INTEGRATED DESIGN PROJECT

Deliverable Overview

Michael D'Argenio – mjdargen@ncsu.edu
Electrical Engineering – SS 2019 – Duke TIP

Key IDP Deliverables

- Project Introduction & Motivation
- Product Requirements
- Roles & Responsibilities
- Hardware System Architecture Diagram(s)
- Software System Architecture Diagram(s)
- Project Management Plan
- Test/Demo Plans
- Issues/Risks

PROJECT INTRODUCTION & MOTIVATION

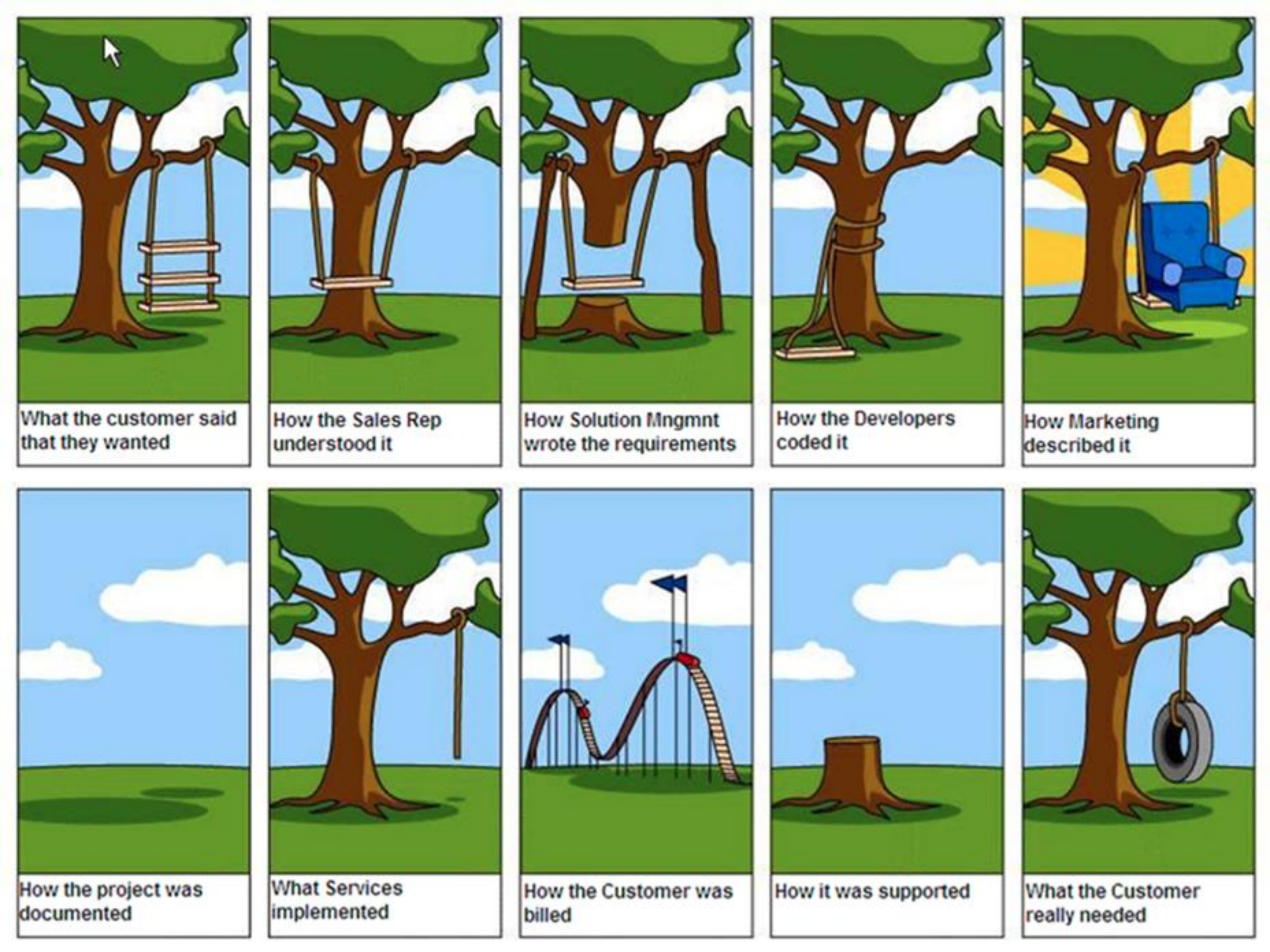
“Elevator Pitch” for your Project

- A quick pitch to get someone interested.
- Should answer the following questions:
  - What is your project?
  - Why are you doing this project?
  - What problem does it solve?
- This will be how you start any presentation or discussion of your project.

PRODUCT REQUIREMENTS

What are product requirements?

- One of the most important project documents.
- Written from the perspective of the end user.
  - What the end user needs or expects.
  - Make sure it fits the end user's actual needs and not their perceived "needs".
- Need to have a clear expectation of the goals before starting the project.
- All teams should be in agreement on requirements.


What the customer said that they wanted
How the Sales Rep understood it
How Solution Mngmnt wrote the requirements
How the Developers coded it
How Marketing described it
How the project was documented
What Services implemented
How the Customer was billed
How it was supported
What the Customer really needed

What is a product requirement?

- Features and functions of the product being developed.
- Includes constraints such as costs, reliability, ease of use, standards compliance, time to develop.
