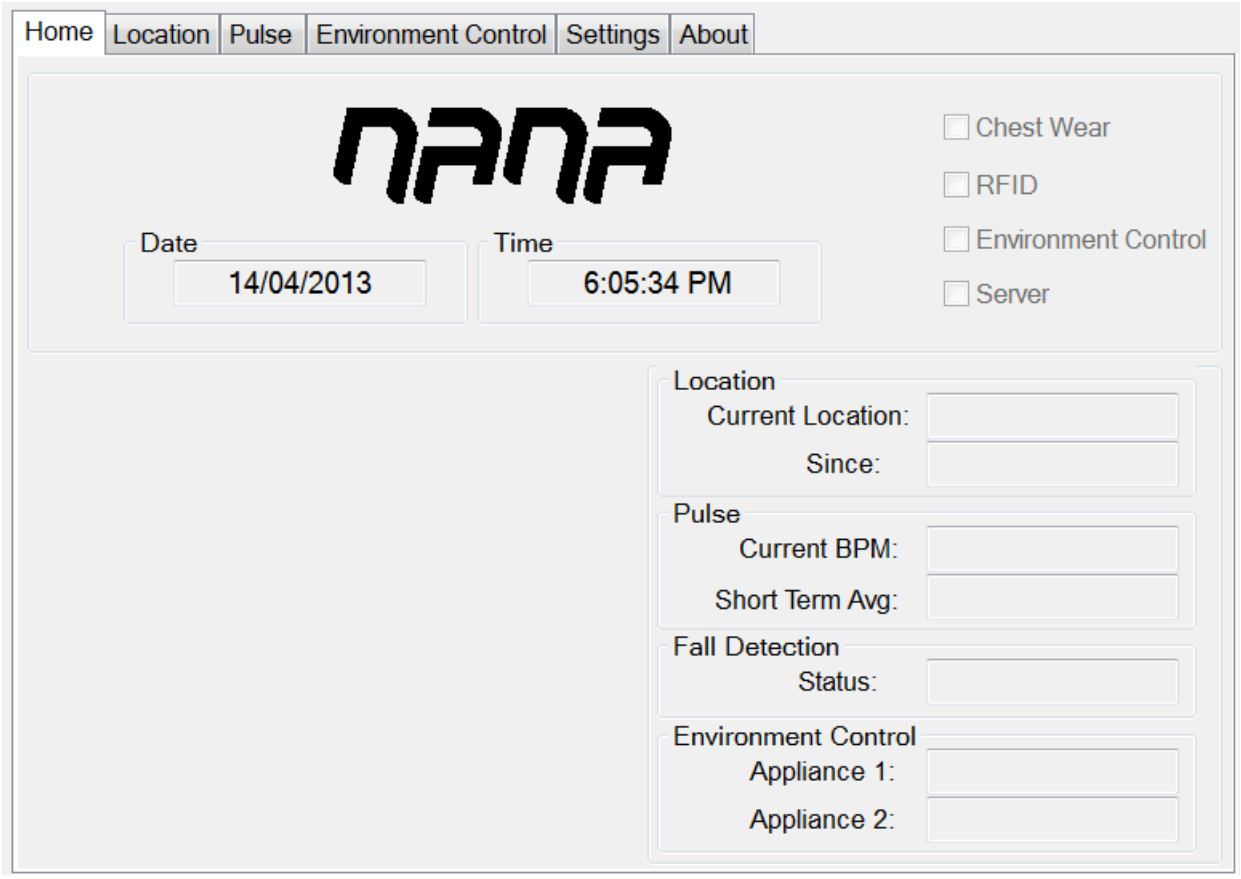
Input Response	Expected Response	Output Response	Pass/Fail
mySensVals[0]	x_temp	x_temp	Pass
mySensVals[1]	y_temp	y_temp	Pass
mySensVals[2]	z_temp	z_temp	Pass
buttonStat	b_temp	b_temp	Pass
heartRate	h_temp	h_temp	Pass

```
//---------------------------------------------------------------------
// Relay Code
//---------------------------------------------------------------------

#define relayOne 7   //relay connector 1
#define relayTwo 9  // relay connector 2

int relayPinTwo = 0;

long relayTwoTimer=0;
int incomingByte = 0;
int dataInbound = 0;


void setup()
{
Serial.begin(9600);
pinMode(relayOne, OUTPUT);
pinMode(relayTwo, OUTPUT);

digitalWrite(relayOne, HIGH); //turns the relay1 off initially
digitalWrite(relayTwo, HIGH); //turns the relay2 off initially
}

void loop()
{
  // if (Serial.available() > 0) {
  // read the incoming byte:
  //   getBTData(); }

  //the above is for Serial Communication

     getBTData();
     updateRelays();
     actuallyUpdateRelays();
     delay(1000);
}

void getBTData() {

        // read the incoming byte:
        //incomingByte = Serial.read();
        incomingByte = 6;
        dataInbound = incomingByte;
        //A simple packet to be sent to this microcontroller
}

void updateRelays() { // for updating the relay...if no command or task is given keep at the relay
       if (dataInbound==0) {   //Do nothing
```

```
                delay(1000);}
        else if (dataInbound==1){
