**F-Robot Project**

Team: Skynet

Team members: Mohammed Hassnain, Erik Lienhard

Email: [mohammed-17@csu.fullerton.edu](mailto:mohammed-17@csu.fullerton.edu), [elienhard@csu.fullerton.edu](mailto:elienhard@csu.fullerton.edu)

Date: May 3, 2017

Course: CPSC 362

Instructor: Dr. Chang-Hyun Jo

**Revision History**

|  |  |
| --- | --- |
| **Team members:** | **Work products/completion date:** |

|  |  |
| --- | --- |
| Mohammed Hassnain | Analysis Class Diagram completed on 3/1/2017  Activity Diagram completed on 2/28/2017  Class responsibility collaborator completed on 2/22/2017  Use Cases completed on 2/10/2017  Functional & Non-Functional Requirements completed on 2/8/2017  2 Interaction Diagram completed on 4/1/2017  Activity Diagram completed on 4/28/2017  State Diagram completed on 4/29/2017  Swim-Lane Diagram completed on 4/10/2017  Preliminary Screen Layout completed on 4/8/2017  Map-User-Objectives completed on 4/8/2017  Design Pyramid completed on 4/8/2017  Content Architecture completed on 4/9/2017  MVC Architecture completed on 4/10/2017  Navigation Semantics Units completed on 4/11/2017  Use of Case Tools completed on 4/29/2017  Communication report, Planning report completed on 5/2/2017 |
| Erik Lienhard | Analysis Sequence Diagram completed on 3/1/2017  Use Cases completed on 2/10/2017  Functional & Non-Functional Requirements completed on 2/8/2017  State Diagram completed on 2/28/2017  Swim Lane Diagram completed on 3/1/2017  Use Case Diagram completed on 2/15/2017  Discussion completed on 4/1/2017  Architectural Context Diagram completed on 4/10/2017  Overall Architectural Structure completed on 4/8/2017  Design Class Diagram completed on 4/28/2017  Refined Architecture Structure Diagram completed on 4/1/2017  Design Problem completed on 4/15/2017  Pattern-Organizing table completed on 4/15/2017  Best Practices- 10 bullets list completed on 4/20/2017  Mobility Environments completed on 4/21/2017  Construction: Code and Test completed on 4/22/2017  Traceability Matrix & Diagram completed on 4/28/2017 |

Table of Contents

Revision History........................................................page: 2

Table of Contents......................................................page: 3

Waterfall Model…………………………………………………..…page: 4

Traceability Matrix & Diagram……………………………..…page: 5-6

Communication……………………………………………………...page: 7

Planning………………………………………………………………..…page: 8

Functional & Non-Functional Requirements…………….page: 9

Use Cases………………………………………………………………...page: 10

UC Diagram ………………………………………………………….….page: 11

Analysis Class Diagram………………………………………….….page: 12

Class Responsibility Collaborator……………………………...page: 13

Activity Diagram ……………………………………………………...page: 14

Swim-Lane Diagram………………………………………………….page: 15

State Diagram…………………………………………………………..page: 16

Analysis Sequence Diagram………………………………………page: 17

Architectural Design Rationale....................................page: 18

Architectural Context Diagram……………………………….page: 19

Overall Architectural Structure………………………………...page: 20

Design Class Diagram L2…………………………………………..page: 21

Design Class Diagram………………………………………………..page: 22

2 Interaction Diagrams……………………………………………..page: 23

Refined Architecture Structure Diagram…….…………….page: 24

Activity Diagram……………………………………………………….page: 25

State Chart……………………………………………………….……….page: 26

Swim-Lane Diagram………………………………………….……...page: 27

Preliminary Screen Layout………………………………….…….page: 28

Map-User-Objectives…………………………………………...…..page: 29

Design Problem…………………………………………………….....page: 30

Pattern Organizing Table………………………………………...page: 31

Design Pyramid for Web-Apps…………..…………….……….page: 32

Content Architecture…………………………………….………...page: 33-34

MVC Architecture………………………………………….………..page: 35

Navigation Semantics Unit………………..…………..…….….page: 36

Best Practices……………………………………..…………..………page: 37-38

Mobile Environments…………………………..………..……....page: 39

Construction: Code & Test……………………..……..………..page: 40-45

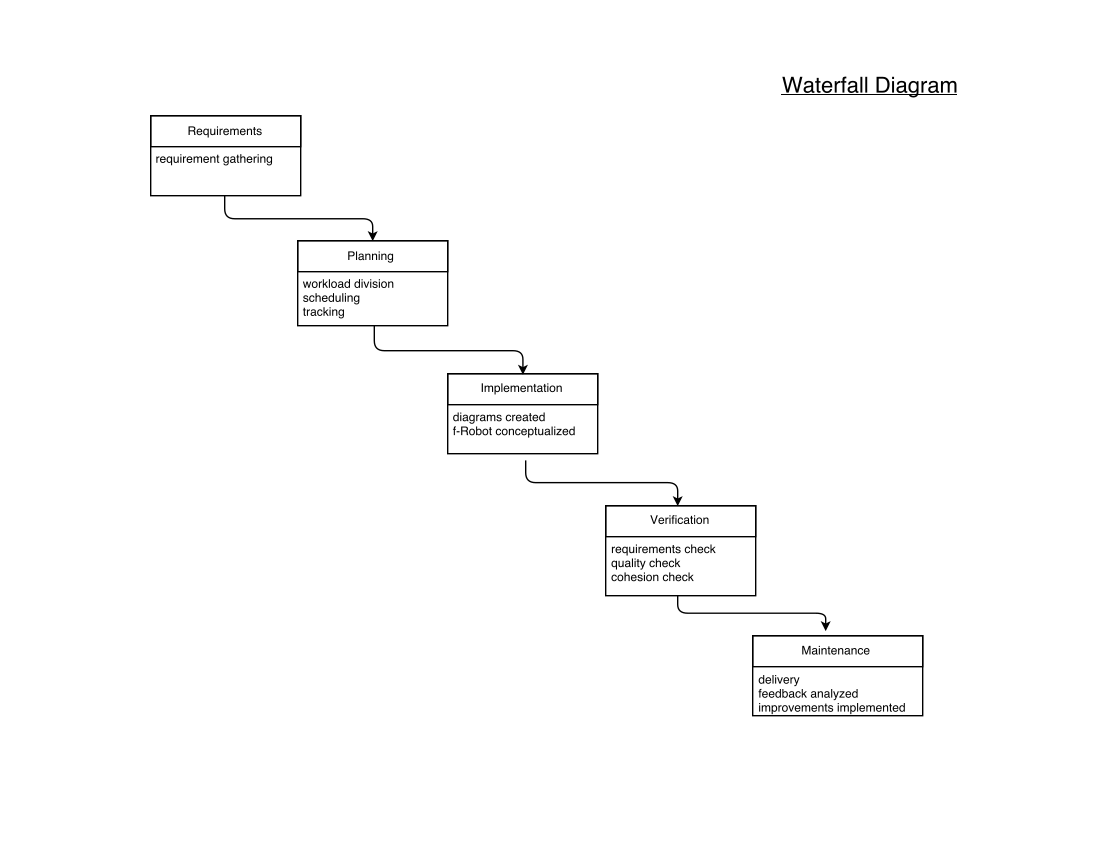
Case Tools....................................................................page: 46

References………………………………………………..…...………..page: 47

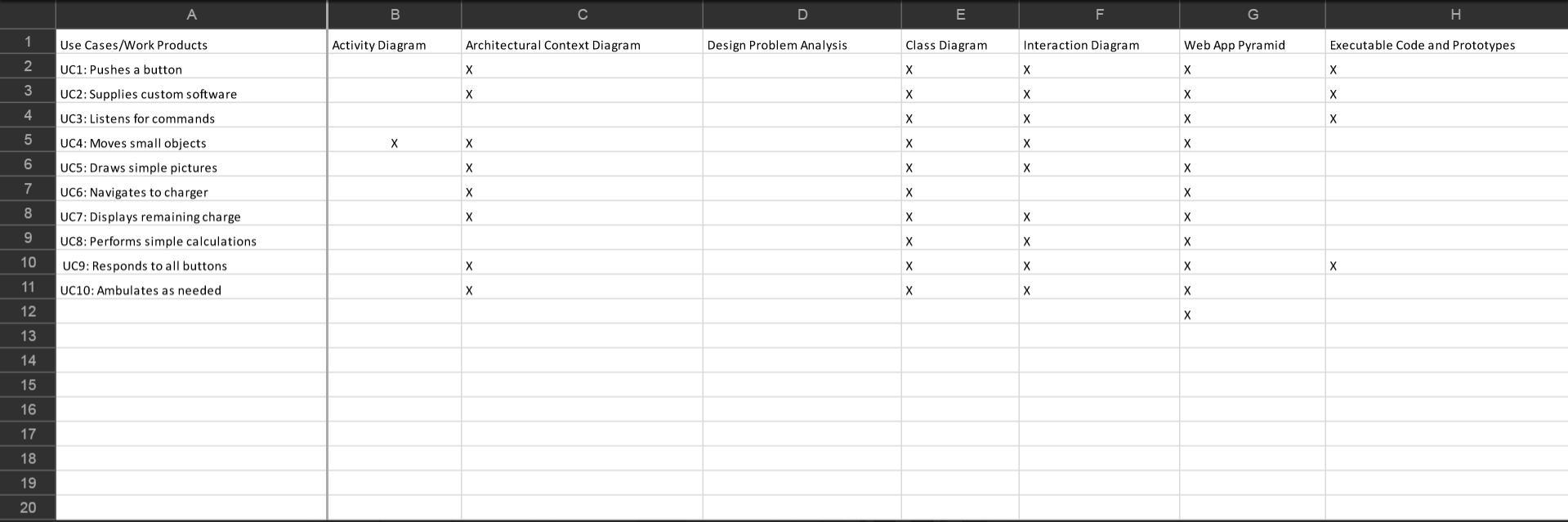
Team Evaluation…………………………………………..…………..page: 48