- Quantify requirements & refine the sponsor problem statement (e.g. instead of "fast", say "100MB/sec) as much as possible to reduce ambiguity.
- Two types of requirements
  - SHALL – required e.g. The system shall be less than $5.
  - MAY – optional e.g. The system may have Bluetooth.

What vs. How

- Requirement "WHATs" vs Design Implementation "HOWs"
  - Have clear understanding on WHAT needs to be done.
  - Avoid specifying HOW it will be done yet.
  - Stating specifically HOW it needs to be done in requirements backs you into a corner.
- Good example: Product shall provide location within 500' accuracy.
- Bad example: Product shall acquire GPS coordinates using u-blox SAM-M8Q.
- Prevents us from using different GPS devices or investigating different methods to locate such as wi-fi triangulation, etc.

Quantifying/Refining Requirements

- Product requirements should be measurable/quantifiable.
- Can refine the requirements as project progresses.

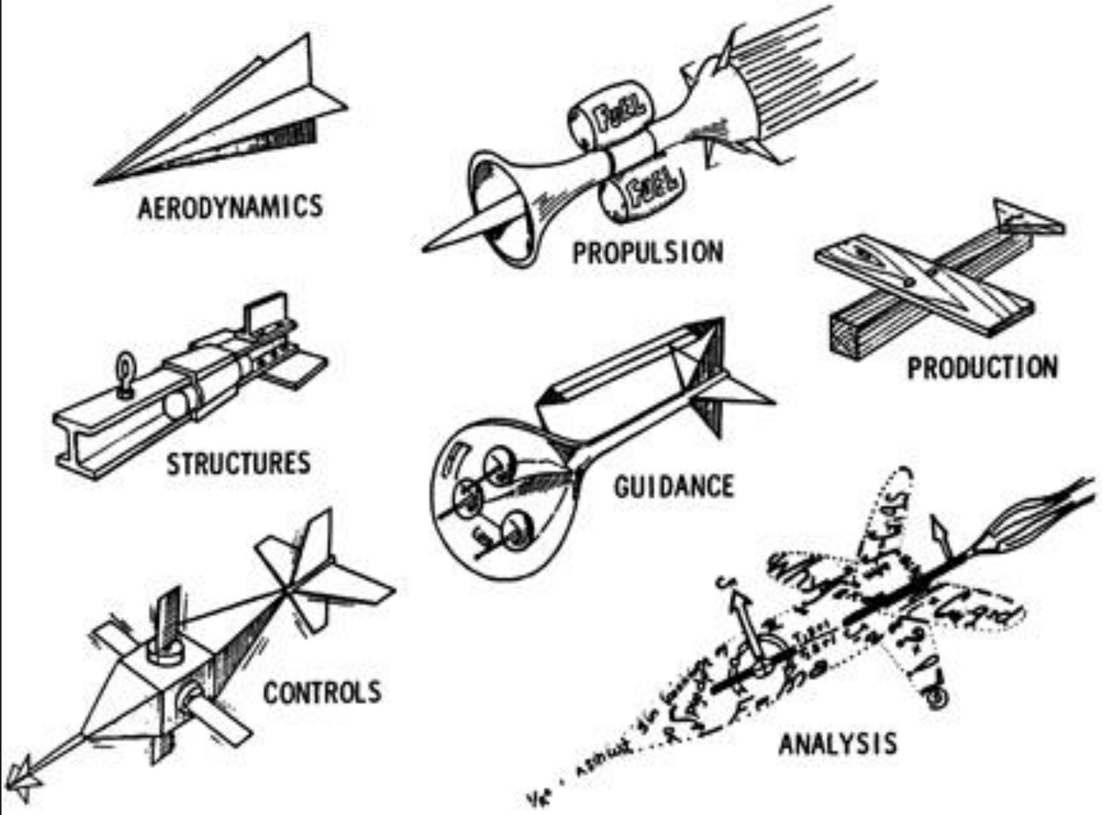
Poor – vague, unmeasurable	Better – specific, quantifiable
Software algorithm shall be fast	Software algorithm shall execute in < 2 secs
System should be small and portable.	System shall weigh < 5 lbs. System shall be small enough to be able fit into a backpack.
Battery shall last for a long time and charge fast.	System shall operate continuously for at least 5 hours on a single charge. Batteries shall be re-chargeable from 0 to full within 2 hours.
Product should be cheap.	Prototype cost shall not exceed $100. Bulk manufacturing cost per unit shall not exceed $20.