                digitalWrite(relayOne, LOW);
                delay(1000); } //Enable relay one
        else if (dataInbound==2) {
                digitalWrite(relayOne, HIGH); //off
                delay(1000);}
        else if (dataInbound==3) {
                digitalWrite(relayTwo, LOW); //on
                relayPinTwo = 0;
                delay(1000);}
        else if (dataInbound==4) {
                digitalWrite(relayTwo, HIGH);  //off
                relayPinTwo = 1;
                delay(1000);}
        else if (dataInbound==5){ //Both off for emergnecy situations
                 digitalWrite(relayOne, HIGH); //off
                 digitalWrite(relayTwo, HIGH);
                  relayPinTwo = 1;
                  delay(1000);}
         else if (dataInbound==6){ //Both on
                  digitalWrite(relayOne, LOW); //on
                  digitalWrite(relayTwo, LOW);
                   relayPinTwo = 0;
                   delay(1000);

        }
}
void actuallyUpdateRelays()
{
        if (relayPinTwo == 0){
               //update relay
               relayTwoTimer++;
               if (relayTwoTimer>=5) { //this shuts off both appliances after 10 seconds of
function...later design, will shut off
                       digitalWrite(relayTwo, HIGH);
                       relayTwoTimer=0;
                         //dataInbound=0;
                 }
              }
         else if (relayPinTwo == 1)
        {
             relayTwoTimer=0;
        }
}
```