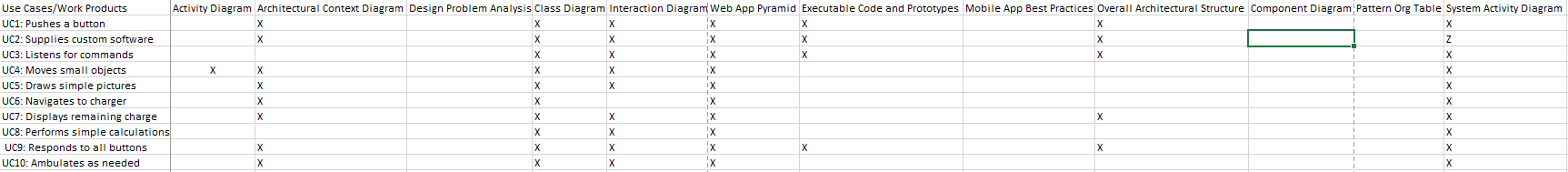
Team Charter........………………………………………..…………..page: 49-51

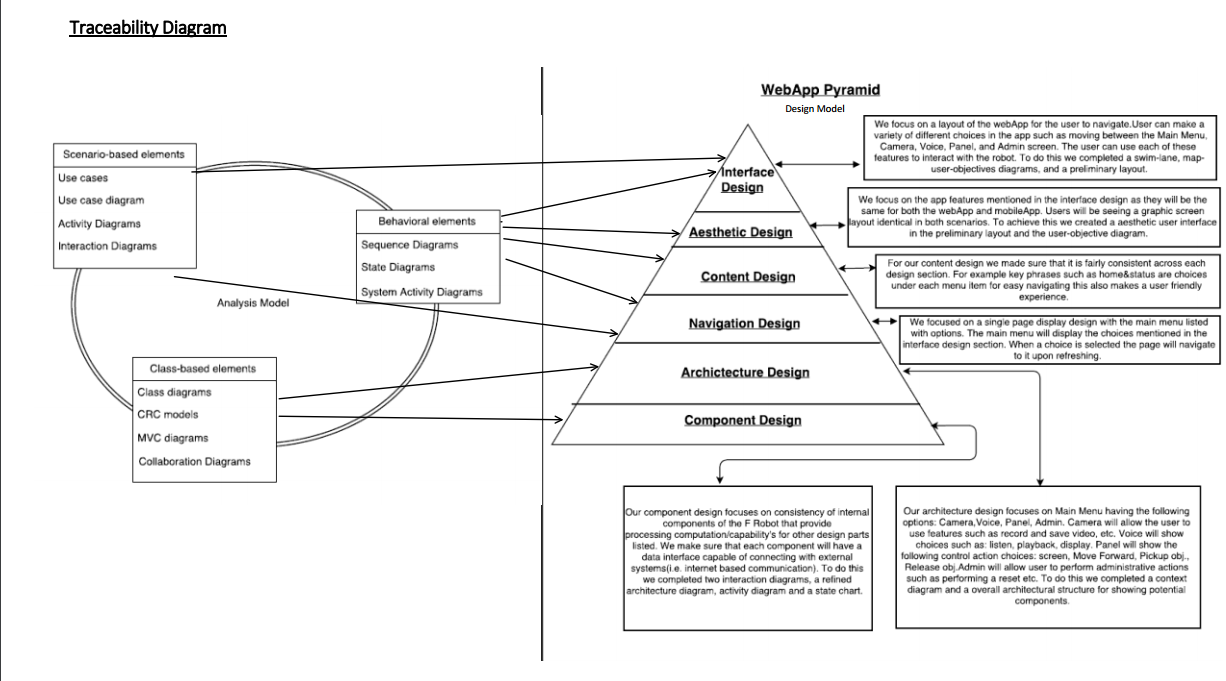
Waterfall Model

****

**Traceability Matrix**







**Communication**

Project Initiation:

Our application is a combination of work products to build a robot system called the Fullerton Robot or for shorthand F-Robot. The robot system will be built to automatically behave as preprogrammed. It will have several features including: being able to speak a few words/statements and show test on a display screen, listen to a voice command, move around a designated area using sensors and a camera, and can hold/release objects. The user will also be able to provide custom software to the robot through USB/wireless transmission for reprogramming features such as programmable buttons.

Requirements Gathering:

Our team has continued this project by adding the design portion of the requirements which will align with the analysis requirements that were necessary for the F-Robot to be able to complete the task features listed in the project initiation. We created an architectural context diagram and overall architectural structure to essentially show the target system with all of its external systems. It also shows the potential components inside of the system and their interactions. Using this we created various models for the component level, user interface, pattern-based, web-app, and mobile-app design. Using this information, we developed a detailed requirement design models that details the progress of developing each step of the design process for the F-robot.

**Planning**

|  |  |
| --- | --- |
| **Products:** | **Estimate:** |
| Design requirements cost | $500 |
| Software application cost | $3,000 |
| Robot system parts:  Camera, sensors, wheels, body structure etc. | $500 |
| Manual labor production cost | $100 |
| Shipping cost | $50 |
| **Total cost** | **$4,150** |

|  |  |
| --- | --- |
| **Tracking:** | **Scheduling:** |
| Software application process:  Developing requirement analysis: FR &NFR, UC, UCD, ACD, CRC, AD, SLD, SD, ASD | 1 month |
| Design process: architectural: ACD,OAS component: CD,ID,RASD,AD,SC  user interface: SLD, PSL,MUO  pattern-based: DP, POT  web app: DPW, CA,MVC, NSU  mobile app: BP,ME  Quality Concepts: Construction code/test | 6 weeks |
| Robot part build production time | 5 weeks |
| **Total time frame** | **15 weeks** |

**Functional Requirements (FRs)**

List the most important 10 FRs.

Start with 10 FRs in the analysis phase.

Prioritize FRs to focus on.

**Functional Requirements**

1. to speak a few words and a few statements (less than 10 words/statements)

2. to be able to receive and integrate custom end-user software either by wireless transmission or USB

3. to interpret voice recognition and to show its text/script/sign-language on the screen

4. to hold/release objects by their hands (less than 10 pounds).

5. to be able to draw on a surface

6. to be able to connect itself to a charging station when its battery is low

7. to show its battery level status

8. to be able to do basic mathematical calculations(add/subtract/divide/multiply) by taking in questions verbally

9. to be able to respond to buttons like power, reset, etc

10. To be able to walk around a room without assistance using a camera

11. ability to display a selected emoticon and play a matching sound

**Non-Functional Requirements (NFRs)**

List the most important 5 NFRs.

Start with at least 5 NFRs, and then finally prioritize/choose/focus on 2 NFRs later.

NFRs include both Quality Attributes and Design Constraints.

**Non-Functional Requirements**

1. Ability to navigate the robot remotely using an app

2. Movement speed- how fast the robot moves

3. Strength - how much weight the robot hold

4. Performance – how long the robot can stay on, how quickly/smoothly it performs actions

5. Listening ability- how well the robot understands voice recognition

6. to be able to remain upright at all times. (robot should not fall over, even while walking or holding stuff)

7. to be able to recover from end-user software that causes some logic failure (robot should continue to work if something goes wrong in its logic)

**Use-Cases**

1. **Pushes a button**-Push voice recognition button to allow the robot to listen for the command. The robot will then interpret the command and react accordingly.

2. **Supplies custom software**-When the robot is given new software by the user, it can incorporate it without causing system instability.

3. **Listen for commands**-Push voice recognition button to allow the robot to listen for the command.

The robot will then interpret the command and show its response on its screen.

4. **Move small objects**-Push voice recognition button to allow the robot to listen for the command.

The robot will interpret the command and react by picking up/releasing the object specified.

If it cannot pick up the object it will display a failure message.

5. **Draw simple picture**-Push voice recognition button to allow the robot to listen for the command.

Then the robot will interpret the command and react by lowering itself and

drawing on the surface at ground-level with the pointer in its hand.

6. **Navigate to charger**-The robot will connect itself to the charging port once its battery is below a certain threshold.