Product Requirements Exercise

Work in teams and determine whether the following requirements are good or bad.

1. "On power loss, battery backup must maintain normal operation for 20 min"
2. "After 3 unsuccessful access attempts software must lock user out of the system"
3. "System must measure blood pressure"
4. "Product must meet FDA Title 21, Part 874 standard for medical devices"
5. "Product may be able to operate outdoors"
6. "Product must use C++ language"
7. "Product must be easy to use for kids"
8. "Accuracy of AC current measurement must be within 1%"
9. "Motor speed error must be <10% within 2sec of pressing start button"
10. "Smartphone display must graph 1 month of user heart rate data"
11. "Product may be cheaper than previous product"
12. "Device must travel faster than the speed of light"

Tradeoff Analysis of Requirements



Case Study: Lighting for Remote Areas

- **Motivation:** Many countries still don't have electricity in remote areas which affects standard of living. The reason for this is cost as well as resource availability. Current solutions are hazardous.
- **Problem Statement:** Goal is to design and develop a cheap and clean light source which can be easily used by anybody. It will not need any maintenance or running cost. It should work on a readily available energy source in remote locations.
- **Exercise:** What could be some key Product Requirements from problem statement?

Example Requirements: Lighting

- Must provide sufficient light for night-time activities (> 30 LUX)
- To be used by the average person
  - Simple on/off switch
  - Ability to easily adjust brightness
  - Simple hanging feature
  - East to setup with common tools available (hammer and screwdriver)
- Should be attractive, but appearance is secondary (function over form)
- Must be robust - operate in -20 C to 140 F, survive a fall from 4 / 5 feet
- Must be portable (Weighs < 3 lbs, smaller than a breadbox)
- Must be low cost (initial cost ~$20) and require no “run” cost
- Must last 4 hours & be easily rechargeable using available energy source
- Must be very simple setup and repair

Solution: GravityLight

- **Problem:** Need new source of light in remote areas. Current Kerosene lamp solutions can be dangerous.
- **Initial Thought:** Solar powered LEDs
- **Final Solution:** Through prototyping, tradeoff analysis, etc. they have come up with a much more innovative solution: GravityLight
- **Observations**: Working with the community instead of for the community allowed the team to be more creative and better serve the people.

