Software Design Objectives

Part I
- Understand Software design process

Part II
- Understand FIRST Code Development Environment
- Understand FIRST program structure
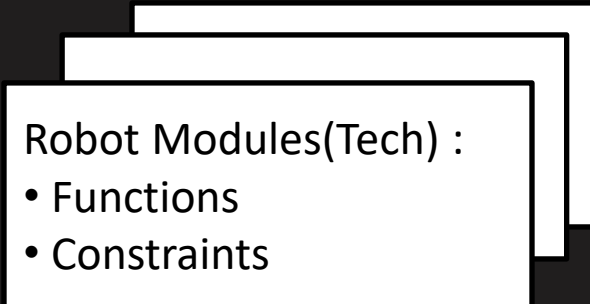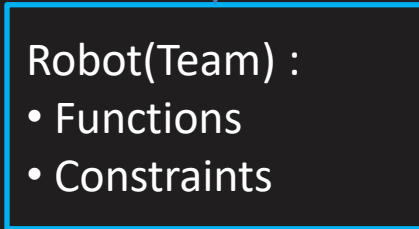- Understand FIRST I/O class library

Part III
- Understand RoboRio Programming

Part I – Software Design Objectives

- Understand the program design development process

Robot Program Development Process

KICKOFF

Robot(Team) :
- Functions
- Constraints

From the kickoff video / game manuals:
- The team develops a game strategy
- The strategy defines the functions the robot has to do and the constraints on the robot

Robot Modules(Tech) :
- Functions
- Constraints

The Technical team (Mechanical/Electrical/Software):
- Groups the functions to the major robot modules per the robot model
- For each robot module a concept and a definition of controlling devices is developed

Robot Program Development Process

Robot Modules(Tech) :
- Functions
- Constraints

Functions

Sensors / Actuators
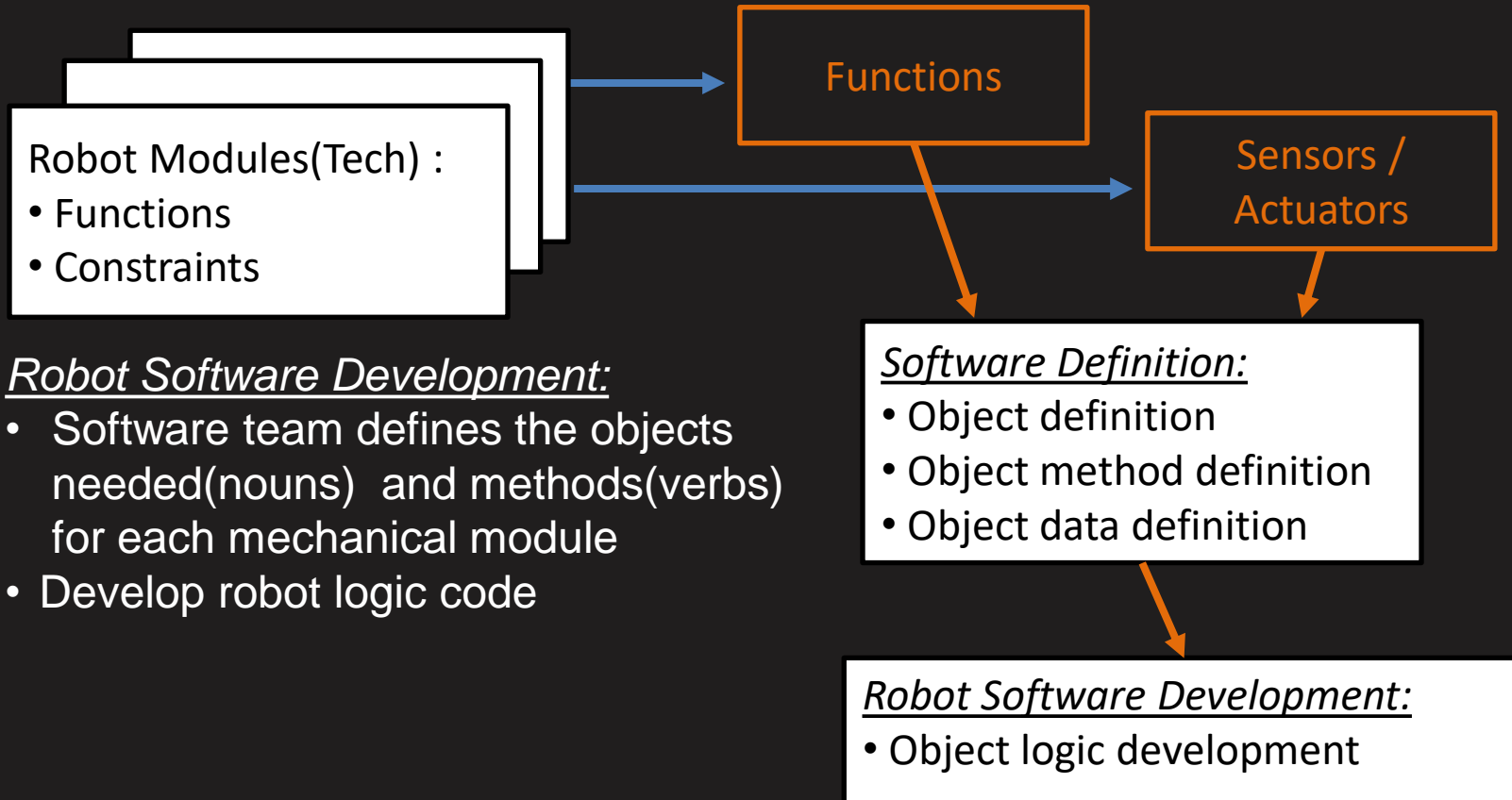
Robot Software Development:
- Software team defines the objects needed(nouns) and methods(verbs) for each mechanical module
- Develop robot logic code