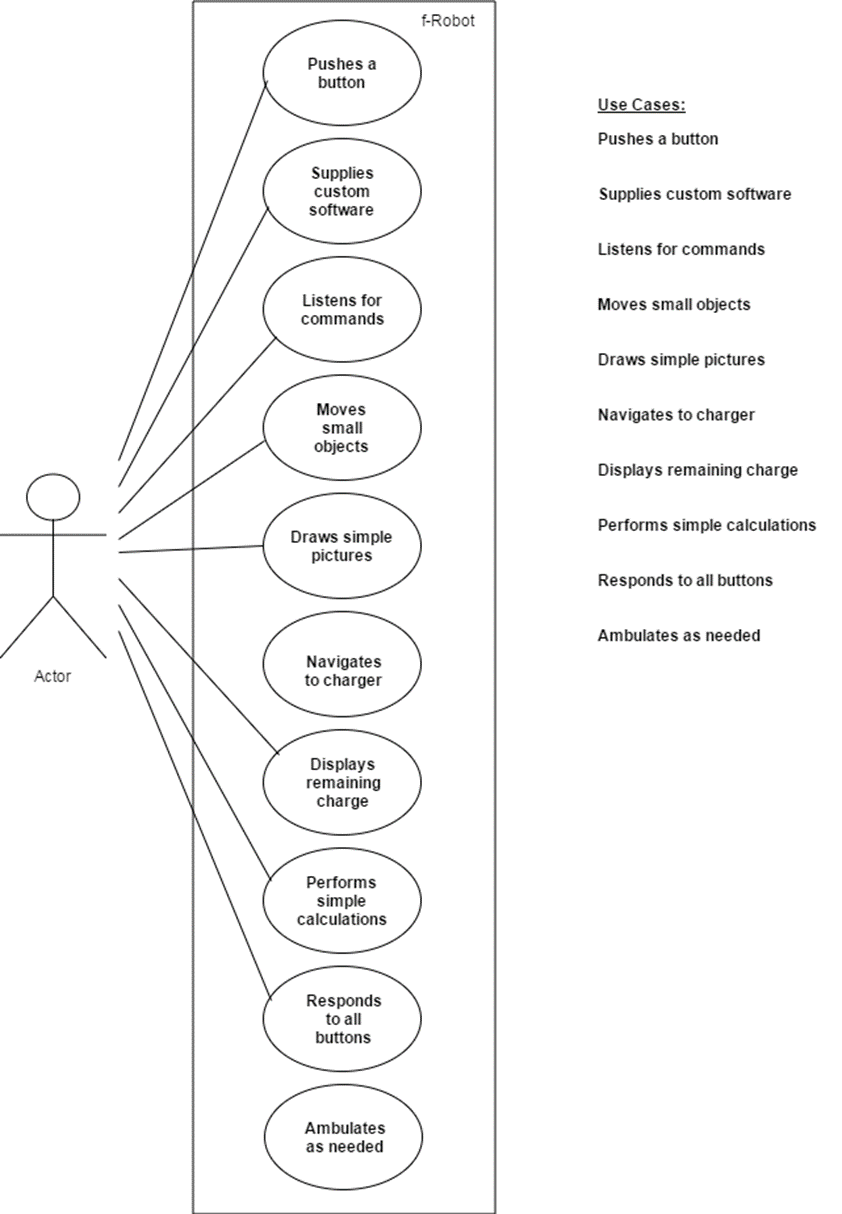
7. **Display remaining charge**-When prompted by the user, the robot will display the battery charge remaining.

8. **Perform simple calculations**-When given verbal instructions by the user to perform some mathematical calculation, the robot will process the result and display it on its screen.

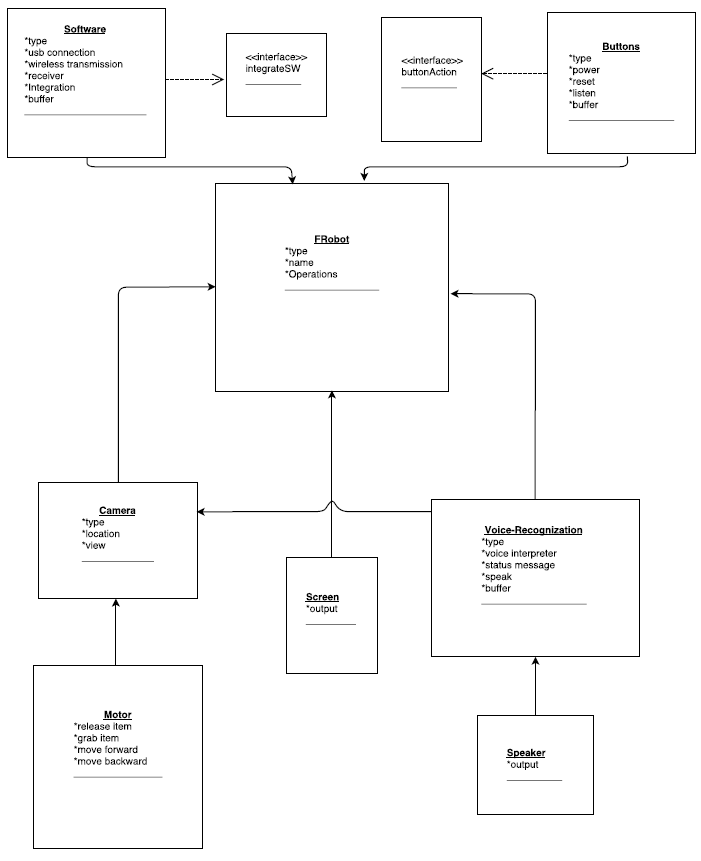
9. **Responds to all buttons**-When a button is pressed, the robot will wait for user input or carry out the instructions

10. **Ambulates as needed**-When performing an action at the command of the user or when charging needs to occur, the robot will ambulate as needed using a camera to perceive its surroundings.

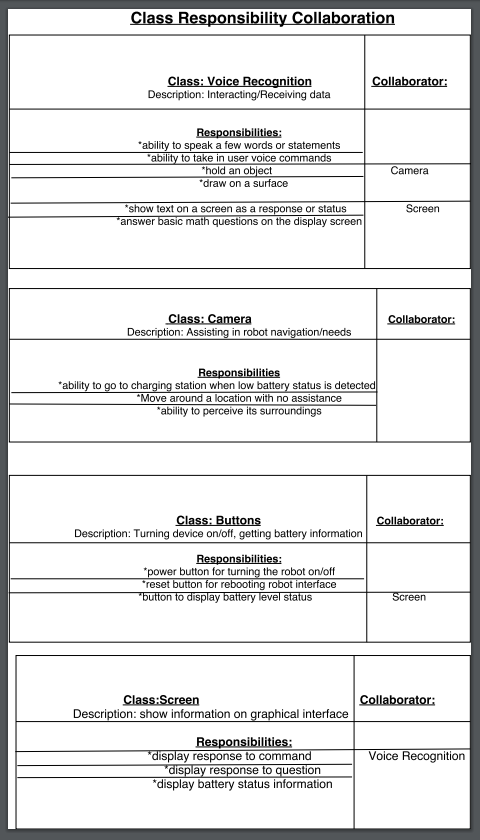
**Use Case Diagram**

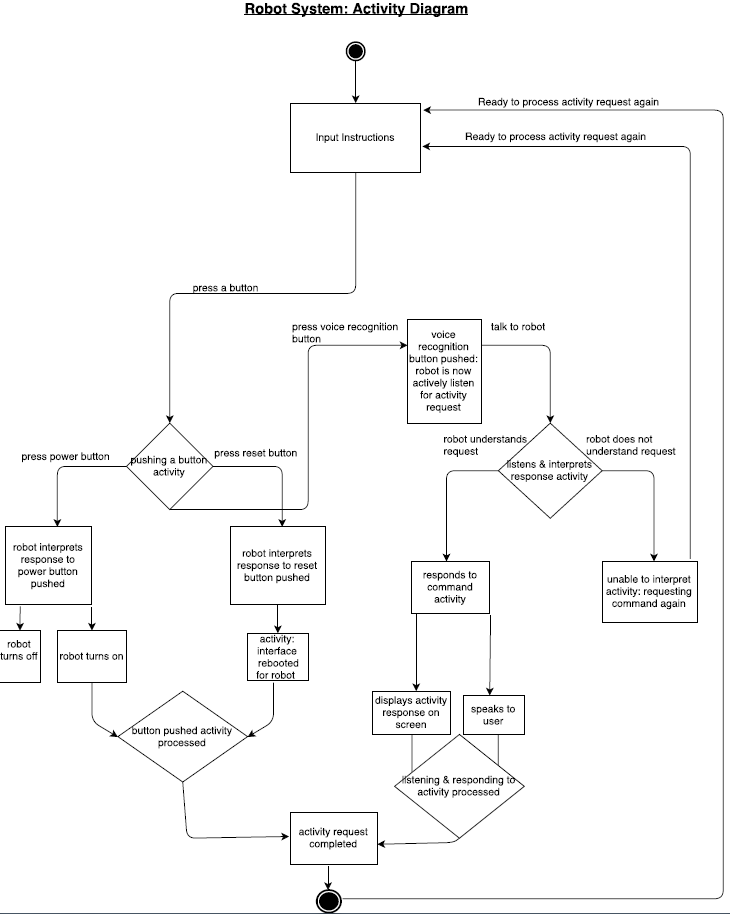


**Analysis Class Diagram**



The **FRobot** class is linked with multiple classes as shown above. The primary classes it is linked with are **camera**, **voice-recognition, screen**, **software**, and **button**. The software class works with regards to **supplying custom software**. The button class works with **pushing a button** on the robot. The camera class works with the robot’s movability and allows the it to **ambulate as needed**. The voice-recognition class enables the user to give it verbal commands when the **listen** button is pressed by the actor. The screen class **displays** various information including **remaining charge** information, responses to **simple calculations**. As shown above each class works with the FRobot class to provide a connected analysis diagram.