ROLES & RESPONSIBILITIES

Collaboration

- Project teams comprised of many different roles:
  - Engineering
    - Mechanical
    - Industrial Design
    - Hardware
    - Software
    - Possibly others
  - Quality
  - Manufacturing
  - Purchasing/Supply Chain
  - Services
  - Marketing/Sales
  - Finance

- In order to be able to effectively work together, need to clearly define who is responsible for what

For Your Project

- Define technical, project management, and documentation roles

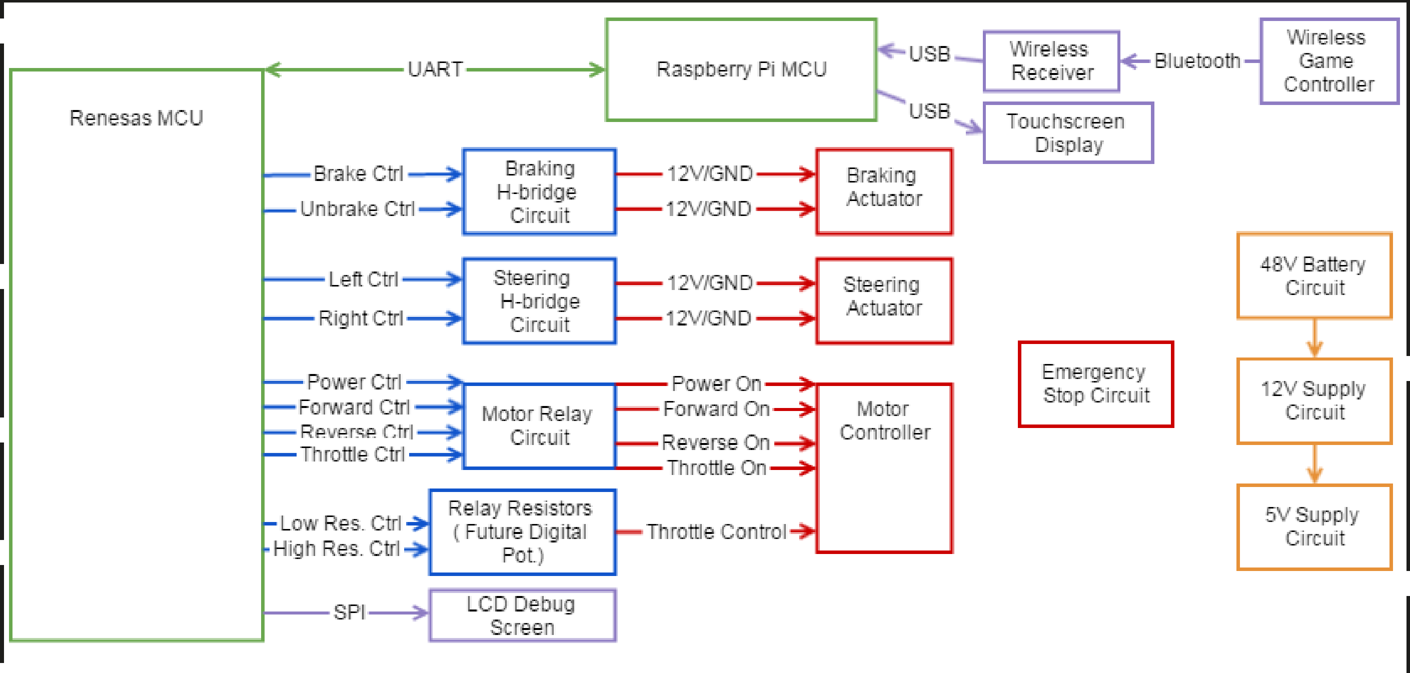
- Finn
  - Project Manager
  - Circuit Designer/Hardware Diagrams
  - Write software to communicate with peripherals
- Jake
  - Documentarian
  - Software Diagrams
  - Main software designer
  - Mechanical housing designer