4.3 Interface

The interface is displaying the data from the serial communication. The input is correctly determined and the output is displayed on the screen. The data from the serial communication should have a format of A#H###B#.



4.3.1 Unit Testing

4.3.1.1 Individual Software Testing

4.3.1.1.1 Black Box Specifications – Expected Conditions

```
//---------------------------------------------------------------------
// Input Parsing Function
//---------------------------------------------------------------------
        private void InputCW(object sender, EventArgs e)
        {
            string acc_temp, b_temp, h_temp;
            acc_temp = ""; b_temp = ""; h_temp = "";

            if (input_cw.Length > 7)
            {
                acc_temp = input_cw.Substring(1, ((input_cw.IndexOf('H') - 1)));
                input_cw = input_cw.Substring(2);

                h_temp = input_cw.Substring(1, ((input_cw.IndexOf('B') - 1)));
                input_cw = input_cw.Substring(input_cw.IndexOf('B'));

                b_temp = input_cw.Substring(1, 1);
                input_cw = input_cw.Substring(2);

                StoreCW(Convert.ToInt32(acc_temp), Convert.ToInt32(h_temp) + med,
Convert.ToInt32(b_temp));
            }
        }
```

Expected Conditions through InputCW()

Input Response	Expected Response	Output Response	Pass/Fail

input_cw.Length > 7	acc_temp indentified h_temp indentified b_temp indentified	accelero stored heart_rate stored emergency_button stored	Pass
input_cw.Length < 7	No input indentified	No change	Pass

```
//---------------------------------------------------------------------
// Get Average Function (Long Terms)
//---------------------------------------------------------------------
        private int GetAvg()
        {
            int sum = 0;
            for (int i = 0; i < (main_counter + 1); i++)
            {
                sum = sum + heart_rate[i];
            }
            setPointH = Convert.ToInt32(1.4 * avg_heart_rate);
            setPointL = Convert.ToInt32(0.75 * avg_heart_rate);
            avg_heart_rate = sum / (main_counter + 1);
            return avg_heart_rate;
        }
```

Expected Conditions through GetAvg()

Input Response	Expected Response	Output Response	Pass/Fail
Zero	Zero	avg_heart_rate = 0	Pass
Normal Heart Rate	Total heart rate over number of readings (main_counter)	Successful	Pass

```
//---------------------------------------------------------------------
// Get Current Average Function (Short Term)
//---------------------------------------------------------------------
        private int GetCurAvg()
        {
            int curAvg;
            int sum = 0;
            if (main_counter >= (n - 1))
            {
                for (int i = main_counter - n - 1; i <= main_counter; i++)
                {
                    sum = sum + heart_rate[i];
                }
                curAvg = sum / n;
                return curAvg;
            }
            else return GetAvg();
        }
```

Expected Conditions through GetCurAvg()

Input Response	Expected Response	Output Response	Pass/Fail
Zero	Zero	avg_heart_rate = 0	Pass
Normal Heart Rate	Total heart rate over n number of readings	Successful	Pass

```
//---------------------------------------------------------------------
// Heart Process Function
//---------------------------------------------------------------------
        private bool HeartProcess()
        {
            if (curr_avg_heart_rate > setPointH || curr_avg_heart_rate < setPointL) return true;
            else return false;
        }
```

Expected Conditions through HeartProcess()

Input Response	Expected Response	Output Response	Pass/Fail
curr_avg_heart_rate < setPointH	False when short term average < 1.4 x long term average	FALSE	Pass
curr_avg_heart_rate > setPointL	False when short term average > 0.6 x long term average	FALSE	Pass
curr_avg_heart_rate < setPointH	True when short term average > 1.4 x long term average	TRUE	Pass
curr_avg_heart_rate < setPointH	True when short term average < 0.6 x long term average	TRUE	Pass

```
//--------------------------------------------------------------------
// CW Process Function
//--------------------------------------------------------------------
        private void CWProcess()
        {
            if (HeartProcess() == true && emergency == false)
            {
                emergency = true;
                Alert(true, "Heart Alert", DateTime.Now.ToString("HH:mm:ss tt"));
            }

            if (accelero[main_counter] == 2 && emergency == false)
            {
                emergency = true;
                Alert(true, "Fall Detection", DateTime.Now.ToString("HH:mm:ss tt"));
            }

            if (emergency_button[main_counter] == 1 && emergency == false)
            {
                emergency = true;
                Alert(true, "Emergency Button", DateTime.Now.ToString("HH:mm:ss tt"));
            }
        }
```

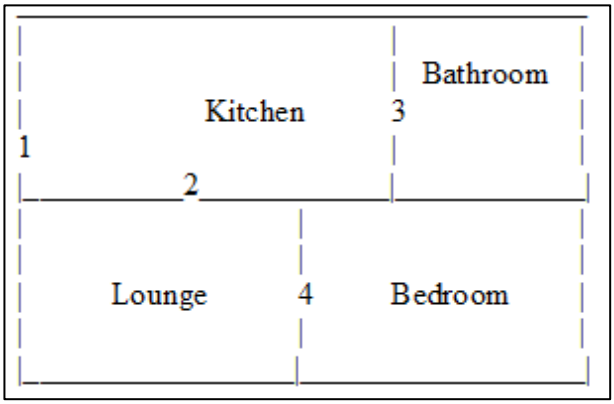
Expected Conditions through CWProcess()

Input Response	Expected Response	Output Response	Pass/Fail
HeartProcess() == true && false	Emergency singal	Heart Alert	Pass
accelero[main_counter] == 2 && emergency == false	Emergency signal	Fall Detection	Pass
Emergency_button[main_counter] == 1 && emergency == false	Emergency signal	Emergency button	Pass
emergency == true	No emergency signal, only one emegency signal at a time	No change	Pass

4.4 RFID

In order for the system to properly read RFID tags, the RFID sensor must register that a tag has passed through it. The base station of the system must then register that a tag has passed through it, and perform an action based on which tag has passed through the sensor.