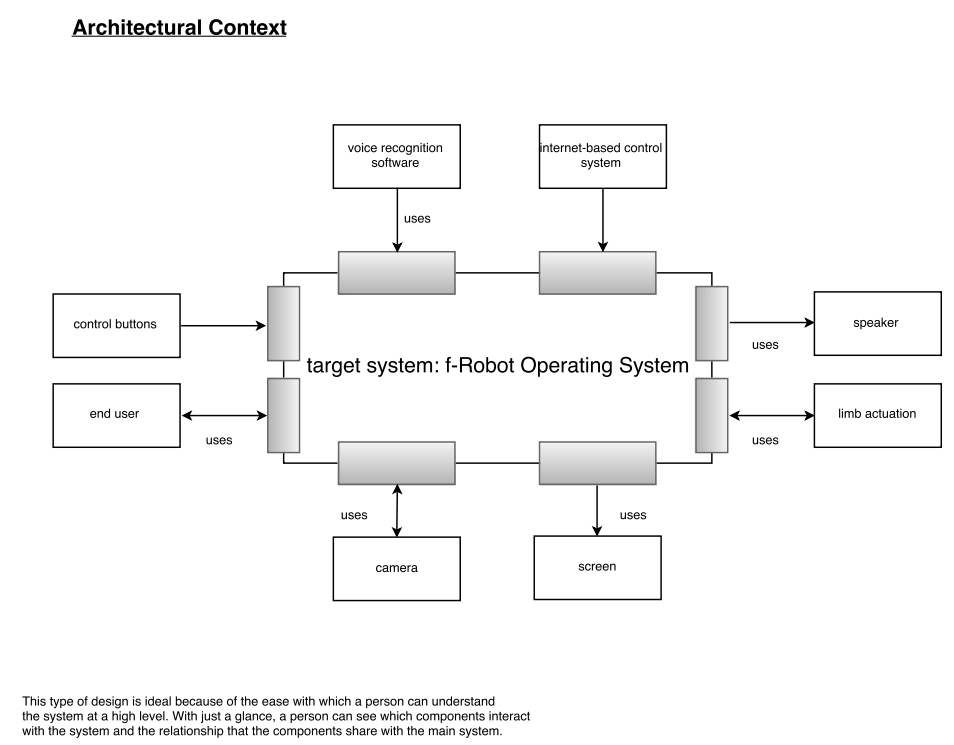
****

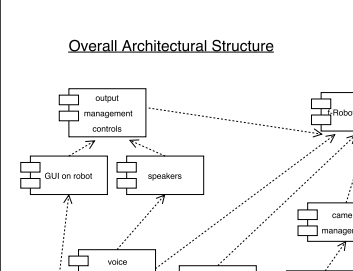
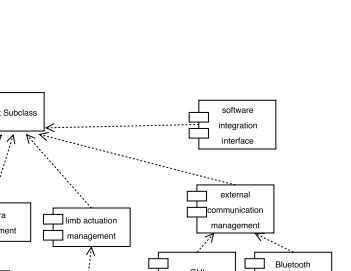
****

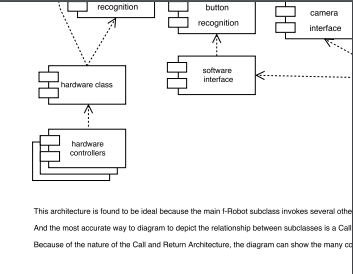
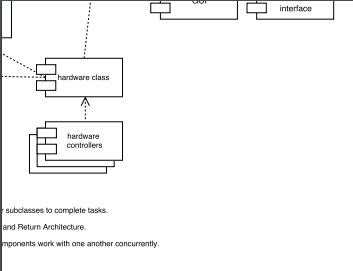
**Analysis Sequence Diagram**

**Architectural Design Rationale**

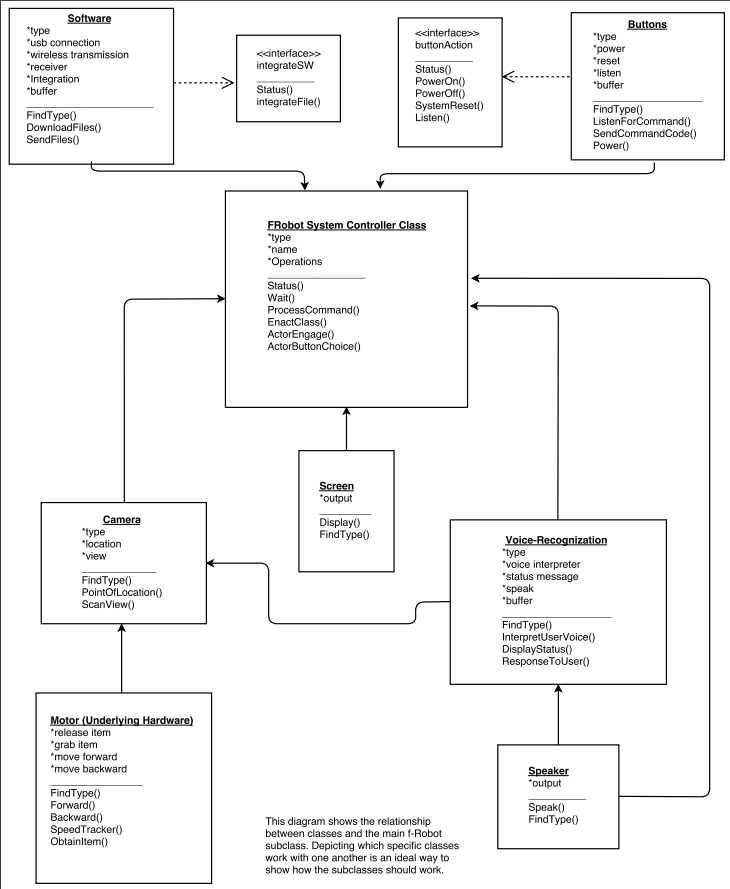
In our project, we considered multiple types of architectural style. From the categories of Data-Centered architectures, Data-Flow Architectures, and Call and Return Architectures, we chose to use a Call and Return Architecture. If one considers the f-Robot as a central resource that is frequently accessed by other components, this type of architectures makes sense. The real strength of Data-Centered Architectures is the integrability. Components can be added or modified without regard to other components. Unfortunately, for the f-Robot, components often need to work together and changing or adding new components requires serious changes to the system. Because of this we looked to a Data-Flow Architecture. Data-Flow Architecture is a good architecture for when data in input and needs to be traced through all transformations until it becomes output. This at first seemed like the ideal architecture for the f-Robot because taking user input and transforming it into visible output is what the goal of the use cases seeks to achieve. However, upon closer inspection, a Data-Flow architecture was not the optimal method to diagram the f-Robot as it follows data sequentially. Sequential data flow is not optimal for the f-Robot needs to do many operations at once. Waiting for data to flow from one filter to another would result in slow responses. This led us to the Call and Return Architecture. This architecture is found to be ideal because the main f-Robot invokes several subclasses to complete tasks. In this way, many components can work with one another concurrently. Because of this, a Call and Return Architecture best suited the needs of the f-Robot project.