HARDWARE SYSTEM ARCHITECTURE DIAGRAMS

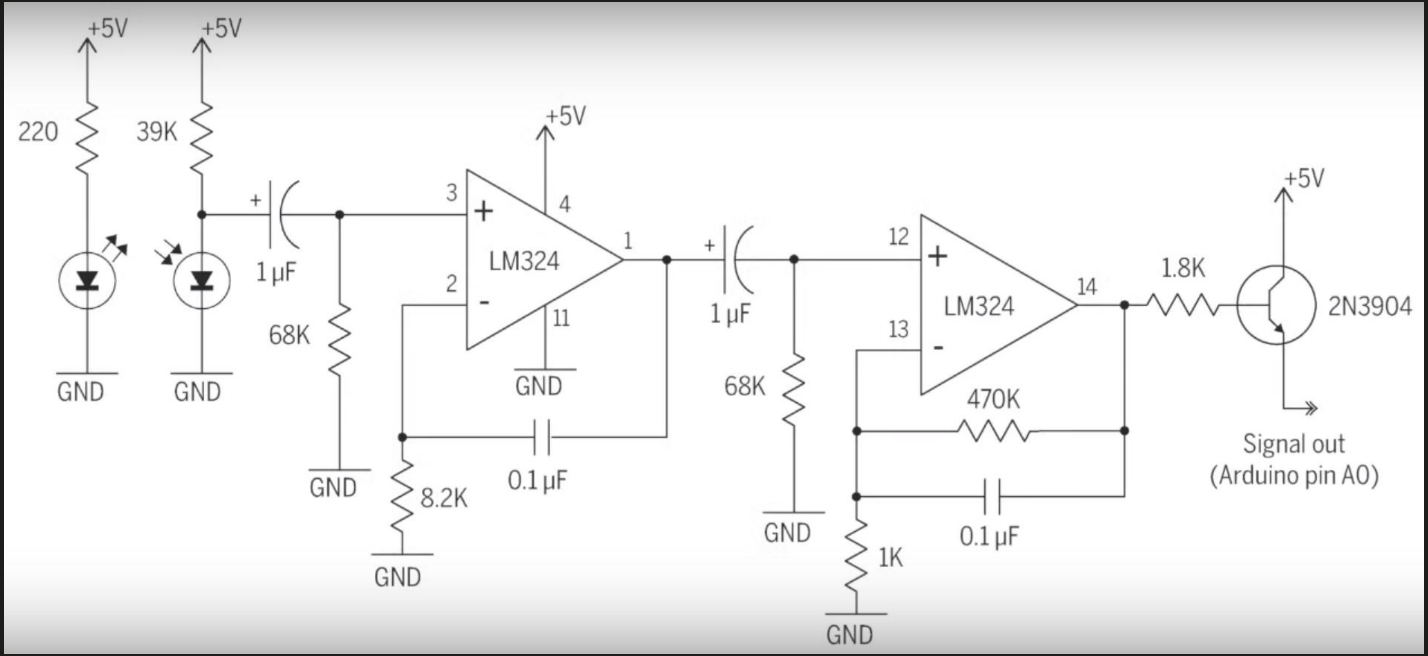
HARDWARE SYSTEM DIAGRAMS

- Should have a high-level system block diagram and possibly several other circuit diagrams.
  - Circuit diagram or schematic – shows all electrical interconnections between individual components
  - High-level system block diagram – shows major interconnections between parts of the system
- Can be hand drawn or you can use the following programs:
  - Fritzing – made by Arduino, will create circuit schematics and breadboard connection diagrams.
  - LTspice – create schematics and run simulations to see how the circuit behaves and see current/voltage plots.