The system currently is reading RFID tags that are scanned. The purpose of the RFID system is to create a non-invasive in-home tracking system that will provide the room that the user is currently located in.



The above diagram is a general layout of our example house. The numbers are doorways, and ideally we would have an RFID reader in each doorway, however, we will be using an RFID tag to represent each individual doorway.

4.4.1 Unit Testing

4.4.1.1 Individual Hardware Testing

4.4.1.1.1 Black Box Specifications – Expected Conditions

The RFID reader must be able to scan an RFID tag, and relay the information contained on the tag to the base station, where it will be identified as one of our individual tags.

Expected Inputs from RFID Reader

Input Response	Expected Response	Output Response	Pass/Fail
Tag #1 scanned	Tag #1 identified	CurDoor= 1	Pass
Tag #2 scanned	Tag #2 identified	CurDoor= 2	Pass
Tag #3 scanned	Tag #3 identified	CurDoor= 3	Pass
Tag #4 scanned	Tag #4 identified	CurDoor= 4	Pass
No tag scanned	No tag identified	No change	Pass
Unknown tag scanned	Tag identified, but not recognized, no update	No change	Pass

4.4.1.2 Individual Software Testing

4.4.1.2.1 Black Box Specifications – Expected Conditions

RFID sensing requires us to register the register the scanned RFID tag, and update the location of the user based on the users current location within the house.

```
//---------------------------------------------------------------------
// Update Location Function
//---------------------------------------------------------------------
void updatePosition(){
	if (curDoor==1)
		if (curLoc==kitchen)
			curLoc=outside
		else if (curLoc==outside)
			curLoc=kitchen
	if (curDoor==2)
		if (curLoc==kitchen)
			curLoc=lounge
		else if (curLoc==lounge)
			curLoc=kitchen
	if (curDoor==3)
		if (curLoc==kitchen)
			curLoc=bathroom
```

```
                else if (curLoc==bathroom)
                        curLoc=kitchen
        if (curDoor==4)
                if (curLoc==lounge)
                        curLoc=bedroom
                else if (curLoc==bedroom)
                        curLoc=lounge
}
```

Expected Conditions through updateLocation()

Input Response	Expected Response	Output Response	Pass/Fail
CurLoc= ANY curDoor= NONE	No change	No change	Pass
CurLoc= ANY curDoor= UNKNOWN	No change	No change	Pass
CurLoc= kitchen curDoor= 1	curLoc= outside	CurLoc= outside	Pass
CurLoc= kitchen curDoor= 2	CurLoc= lounge	CurLoc= lounge	Pass
CurLoc= kitchen curDoor= 3	CurLoc= bathroom	CurLoc= bathroom	Pass
CurLoc= kitchen curDoor= 4	No change	No change	Pass
curLoc= outside curDoor= 1	CurLoc= kitchen	CurLoc= kitchen	Pass
curLoc= outside curDoor= 2, 3, 4	No change	No change	Pass
curLoc= bathroom curDoor= 3	CurLoc= kitchen	CurLoc= kitchen	Pass
curLoc= bathroom curDoor= 1, 2, 4	No change	No change	Pass
curLoc= lounge curDoor= 2	CurLoc= kitchen	CurLoc= kitchen	Pass
curLoc= lounge curDoor= 4	CurLoc= bedroom	CurLoc= bedroom	Pass
curLoc= lounge curDoor= 1, 3	No change	No change	Pass
curLoc= bedroom curDoor= 4	CurLoc= lounge	CurLoc= lounge	Pass
curLoc= bedroom curDoor= 1, 2, 3	No change	No change	Pass
CurLoc= ANY curDoor= NONE	No change	No change	Pass
CurLoc= ANY curDoor= UNKNOWN	No change	No change	Pass

4.4.1.2.2 White Box Specifications – Path Completeness

For RFID reading, the total number of output responses is 15, obtained from the function updateLocation.

