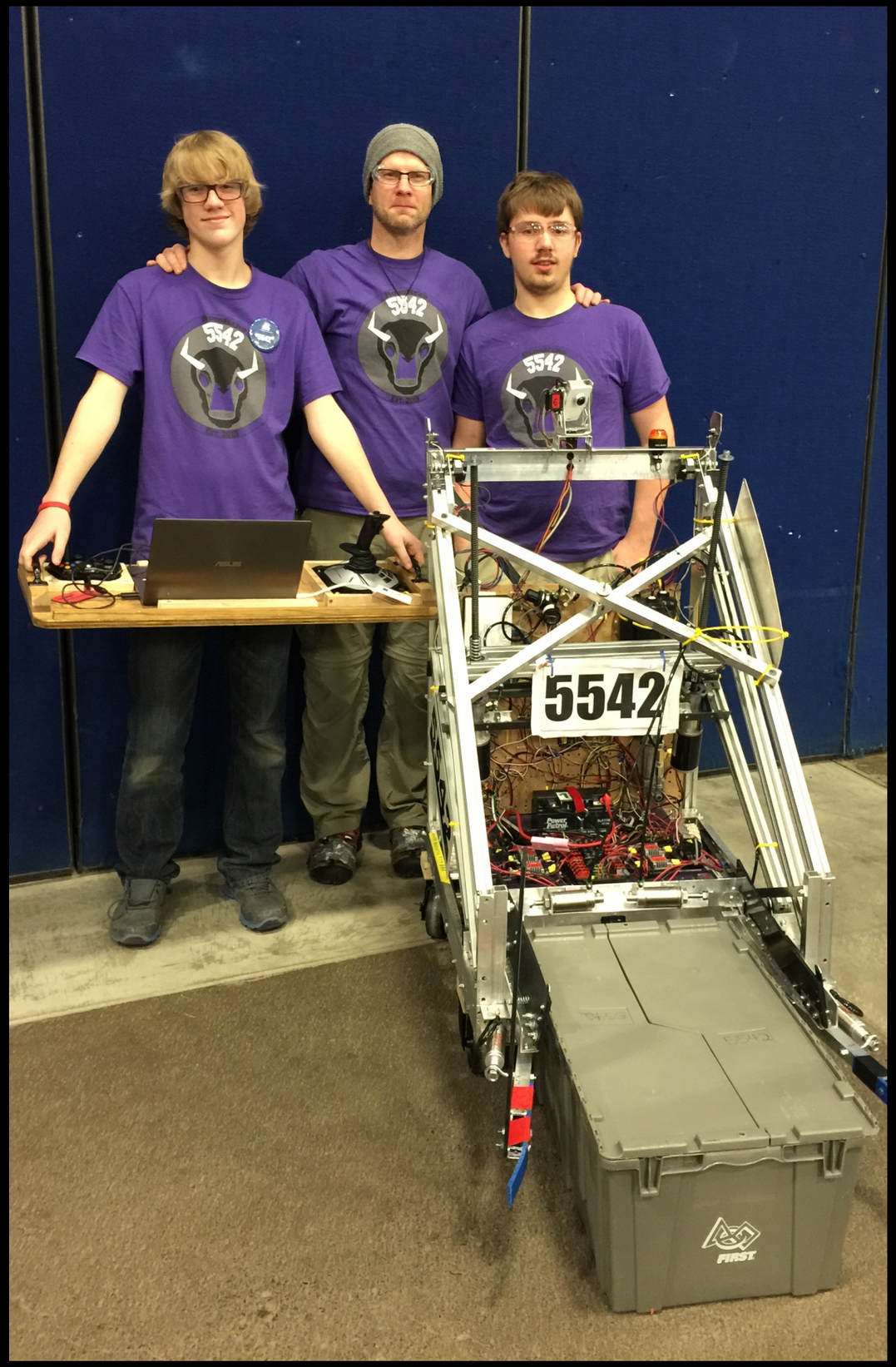
FRC Control System Basics

Coding Languages - C++ / Java

Chris Roadfeldt

chris@roadfeldt.com



Who am I and what do I know?