High-Level System Block Diagram



Circuit Diagram/Schematic



SOFTWARE SYSTEM ARCHITECTURE DIAGRAMS

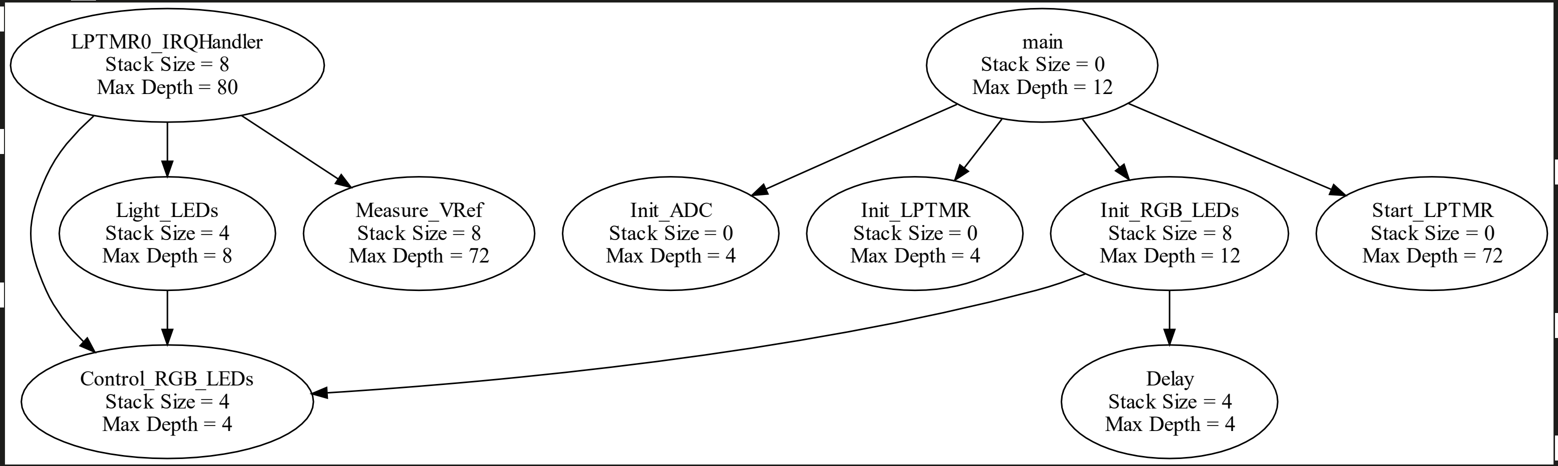
Software System Diagrams

- There are many different types of software diagrams. Here are some that may be the most useful to you:
  - Call Graph
  - Control Flow Graph
  - State Machine Diagram
  - Sequence UML Diagram
- However, there are many, many others that you can use to help design your software:
  - Data/Control Dependence Graph
  - Timing Diagrams
  - Pseudocode
  - And more…