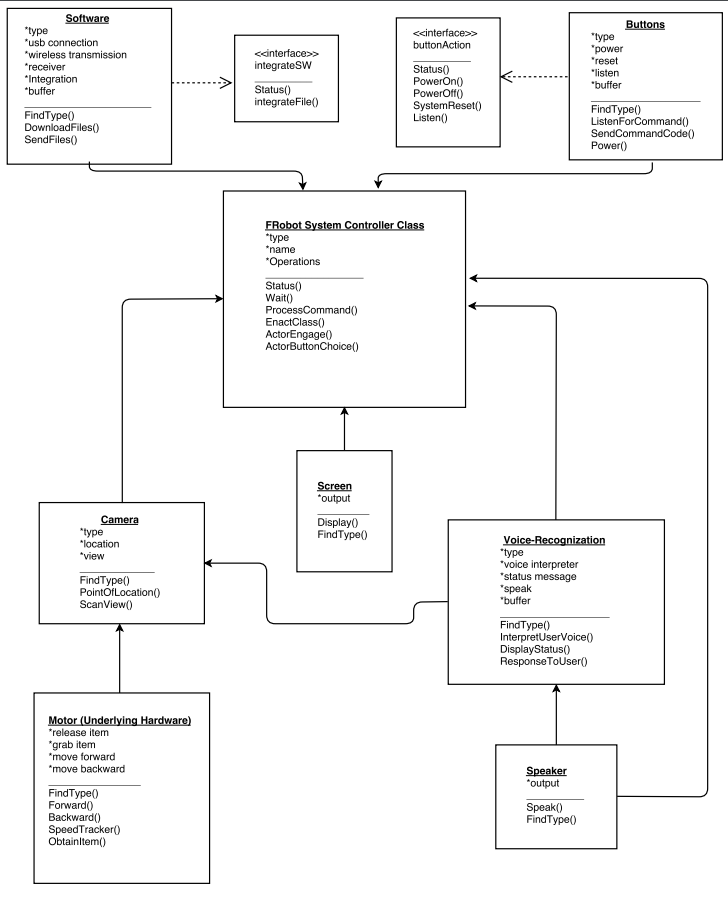
 

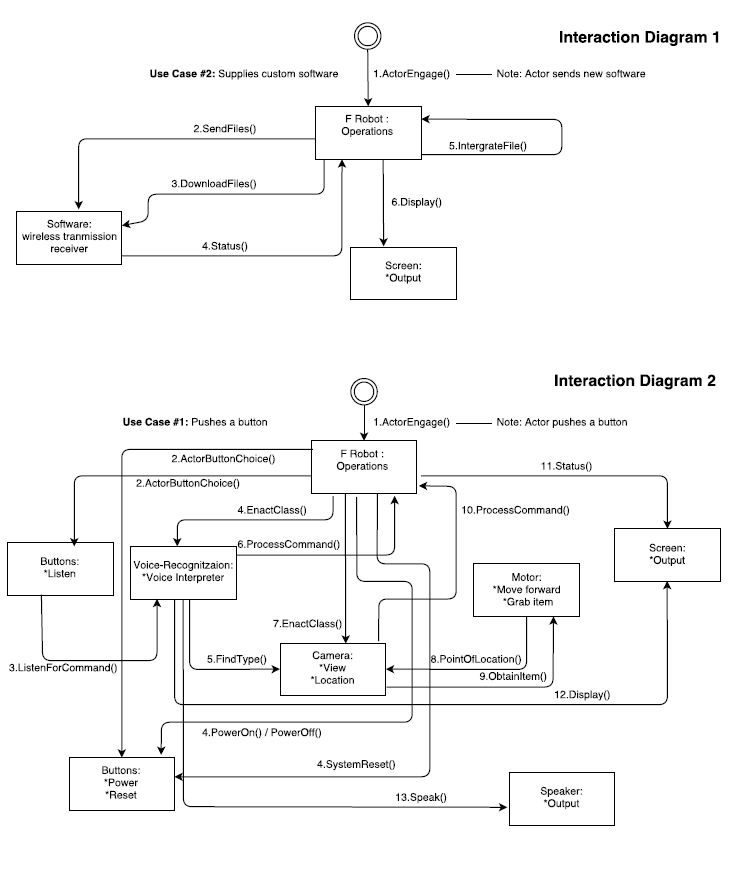
 

**Design Class Diagram L2**

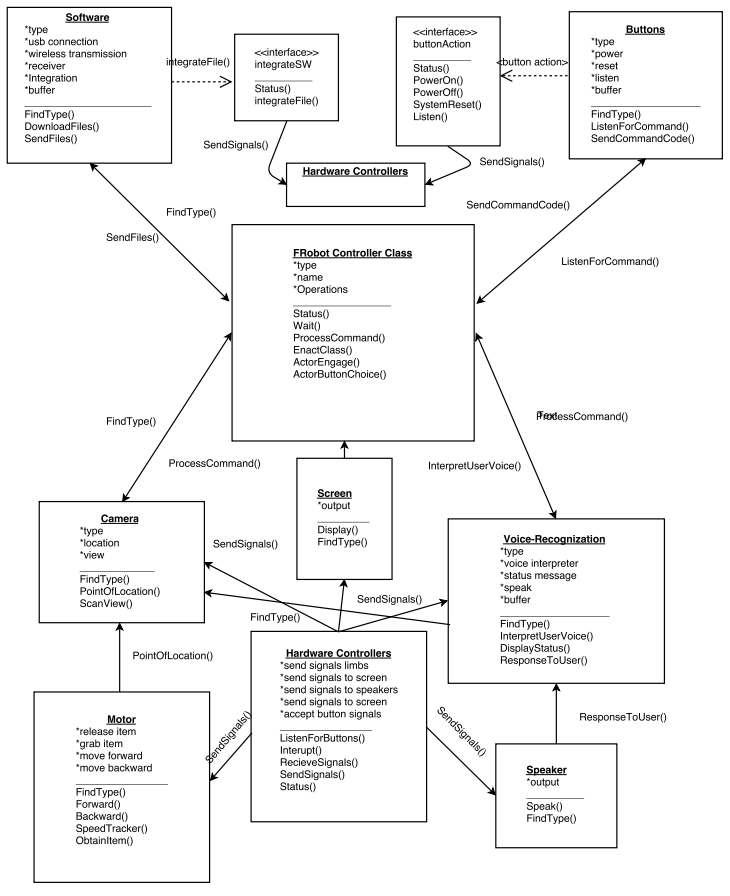


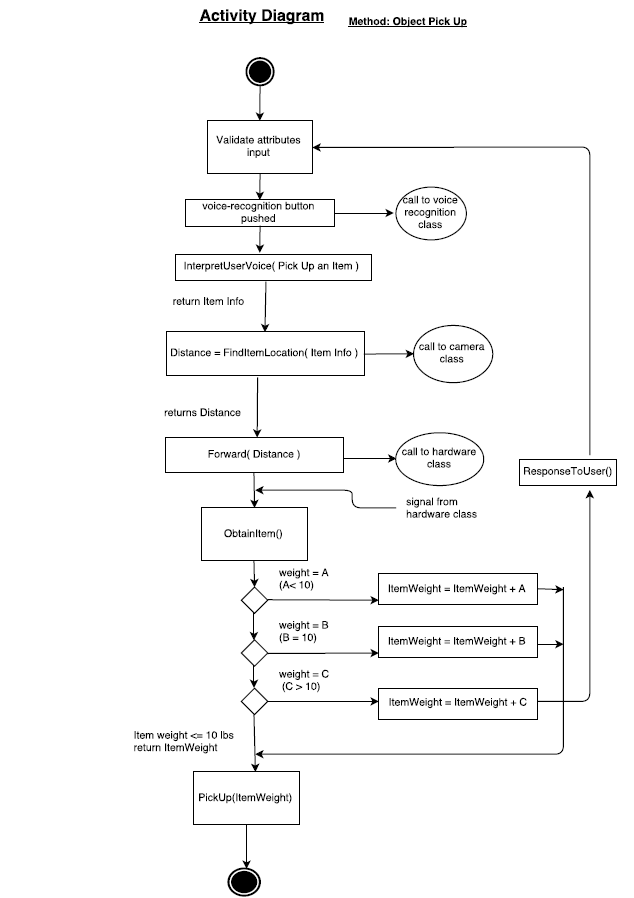
**Design Class Diagram**

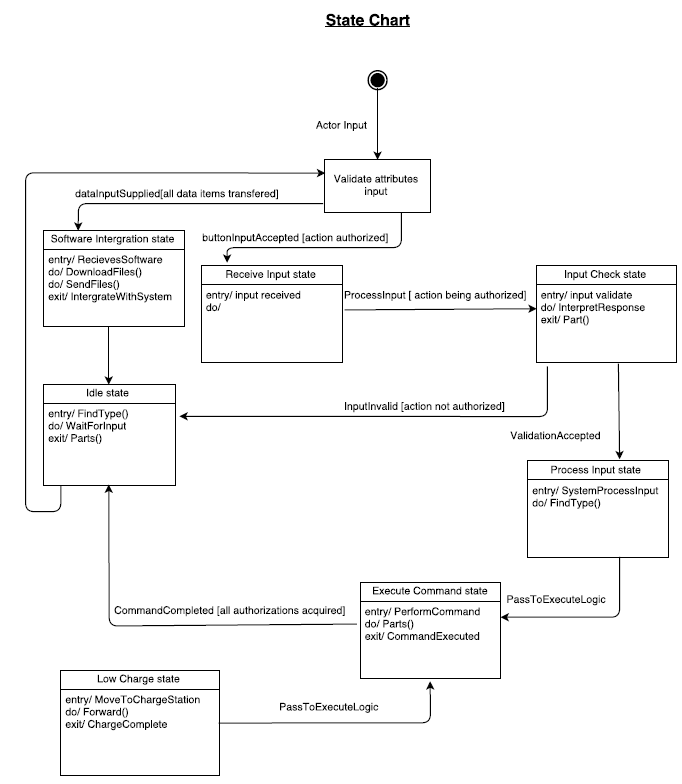




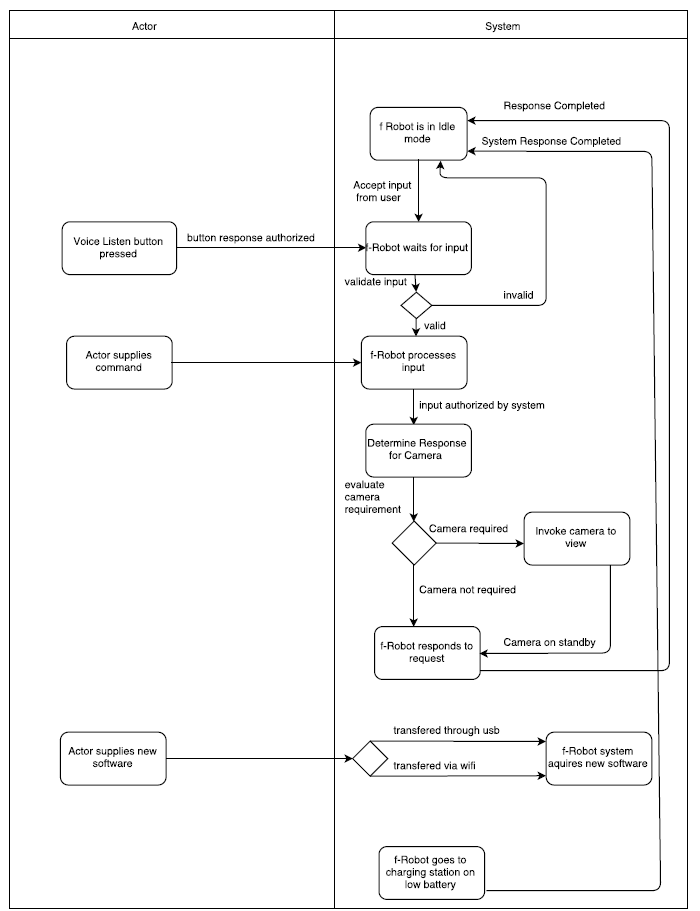
**Refined Architecture Structure Diagram**