- Lead CSA
- CIS (4607) mentor.
- Enough to be dangerous… :)
- Proper research and planning saves time and frustration.
- Taking on a challenge brings growth.
- Perl, Bourne, some C, some C++, some Java, etc…
- Basic electronics, 3d printing, Unix, Linux
- General geekery

Topics

- Facilitate and promote discussion intra-team and inter-team.
- What is WPILib?
- Role of WPILib
- WPILib Framework overview
- WPILib base classes
- Pros / Cons of C++ and Java
- If there is time.
    - Driver input to robot action
    - Role of FMS
    - Eclipse
    - Source Control - GIT

Resources / Helping each other out

- http://wpilib.screenstepslive.com/s/4485
  - The resource for FRC control system documentation
- https://www.reddit.com/r/FRC/
  - Discussion group related to FRC
- http://www.chiefdelphi.com/forums/portal.php
  - Battle tested, well known site for all things geeky, robotics and FRC related.
- Can not emphasize enough the importance of reaching out to mentors, other teams or discussion groups.
- Never be afraid to ask your question.
- Ask your questions anytime during this discussion.

What is WPILib?

- The WPI Robotics library (**WPILib**) is a set of software classes that interfaces with the hardware and software in your FRC robot's control system. There are classes to handle sensors, motor speed controllers, the driver station, and a number of other utility functions such as timing and field management.

- Generalized abstraction layer between your code and the hardware.

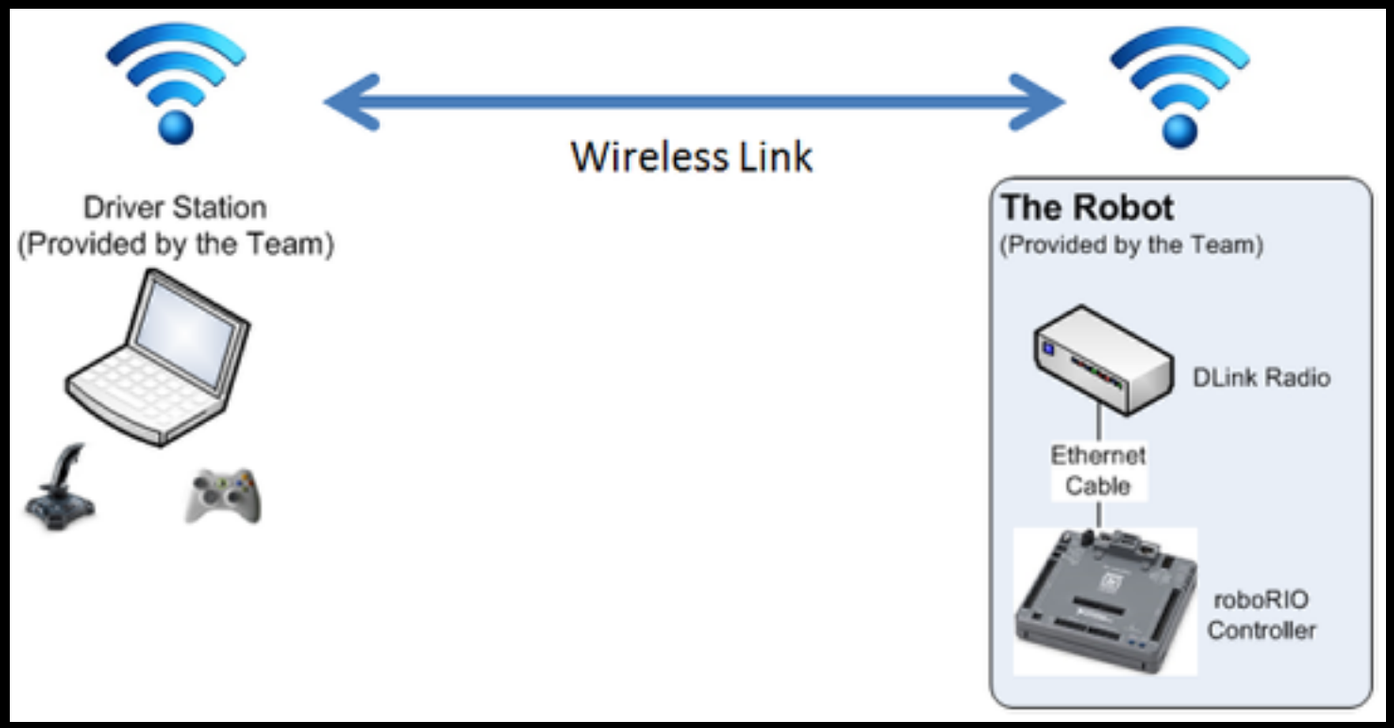
- Created by Worcester Polytechnic Institute Robotics Resource Center