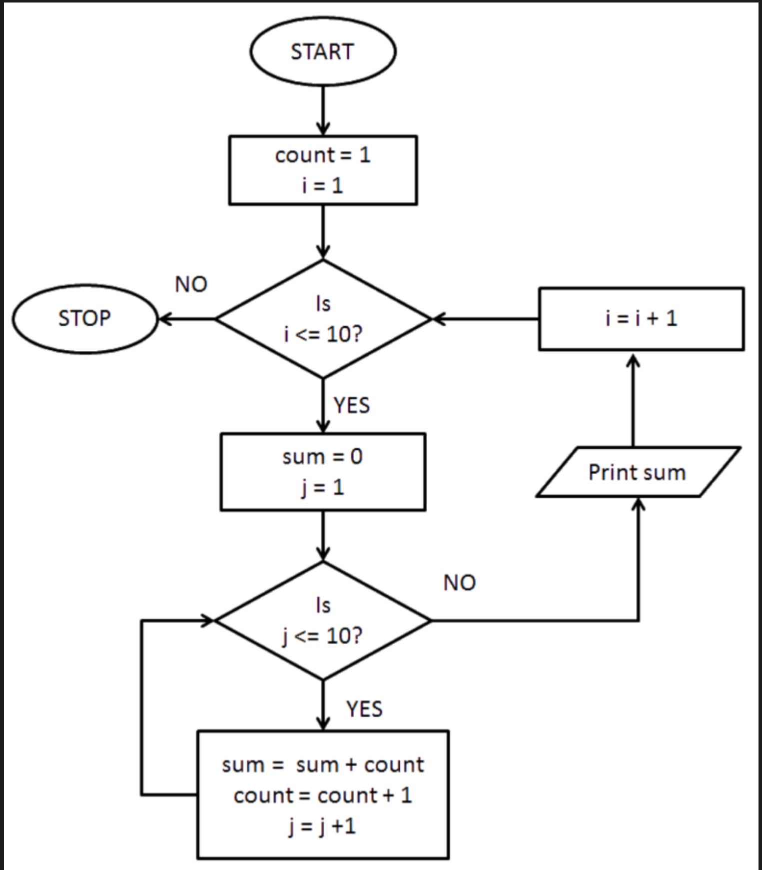
For Your Project

- Select any of the graphs or diagrams that help you design the software for your specific application.
- May need to use several different types of graphs to help you understand how all parts of the system should work together.
- Control flow graphs, state machine diagrams, and sequence UML diagrams will likely be the most useful for your projects.
- Writing pseudocode before you begin can be very helpful.
- Start with higher-level diagrams, then move to lower-level diagrams.

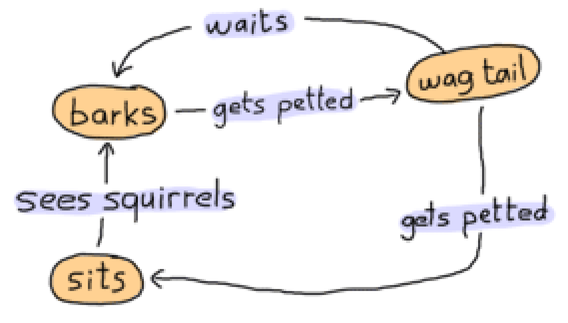
Call Graph


LPTMR0_IRQHandler
Stack Size = 8
Max Depth = 80
main
Stack Size = 0
Max Depth = 12
Light_LEDs
Stack Size = 4
Max Depth = 8
Measure_VRef
Stack Size = 8
Max Depth = 72
Init_ADC
Stack Size = 0
Max Depth = 4
Init_LPTMR
Stack Size = 0
Max Depth = 4
Init_RGB_LEDs
Stack Size = 8
Max Depth = 12
Start_LPTMR
Stack Size = 0
Max Depth = 72
Control_RGB_LEDs
Stack Size = 4
Max Depth = 4
Delay
Stack Size = 4
Max Depth = 4