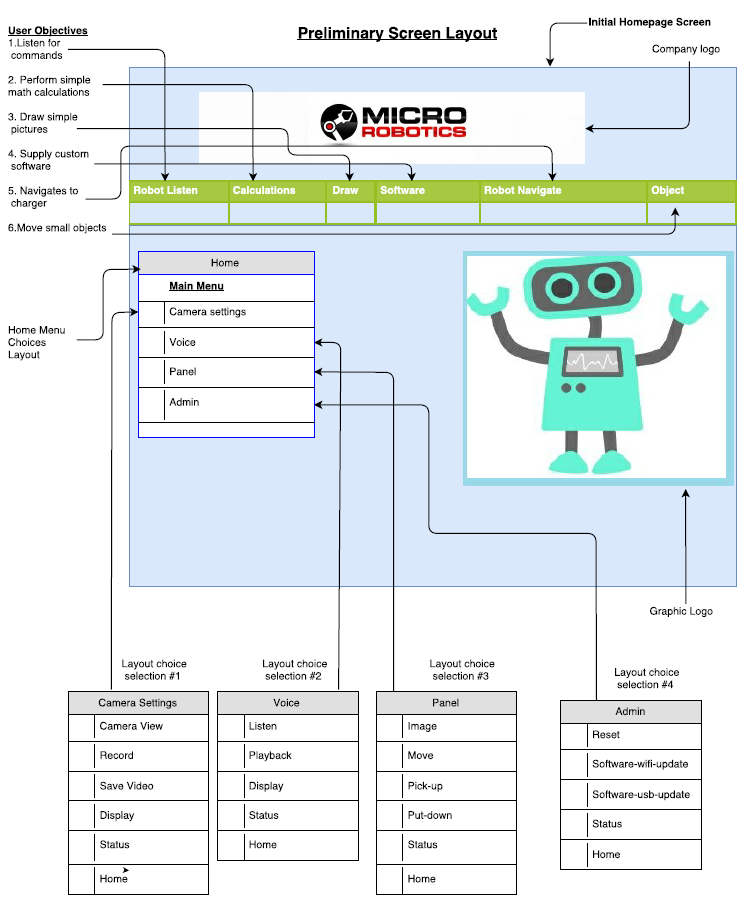


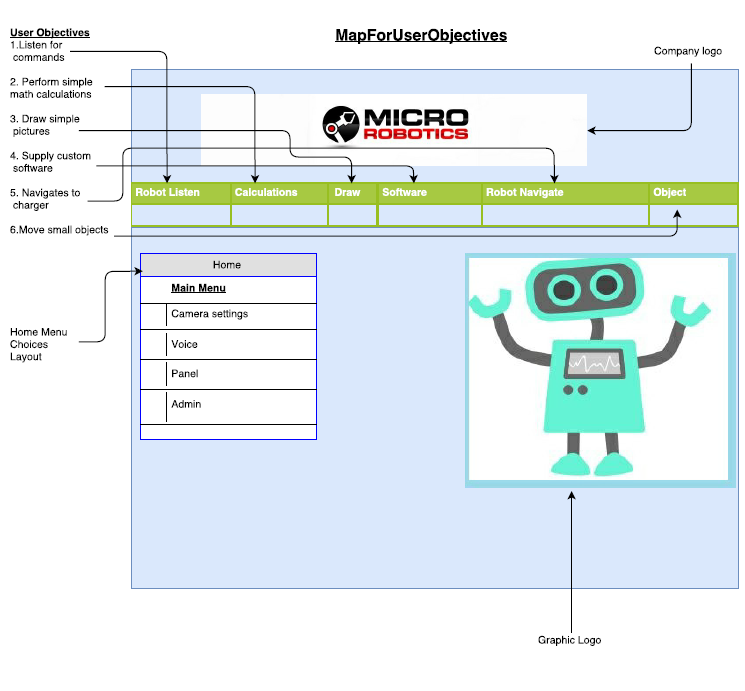




**Swim-Lane Diagram**







**Design problem:**

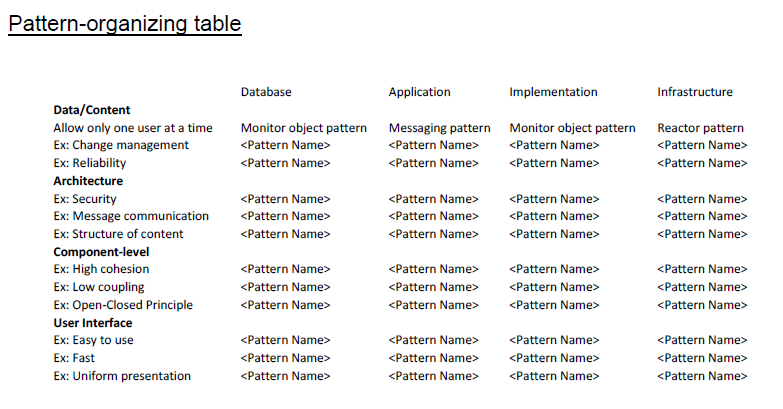
Allow only one authenticated user to login and control the f-Robot at any one time.

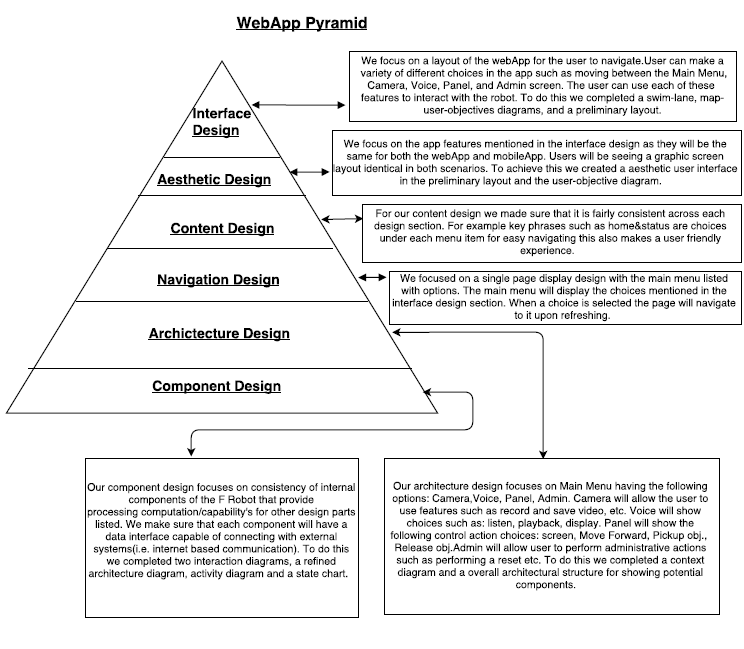
Candidate design patterns:

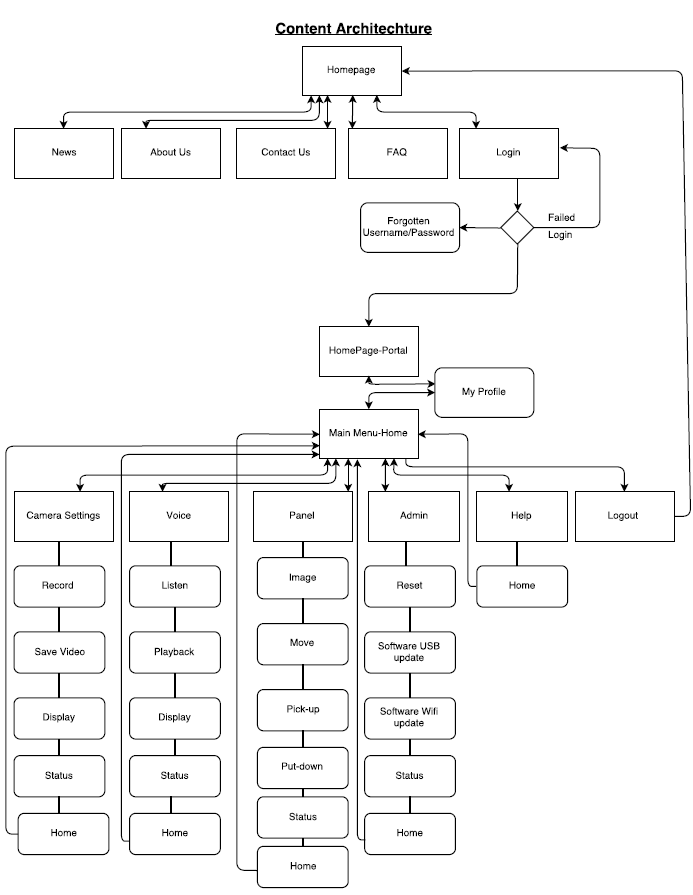
* Make use of a monitor object
* Make use of a messaging pattern
* Make use of a balking pattern