Software Definition:
- Object definition
- Object method definition
- Object data definition

Robot Software Development:
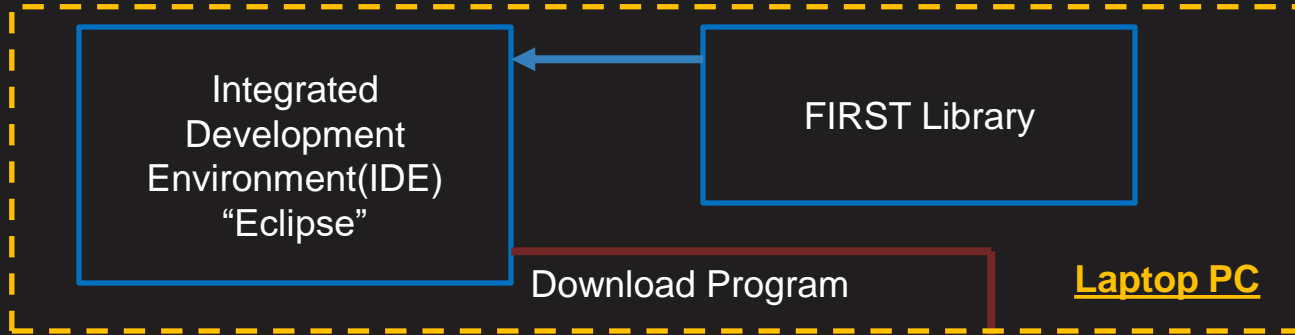- Object logic development

Software Design Process Steps

1 Determine and organize functions robot has to perform from robot functional requirements or mechanical module description.

2 Determine electrical hardware interface to computer I/O – electrical CID document

3 Develop SOFTWARE PLAN: description of logic flow

4 Develop program code per team style guide

5 Software team review of code

6 Test code: I/O functionality and then module logic
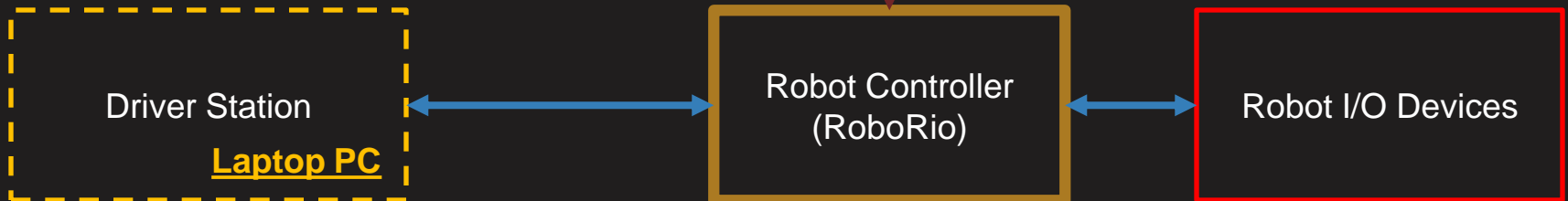
Part II FIRST JAVA Program Stucture Objectives