- http://wpilib.screenstepslive.com/s/4485/m/13809/l/241852-what-is-wpilib

WPI

Driver Input to Robot Action

- Xbox / Joystick - AKA Operator Interface
- DS ->
- Laptop / OS ->
- Network ->
- RoboRio / OS->
- Robot Code / WPILib ->
- Stage appropriate code segment ->
- command ->
- subsystem ->
- WPILib HAL ->
- CAN / PWM / GPIO ->
- Motor controller



Operator Interface

- Takes input from Operator / Driver
  - Joystick, XBos controller, Arduino, etc…
- Sends input to Driver Station

- Can be any applicable device you want.
- Anything not part of system of parts will need to be custom coded.
- Provides Feedback to Operator / Driver
  - SmartDashboard, ShuffleBoard - NEW!

Driver's Station

- Controls the Robot, only thing that controls the robot.
    - Sends driver input, dashboard inputs, etc
- Captures and logs robot runtime information
- Interfaces with FMS.
- Polls Robot at regular intervals for health check, positive control confirmation and timing.
    - When in trouble, HIT THE SPACE BAR!

Role of FMS

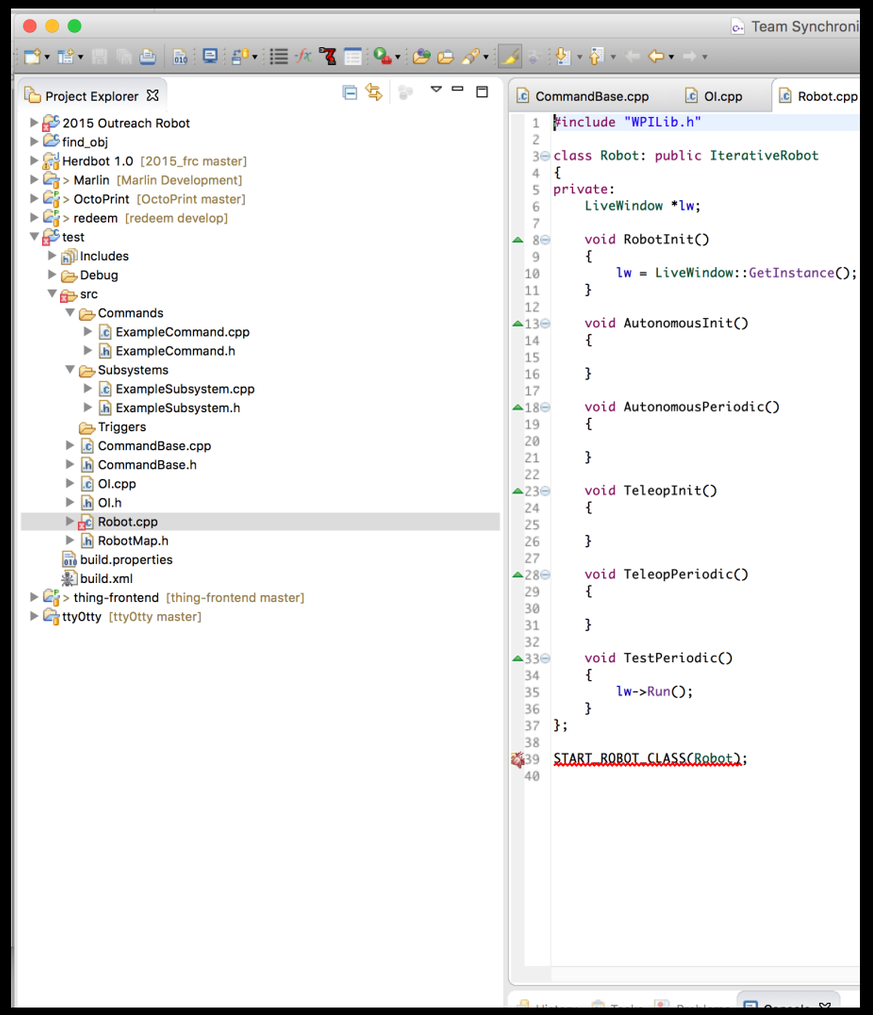
- FMS controls
  - Field electronics
  - Network infrastructure
    - Match scheduling
    - All field Hardware
      - Timers, team lights, estops, etc
      - Scoring in real-time
    - Audience screen
    - Uploading results to internet
  - FMS talks to each team's Driver Station
  - FMS Does NOT control the robot directly
  - FMS Whitepaper
    - http://wpilib.screenstepslive.com/s/4485/m/24193/l/291972-fms-whitepaper