Elaboration of chosen pattern: Use of a monitor object

The f-Robot should not be able to receive commands from two different web sources at the same time as this may cause instability within the system. Only one authenticated user at any one time should be able to issue commands or change settings while connected to the f-Robot. Making use of a monitor within the operating system of the f-Robot would be a good solution to the design problem. A monitor object would be ideal because it would block unwanted operation until certain condition are met. That way if two or more different users try to access the robot via the web, only one user will be successful. Furthermore, a monitor object would be beneficial to the f-Robot because it would ensure that methods and variables are still available to use in a safe manner when interacting with the robot physically. This property would allow the f-Robot to safely receive setting or software updates via the web, while still maintain physical usability. Finally, when the monitor object releases its lock, it will be able to signal other users trying to access the f-Robot, that the f-Robot can be logged into once more.

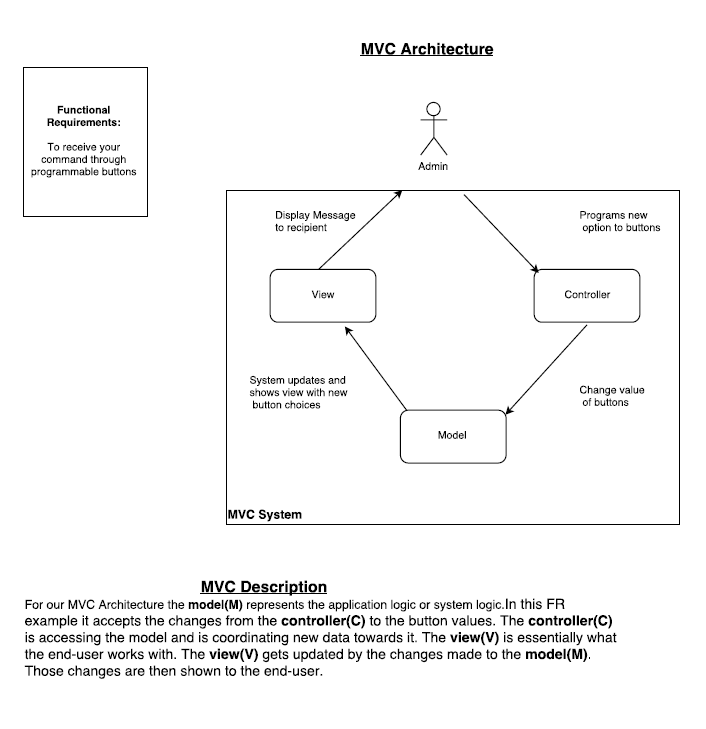


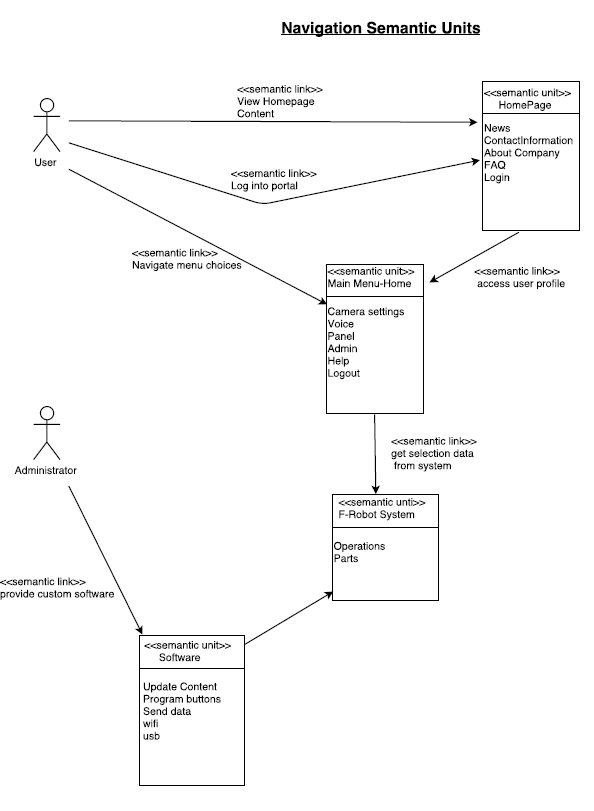




Content Architecture Explanation

For the content architecture diagram, we took an approach to develop the clear layout of the diagram using a basic linear design combined with structural concepts such as linear with diversions. This was the ideal method for executing on our layout because it provides a smooth flow of content objects laid out in a format that is easier and simple for the user to work with. The content architecture has a straight forward layout. There is a homepage with multiple topics such as news, about us, contact us, FAQ, and login. This was the route taken because we wanted to develop a friendly introduction homepage. When the user clicks on login they are taken to their account profile where they can access the main menu-home to utilize the key features of the F-Robot. Each option under the main menu also has a a list of different choices and a status and home button. The status button is for the response state of the F-Robot. The home button is a convenient way for the user to go back to the main menu-home if they want to select another option. Main menu-home also has a logout button for once the user is done working. It will direct the user to the Homepage.





**Mobile App Design – Best Practices**

* Identify your audience.

We must consider the types of people that will be using the application. Older people may not be interested in the f-Robot application and very small children may not be able to comprehend its use. The types of people who may use the f-Robot app are those interested in a more simplistic robot.

* Design for context of use.

The context for use of the f-Robot application will most likely take place within the home or possibly within a park. Therefore, the f-Robot is designed for home use.

* There is a fine line between simplicity and laziness.

The few most often used options will be placed prominently on the home screen. Other less useful options will be categorized within subsequent menus.

* Use the platform as an advantage.

The mobile device platform can be used advantageously as a remote controller for the robot. The platform will allow users to interact with the f-Robot even when physical contact is unavailable.

* Make scrollbars and selection highlighting more salient.

The use of high-contrast scrollbars and selection highlighting will allow all users, even those will colorblindness to accurate determine their place or selection on each page.

* Increase discoverability of advanced functionality.

Advanced functionality is to be separated and categorized within its own page. The advanced functionality will be one of the concluding options placed on the home screen.

* Use clear and consistent labels.

The use of widely recognized widgets and labels provided by localization will provide users with a clear set of options and progression/regression.

* Clever icons should never be developed at the expense of user understanding.

Employing standard icons that are recognizable or easily discernable will prevent users from having to exhaustively search for their goal.

* Support user expectations for personalization.

People like to personalize their surroundings to better suit themselves and/or to show their individuality. The ability to change icons/widgets, add functionality, and alter colors scheme with within the f-Robot and the accompanying app, will allow users to personalize as they see fit.

* Long scrolling forms trump multiple screens on mobile devices.

Doing what can be done with one long scrolling form on multiple screens is bothersome, inefficient, and complicated. User want to access information quickly and without having to fetch many pages. Long scrolling screens, when needed, make more sense on mobile devices.

**Mobility Environments**

The environment best suited to the needs of our f-Robot app would probably be the Construct2 IDE made by Scirra. The Construct2 IDE is best suited for our needs because it comes with a wide variety of features that are needed for the f-Robot app. Furthermore, with this IDE, the f-Robot app can be created in such a way to run on most mobile devices.

Middleware solutions will be taken care of by the Enterprise Mobility Platform provided by Smartsoft Mobile. This middleware will ensure reliable, on-demand access to the f-Robot application.

**Construction: Code**

**Main CPP**

////// Main Function////////

#include <iostream>

#include <string>

#include <fstream>

#include <mutex>

#include "SoftwareInterface.h"

#include "ButtonInterface.h"

#include "Camera.h"

#include "MovementInterface.h"

#include "Speaker.h"

#include "VoiceRecog.h"

#include "Screen.h"

using namespace std;

int main()

{

bool robotON = true;

bool connection = true;

bool ready = false;

char type;

string name;

string operations[250];

softwareInterface softwareObject;

motor motorObj;

screen screenObj;

speaker speakerObj;

voice\_recognition voiceRecogObj;

buttonInterface buttonObj;

camera cameraObj;

while (robotON)

{

if (connection)

{

ready = softwareObject.status();

if (ready != true)

{

// f-Robot will wait until processing is finished

}

// determine the connection type

// for demonstration purposes, it is given a type

softwareObject.findType('i');

cout << "Ready to recieve files.\n\n";

cout << "Files received.\n\n";

softwareObject.downloadFiles();

connection = false;

robotON = false;

}

}

cout << "f-Robot shutting down.\n\n";

system("pause");

return 0;

}

void Status()

{

// ...

}

void Wait(mutex myMutex)

{

// ...