- Understand FIRST Code Development Environment

- Understand the JAVA program structure of FIRST

- Understand the FIRST Classes covered in the "WPILib" library

FIRST Software Development Environment

Software Development:

Integrated Development Environment(IDE) "Eclipse"

FIRST Library

Download Program

Laptop PC

Software Execution:

Driver Station

Laptop PC

Robot Controller (RoboRio)

Robot I/O Devices
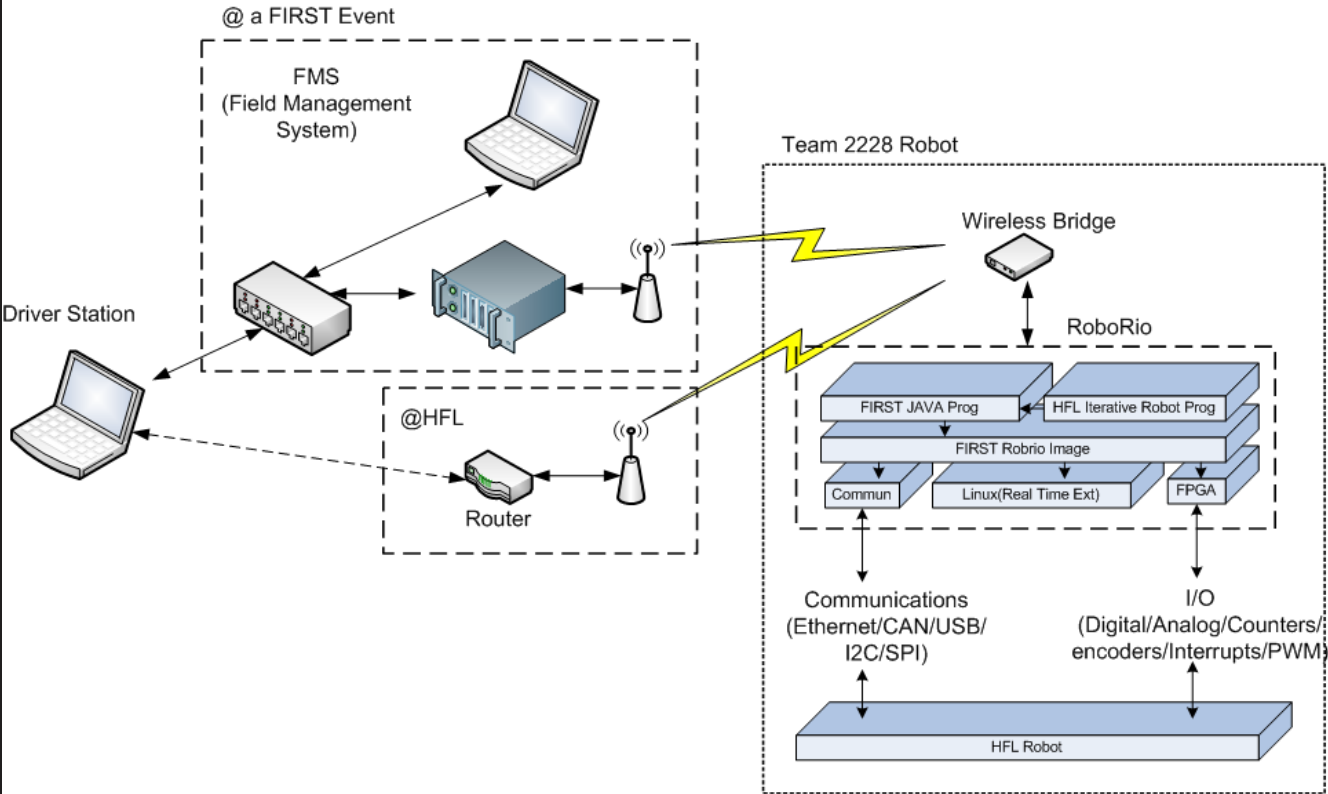
• Robot monitor
• Human input(Joy Stick / Switches)