RoboRio / OS

- The Brains of the Robot
- Runs Linux
- Robot Code runs as app
- Provides several interface buses
  - CAN, I2C, PWM, DIO, SPI, MXP
- Controls everything directly or indirectly on Robot.

Robot Code

- Written using WPILib or WPILib wrapper.
- Officially supported languages
  - C++, Java, LabView
- Unofficial languages
  - Python
- Takes input from DS and uses RoboRio busses to control hardware devices.
- Must be in control of all physical actions the robot can make.



Motor Controllers

- Controls electrical power sent to motors.
- Connected to RoboRio via a Bus. eg; CAN
- Takes input from Robot Code only.
- Connected to PDP for power source.
- Can be pro-active or reactive.
  - Talon SRX / Victor SPX vs Talon SR / Victor 883

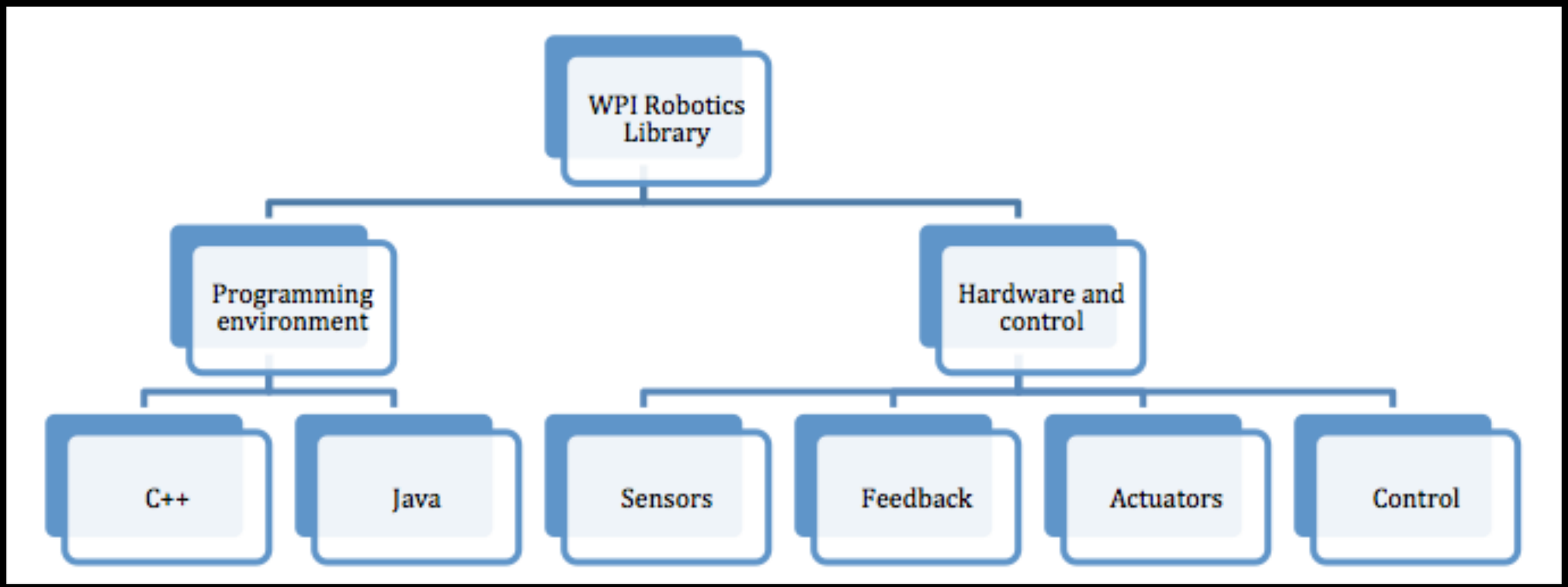
Q & A

Role of WPILib

- Provides a generalized common programming framework, API and HAL to control your robot.
- Provides templates for program logic flow.
    - AKA: Base Classes
- Provide generalized access to various hardware components.
    - Sensors, motor controllers, rangefinders, servos, CAN bus, I2C, GPIO, etc..
- Integrates with Driver's Station and Smart Dashboard
- Allows for code re-use.
    - WPILib is thoroughly tested code.
    - Carry over code from previous years.
    - Intra and Inter-team sharing of code
    - etc…

WPILib Framework

High level overview of components.



Base Classes

- Sample Robot
  - Allows for total control over program flow.
  - Code is called once per stage.
    - Extremely flexible
    - Learning curve depends on complexity.
- Iterative Robot
  - Provides a structure to build robot code from.
  - Minimal learning curve
  - Code is called in a periodic and iterative nature.

Base Classes

- Timed Robot
  - Based on Iterative.
  - Internally Timed - More resilient / robust robot actions that require timing.
  - Possible sync issues with controls. Breaks sync with DS.