}

void ProcessCommand(string operations, bool ready, softwareInterface softwareObject,

motor motorObj, screen screenObj, speaker speakerObj, voice\_recognition voiceRecogObj,

buttonInterface buttonObj, camera cameraObj)

{

// ...

}

void EnactClass(softwareInterface softwareObject, motor motorObj, screen screenObj,

speaker speakerObj, voice\_recognition voiceRecogObj, buttonInterface buttonObj,

camera cameraObj)

{

// ...

}

**Header Files**

//// Button Interface///////////

#ifndef button\_interface

#define button\_interface

#include <iostream>

#include <string>

#include <mutex>

using namespace std;

class buttonInterface

{

private:

bool power, reset, listen;

char type;

string buffer[250];

public:

bool status()

{

// ...

}

bool SendCommandCode(string \* buffer)

{

// ...

}

bool ListenForCommands(bool listen)

{

// ...

}

void findType(char type)

{

// ...

}

};

#endif // !button\_interface

///// Camera ///////

#ifndef cameraClass

#define cameraClass

#include <iostream>

#include <string>

#include <mutex>

#include "MovementInterface.h"

using namespace std;

class camera

{

private:

bool power, reset, watch;

char type;

string location;

string view[250];

public:

bool status()

{

// ...

}

bool PointOfLocation(string location)

{

// ...

}

bool ScanView(string \* view)

{

// ...

}

void findType(char type)

{

// ...

}

};

#endif // !cameraClass

/////// Movement Interface///////////////////

#ifndef movement\_interface

#define movement\_interface

#include <iostream>

#include <string>

#include <mutex>

#include "Camera.h"

using namespace std;

class motor

{

private:

bool release, grab, moveForward, moveBackward;

string instructions[250];

public:

bool status()

{

// ...

}

bool Forward()

{

// ...

}

bool Backward()

{

// ...

}

bool GrabAndRelease(bool open)

{

// ...

}

bool ObtainItem()

{

// ...

}

void SpeedTracker()

{

// ...

}

void findType(char type)

{

// ...

}

};

#endif // !movement\_interface

/////// Screen ////////

#ifndef screen\_class

#define screen\_class

#include <iostream>

#include <string>

#include <mutex>

using namespace std;

class screen

{

private:

bool status;

string output[250];

public:

bool status()

{

// ...

}

void Display()

{

// ...

}

void findType(char type)

{

// ...

}

};

#endif // !screen\_class

////// Software Interface//////////

#ifndef software\_interface

#define software\_interface

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class softwareInterface

{

private:

bool integration, wirelessConn, integration, usbConn;

char type;

fstream receiver;

string buffer[250];

fstream file;

public:

bool status()

{

if (this->integration != true)

{

cout << "f-Robot waiting for input files.\n\n";

return true;

}

else

{

cout << "f-Robot is busy!\n\n";

return false;

}

}

bool SendFiles(string \* buffer)

{

// call to hardware controller to integrate new instructions

// assume instructions successfully integrated

return true;

}

bool downloadFiles()

{

bool status;

int i = 0;

this->integration = true;

fstream file;

file.open("Text.txt", ios::in);

while (!file.eof())

{

getline(file, buffer[i]);

i++;

}

while (this->integration == true)

{

i = 0;

status = SendFiles(this->buffer);

if (status == true)

{

cout << "Software successfully integrated!\n\n";

this->integration = false;

i++;

return true;

}

else

{

cout << "Error: downloadFiles(fstream file)\n\n";

this->integration = false;

return false;

}

}

}

void findType(char type)

{

if (type == 'u')

{

cout << "USB connection detected!\n\n";

this->type = 'c';

}

if (type == 'i')

{

cout << "Internet connection detected!\n\n";

this->type = 'i';

}

if (type == 'b')

{

cout << "Bluetooth connection detected!\n\n";

this->type = 'b';

}

}

};

#endif // !software\_interface

/////// Speaker /////////

#ifndef speaker\_class

#define speaker\_class

#include <iostream>

#include <string>

#include <mutex>

using namespace std;

class speaker

{

private:

bool status;

string output[250];

public:

bool status()

{

// ...

}

void Speak()

{

// ...

}

void findType(char type)

{

// ...

}

};

#endif // !speaker\_class

/////// Voice Recognition ///////

#ifndef voice\_recognition\_class

#define voice\_recognition\_class

#include <iostream>

#include <string>

#include <mutex>

#include <chrono>

#include "Speaker.h"

#include "Camera.h"

using namespace std;

class voice\_recognition

{

private:

struct interpreter

{

string inBuffer[250];

string outBuffer[250];

int status;

bool processing;

};

bool classStatus, speaking;

char type;

string instructions[250];

public:

bool DisplayStatus()

{

// ...

}

bool InterpretUserVoice(interpreter obj)

{

// ...

}

void ResponseToUser(interpreter obj)

{

// ...

}

void SignalToScreen(string \* instructions)

{

// ...

}

void findType(char type)

{

// ...

}

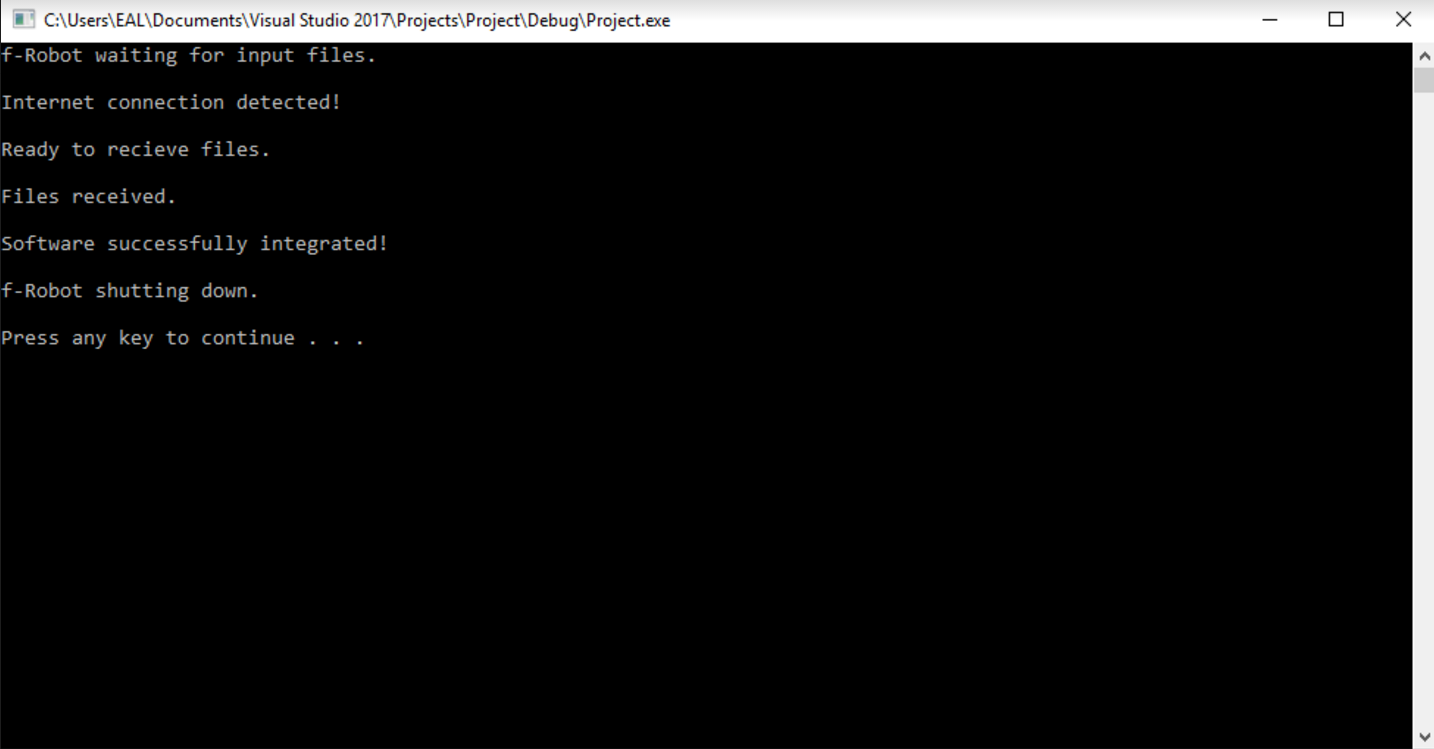
};

#endif // !voice\_recognition\_class

//// Voice //////

#pragma once

**Construction: Testing**



**Case Tools**

For this project, we utilized the textbook (Software engineering: a practitioner's approach) required by the course to complete much of the analysis models and designs shown above. We also used the draw.io software available on the web to illustrate each model/diagram. This web based diagramming application allowed us to work hands on with editing and reshaping designs that needed corrections as well. We also utilized Microsoft Visual Studio to test the code that we constructed. The debugging features available in the software helped with narrowing down bugs that were found and allowed for a smooth transition from code development to testing for verification.

**References**

JGraph Ltd. "Free Flowchart Maker and Diagrams Online." *RSS*. N.p., 2005. Web. 07 Mar. 2017.

Pressman, R. S., & Maxim, B. R. (2015). Software engineering: a practitioner's approach. New York, NY: McGraw-Hill Education.