The path of the RFID data is as follows. The sensor receives the tag data from the RFID tag. Then, this data is sent to the base station through the USB cable. The program then reads the data from the tag, and identifies the tag as one of the four individual tags. Once it has done this, the program recognizes that the user has passed through one of the four doors. Finally, based on the users current position within the house, and the current door they are passing through, the program updates their location.

5. Integration Testing

Integration testing was done with various parts of our system before putting it altogether.

Parts Used	Expected Response	Output Response	Pass/Fail
-System is on for 5min -Heart Data is sent -Accelerometer Data is sent -Button is pushed -RFID is excluded	The system should be active and all the sensors reading data correctly. Except RFID	Successful, data was read correctly	Pass
-System is on for 5min -Heart Data is sent -Accelerometer Data is sent -Button is not pushed -RFID is used	The system should be active and all the sensors reading data correctly. Button not read.	Successful, data was read correctly	Pass
-System is on for 5min -Heart Data is excluded -Accelerometer Data is sent -Button is pushed -RFID is not used	The system should be active and all the sensors reading data correctly. Heart data excluded.	Successful, data was read correctly	Pass
-System is on for 5min -Heart Data is sent -Accelerometer Data is sent -Button is not pushed -RFID is not used	The system should be active and all the sensors reading data correctly. RFID and Button not read.	Successful, data was read correctly	Pass
-System is on for 5min -Heart Data is sent -Accelerometer Data is excluded -Button is pushed -RFID is used	The system should be active and all the sensors reading data correctly. No Accelerometer data.	Successful, data was read correctly	Pass
-Voice Recognition -"Lights ON"	The system should be active and all the sensors reading data correctly. Should turn on the lamp	Successful, data was read correctly. Lamp was turned on	Pass
-Voice Recognition -"Lights OFF"	The system should be active and all the sensors reading data correctly. Should turn off the lamp	Successful, data was read correctly. Lamp was turned off	Pass
-Voice Recognition -"Kettle ON"	The system should be active and all the sensors reading data correctly. Should turn on the kettle	Successful, data was read correctly. Kettle was turned on	Pass
-Voice Recognition -"Kettle OFF"	The system should be active and all the sensors reading data correctly. Should turn off the Kettle	Successful, data was read correctly. Kettle was turned off	Pass

6. System Testing

6.1 Facility Testing – does the system provide all the required functions

Our system requires several functions in order to be successful. These functions are heart rate sensing, accelerometer sensing, emergency button sensing, wireless communication, in-home location tracking, and a software interface that interprets the data received from the different sensors. All of these functions are currently incorporated into our system, and all of them, with the exception of wireless communication are functional at the writing of this report.

Expected conditions and testing of each of these different functions are included above in the report.

6.2 Endurance Testing – will the system continue to work for long periods

Endurance testing that occurred with the prototype#1 was testing the system for lengths of periods. The purpose of this test is to ensure it can run smoothly for long lengths without any interruptions.

Input Response	Expected Response	Output Response	Pass/Fail
System is on for 5min	The system should be active and all the sensors reading data correctly.	Successful, data was read correctly	Pass
System is on for 20min	The system should be active and all the sensors reading data correctly.	Successful, data was read correctly	Pass
System is on for 40min	The system should be active and all the sensors reading data correctly.	Successful, data was read correctly	Pass
System is on for 2Hours	The system should be active and all the sensors reading data correctly.	Successful, data was read correctly	Pass

6.3 Usability Testing – can the user use the system easily

From our first prototype, the user only uses an emergency button to make an emergency. This system is designed to make it very accessible and easy to use for anyone.

Input Response	Expected Response	Output Response	Pass/Fail
Button User Response	The ALERT in the interface signals emergency.	Successful, button was read correctly	Pass

6.4 Performance Testing – how good is the response time

From our first prototype, its response time is quite fast. We measured data gathering to be at 0.8sec (all sensors) which would then be sent through serial communication to the interface. Also, on the interface side, if there was any RFID tag used, the result of the reader picking up a response was instant.