- Command Based Robot
  - Provides a structure to build robot code from.
  - Moderate learning curve
  - Code is called in a periodic / iterative and branched based on stimuli.

Sample Robot - C++

- You control the logic flow.
- Provides very basic code to interface with robot.
- All functionality is in your code.
- Most logic flow is in your code.
- Useful for quick testing or very advanced logic flow.
- Two stages
  - Autonomous
  - TeleOp
- Requires loop to continue to function.
- Code readability depends on complexity.
- All code in one file.

Robot.cpp

```
1  #include "WPILib.h"
2
4  * This is a demo program showing the use of the RobotDrive class.
13 class Robot: public SampleRobot
14 {
15     RobotDrive myRobot; // robot drive system
16     Joystick stick; // only joystick
17
18 public:
19     Robot() :
20             myRobot(0, 1),  // these must be initialized in the same order
21             stick(0)        // as they are declared above.
22     {
23         myRobot.SetExpiration(0.1);
24     }
25
26     /**
27      * Drive left & right motors for 2 seconds then stop
28      */
29     void Autonomous()
30     {
31         myRobot.SetSafetyEnabled(false);
32         myRobot.Drive(-0.5, 0.0);   // drive forwards half speed
33         Wait(2.0);                  //    for 2 seconds
34         myRobot.Drive(0.0, 0.0);    // stop robot
35     }
36
37     /**
38      * Runs the motors with arcade steering.
39      */
40     void OperatorControl()
41     {
42         myRobot.SetSafetyEnabled(true);
43         while (IsOperatorControl() && IsEnabled())
44         {
45             myRobot.ArcadeDrive(stick); // drive with arcade style (use right stick)
46             Wait(0.005);                // wait for a motor update time
47         }
48     }
49
50     /**
51      * Runs during test mode
52      */
53     void Test()
54     {
55     }
56 };
57
58 START_ROBOT_CLASS(Robot);
59
```