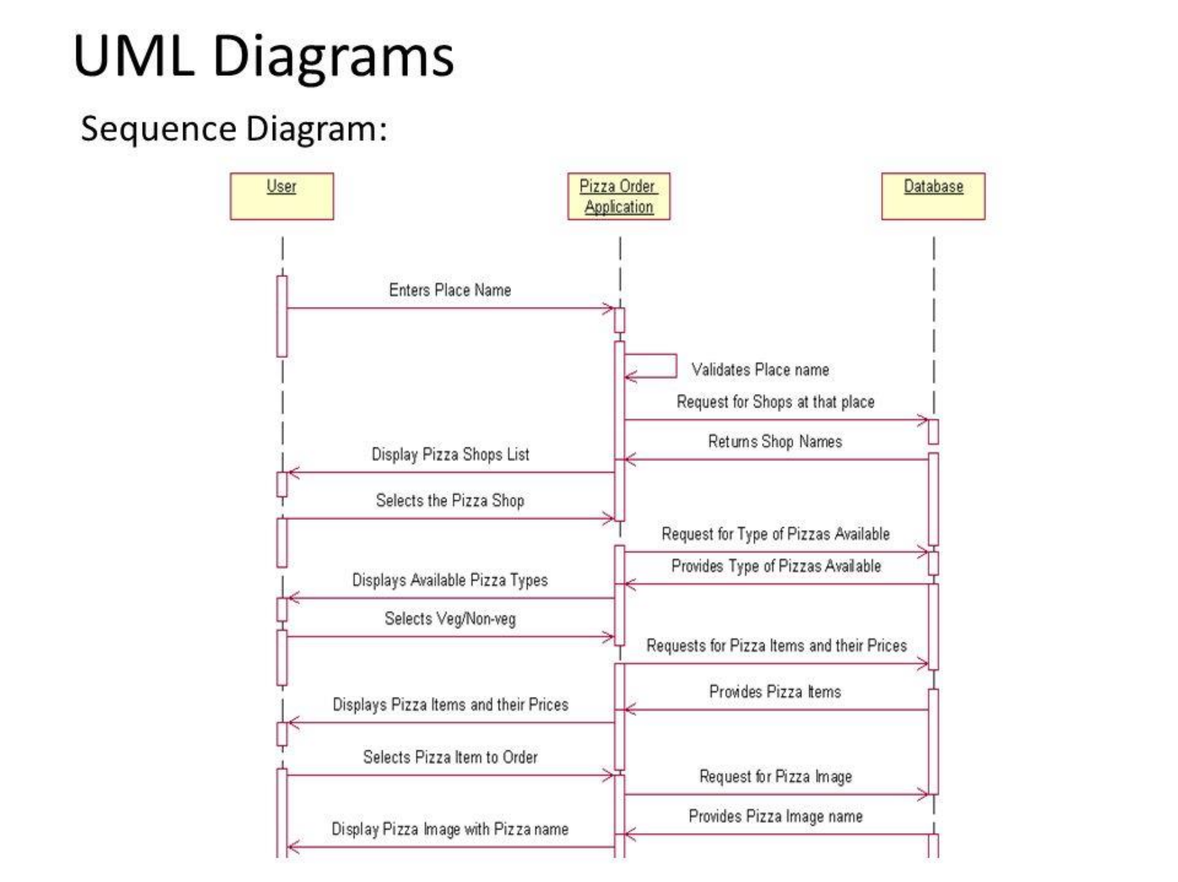
Control Flow Graph



State Machine Diagram



Sequence UML Diagram



PROJECT MANAGEMENT PLAN

Gantt Chart

- Break down project into individual tasks.
- Schedule each individual task based on how long it will take and when it should be completed
- Keep in mind dependences.
- Assign each task to specific people.
- Keep careful track of milestones, deliverables, and test plans.
- Use this template: IDP Gantt Chart Template

Example Gantt Chart

			Mon			Tue			Wed			Thu			Fri	
Task	**Owner**	**% Complete**	**M**	**A**	**E**	**M**	**A**	**E**	**M**	**A**	**E**	**M**	**A**	**E**	**M**	**A**
Task 1	Finn	100%														
Task 2	Jake	100%														
Task 3	Finn	100%														
Task 4	Jake, Finn	100%														
Task 5	Jake	25%														
Task 6	Finn	50%														
Task 7	Jake	0%														
Task 8	Finn	0%														
Task 9	Jake	0%														
Task 10	Finn	0%														
Task 11	Jake	0%														
Task 12	Finn	0%														

Legend

- Preliminary Design Review
- Alpha Demo
- Beta Demo
- Final Demo
- Task

TEST/DEMO PLAN

Test Plan

- Test your project to make sure it meets each one of your requirements.
- Test early
  - Can’t just test at the end and hope it works
  - Break project up into subsystems
  - Test subsystems then test whole system
- Test iteratively
  - Test after every step/change to make sure nothing has broken

Demo Plan

- Make sure your project management plan and test plans align with the milestone demos.
- Milestones
  - Alpha – successful demo of all subsystems
  - Final – successful demo entire system

ISSUES/RISKS

Issues

- An issue is an unexpected problem or unsettled matter.
- An issue is not a future concern (that's a risk).
- It could be a technical or project management issue.
- They can affect project progress and delay the project.
- Be sure you correctly diagnosed issue and have a plan for overcoming issue.

- Example
  - Issue – Not able to successfully communicate with sensor.
  - Plan – Use scope to check communications. Seek help.

Risks

- A risk is a future event that could negatively impact the project progress or result i.e. an issue that may arise down the road.
- Decide whether to Accept/Mitigate/Avoid
  - Accept: Risk is minimal. If it issue arises you can live with it. Accept and do nothing.
  - Mitigate: Risk is fairly large. Anticipate that risk could occur and preemptively identify back-up plans.
  - Avoid: Risk is too great. Avoid and go a different route.

- Example
  - Risk – May not enough GPIO ports to drive all LEDs.
  - Result – Mitigate – Investigate demultiplexing GPIO ports.