Input Response	Expected Response	Output Response	Pass/Fail
Hardware Sending	The system should sent data in less than a second	Successful, data was read correctly	Pass
RFID Tag Testing	The system should be instant with tag reading	Successful, system detects the tag instantly	Pass

6.5 Recovery Testing – how well does the system recover from failure

The first prototype does not have a backup feature if failure were to occur. However, if our system were to give inaccurate results, we would reset the Ardunio and the functionality would be returned to its original form.

7. Validation

Goals	Explanation	Pass/Fail
Easy to Use	The system does not require user input unless a emergency situation occurs. The system has one button and is discretely hidden on the chest.	Pass
Reliable and Robust	The product is robust as we tested falls from 4 feet above ground. The casing was able to protect our system from shock.	Pass
Affordable	The prototype is under our outlined budget of $1000. Even with buying many replacement parts.	Pass
Safety	The system is external and safe to use. It is encased in a plastic container and there is no chance of accidental shock.	Pass
Effective	The product alerts the monitoring station within 30 seconds of a potential emergency situation. This is the same as we outlined in our device time constraints.	Pass
Security	The system is secure because it uses Bluetooth set on invisible. Unauthorized users will not know how to pairing name with it and will have to guess the pairing code even if they do.	Pass
Privacy, Legal and Ethics	The system maintains privacy in our location system by using RFID tags to identify the general area that the user is in, instead of giving an exact location.	Pass

8. Contribution

Nicholas	• Completed section 2.0 and 3.2 • Helped arrange meetings for Sept-Feb • Completed editing of Version 1 System Design
Josh	• Completed section 4.4 • Completed editing of Version1 V & V Document
Abhishek	• Completed section 4.2 • Completed editing of Version1 V & V Document
Junaid	• Completed section 1 through 1.6, 3.3 and 3.4 • Initialized document template • Proofreading and revisions for V & V Document
Azzam	• Completed section 5.0 and 4.3 • Completed editing of Version 1 V & V Document
Ahmad	• Completed section 4.3 • Completed editing of Version 1 V & V Document

9. Log Progress

Date	Event
Sept 29	Thode B103 • Uploaded expertise to the Google Docs page • Met with the group to discuss goals • Brainstorming of ideas
Oct 8	**Goals Report Due**
Oct 11	Thode B104 • Discussion of requirements • Started to research materials needed for the project
Oct 15	Thode B107 • Research on existing technologies as a guideline • Researched some materials to use (microcontrollers, sensors, computer software) • Researched some case studies regarding products out there (the behavior, health concerns, common mistakes)
Oct 18	Thode 204 • Discussed issues with Goals and kept moving with the requirements
Oct 23	Thode B106 • Discussion of Use Cases for the Product • Preparation for the draft system requirements
Oct 25	Thode B104 • Discussion of final requirements and the uses cases for the product
Oct 25	**Draft Systems Requirement Due**
Nov 1	Thode B103 • Set up hardware/software page on Google Drive to start shopping list of materials
Nov 6	Thode B106 • Looked into what hardware we needed and what software to use • Discussed any communication protocols outside of the black box • Discussed details of how the systems talk to each other(hardware and software)
Nov 9	Parts are assigned to everyone for Draft System Design
Nov 12	Tron Lab • Completed loose ends to Draft System Design
Nov 15	**Draft System Design Due**
Nov 27	B107 – completing the datasheets, and editing the draft system from TA comments. Working on draft system requirements.
Dec 4	B107 – compiling the report with all the potential materials to use and trying to see if the system would work
Dec 10	**System Components Requirements Due**
Jan 8	Thode Library – disscussing on new major change in design, and working on system component design
Jan 15	Thode Library – working on system component design

Jan 18	**System Component Design Due**
Feb 12	**Implementation Rev 0**
Mar 7	**Final Requirements**
Mar 14	**Final Design**
Apr 4	**Implementation Rev 1**
Apr 12	**Final Verification and Validation**