Iterative Robot - C++

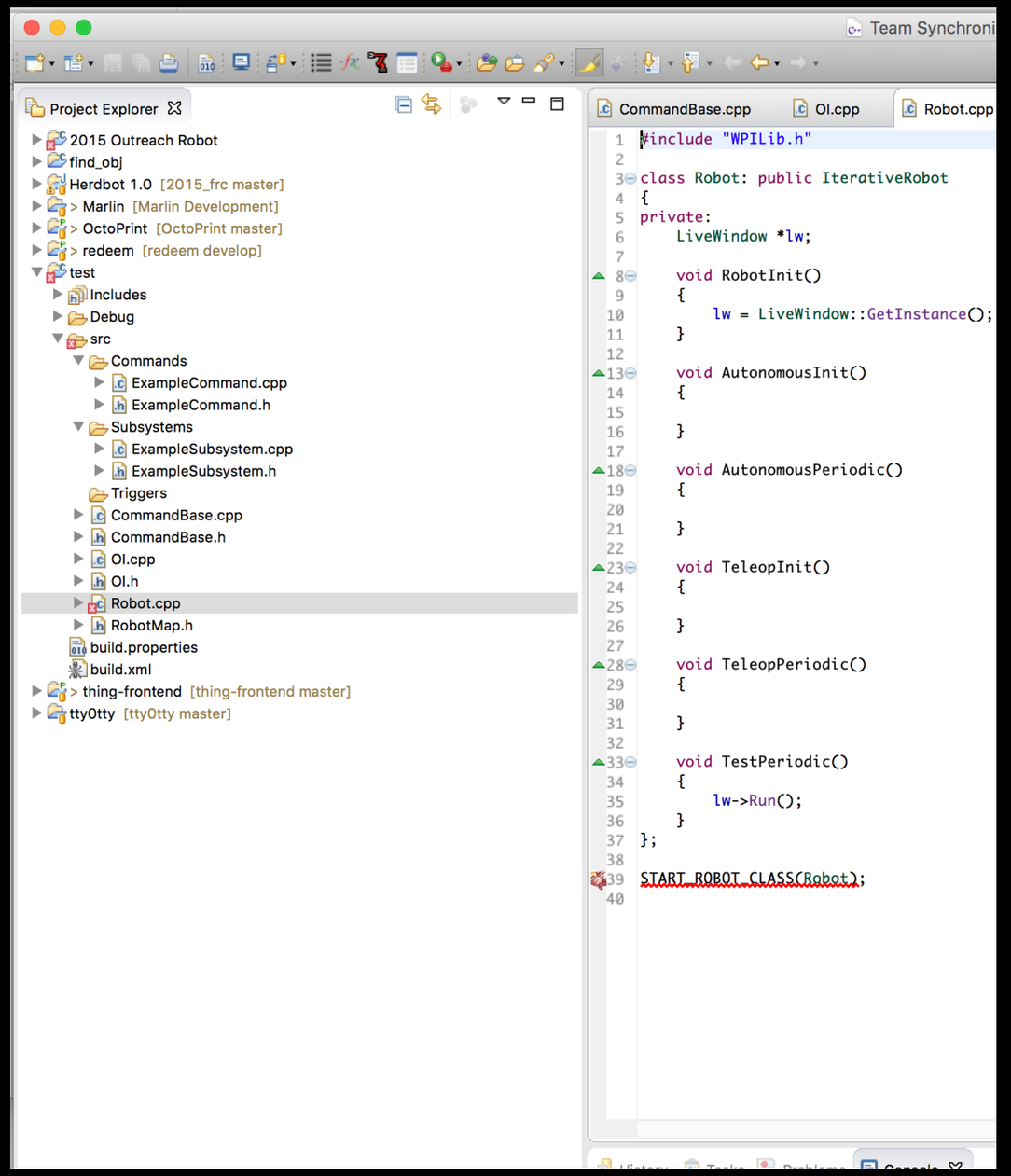
- You kinda control the logic flow.
- Provides basic logic flow.
- Robot control is mostly handled by your code.
- Four stages
  - Each stage of has two modes
    - Init - Called once at start of stage. Robot environment setup code goes here
    - Periodic - Called approximately every 20ms during stage. Robot control code goes here.
- All code in periodic stages is run when called.
- Good for basic operation with minimal complexity.
- Code readability depends on complexity.
- All code in one file.

Robot.cpp

```
1  #include "WPILib.h"
2
3  class Robot: public IterativeRobot
4  {
5  private:
6      LiveWindow *lw;
7
8      void RobotInit()
9      {
10         lw = LiveWindow::GetInstance();
11     }
12
13     void AutonomousInit()
14     {
15
16     }
17
18     void AutonomousPeriodic()
19     {
20
21     }
22
23     void TeleopInit()
24     {
25
26     }
27
28     void TeleopPeriodic()
29     {
30
31     }
32
33     void TestPeriodic()
34     {
35         lw->Run();
36     }
37 };
38
39 START_ROBOT_CLASS(Robot);
40
```

Command Based Robot - C++

- You split up and kinda control the logic flow.
- Based off of iterative.
- Good for any level of operation complexity.
- Code is broken up into separate files based on purpose.
- Code readability has a higher learning curve.
  - Might be overkill for simple robot operation.
  - Code is granularly scoped. Subjectively easier since each file has limited purpose.
- Code is broken up into small chunks based on action.
- More complex actions are built using those smaller actions.



C++

- Pros
    - Speed - In theory, compiled C++ code is faster than Java.
    - Broad support - Numerous resources available for learning C++ on the net.
    - Battle tested - C++ has existed in some form since 1979.
    - Object orientated
    - Similar in methodology to other object orientated languages - in part because other object orientated languages are strongly influenced by C++
    - Ability to incorporate wide range of 3rd party libraries.
    - Threading
- Cons
    - Requires some level of programming knowledge
    - Moderate learning curve if you are new to programming in general
    - Memory Management…….
    - Not graceful when it crashes - Can be mitigated with try / catch.
    - Debugging robot code requires additional configuration and components.
        - http://wpilib.screenstepslive.com/s/4485/m/13810/l/145321-debugging-a-robot-program
    - Threading….

Java

- Pros
    - Acceptable speed - depending on the task it can be nearly as fast as compiled C++
    - Broad support - Numerous resources available for learning Java on the net.
    - Broadly taught in schools and colleges.
    - Similar in methodology to other object orientated languages - Subjectively better implementation.
    - Ability to incorporate wide range of 3rd party libraries.
    - Memory management is mostly automatic.
    - Threading
    - Git!
- Cons
    - Requires some level of programming knowledge
    - Moderate learning curve if you are new to programming in general
    - Garbage Collector - if misconfigured can cause significant logic delays.
    - Our experience is that java was not very responsive.
    - Threading….

Eclipse

- Open source IDE
    - Integrated Development Environment
- Default FRC IDE
- Has plugin infrastructure to extend functionality
- FRC has eclipse plugins for C++ and Java
- Contains debug environment
- Available on wide variety of Operating Systems
- Integrates very well with source control
- Very well documented both in General Eclipse Usage and FRC integration
- FRC documentation
    - https://wpilib.screenstepslive.com/s/4485/m/13809/l/145002-installing-eclipse-c-java
- Is responsible for replying code to robot
- Requires toolchain installation for compiling C++ cross compilation
- Requires Java Development Kit and Tools and configuration of Java environment

GIT

www.github.com

- Git documentation comes with KOP
- Source control is a very very good thing
- Allows for distributed code creation
- Allows for code reversion and history tracking
- Allows for code branching, tagging, release and Pull Requests
- Combined with Eclipse, allows for a developer to code on any OS of their choice.
- Requires some level of training to fully take advantage
- Best practices - subjective but very well tested
    - One official repository
        - Create development and production branches
    - Each developer should create their own branch / repository of the official development branch
    - Code updates to the official development branch should be done via Pull Requests
    - Code updates to the official production branch should be done via Pull Requests from the development branch
    - Learn how to use commit / check out / clone / pull / tags / merge / rebase / Pull Requests.