• Robot Program

Block Dia: FIRST Hardware Structure for Software

General Program Class Structure

Robot.java

FIRST program Template - iterativeRobot

AcquisitionModule.java

Robot Module Class

OrientaionModule.java

Robot Module Class

ExecutionModule.java

Robot Module Class

RobotMap.java

Robot I/O Map

FIRST WPILib

FIRST I/O Classes

FIRST I/O Classes

FIRST I/O Classes
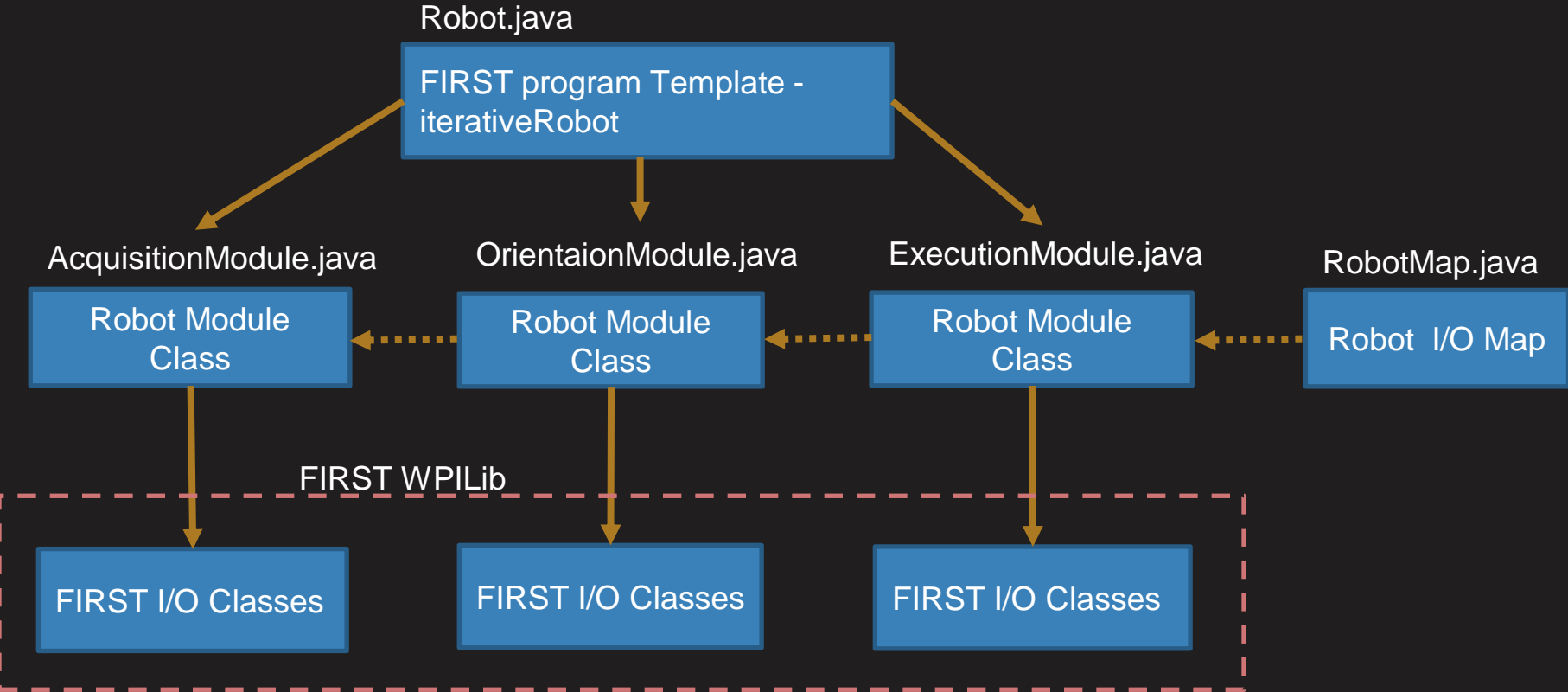
Robot Class Diagram Example
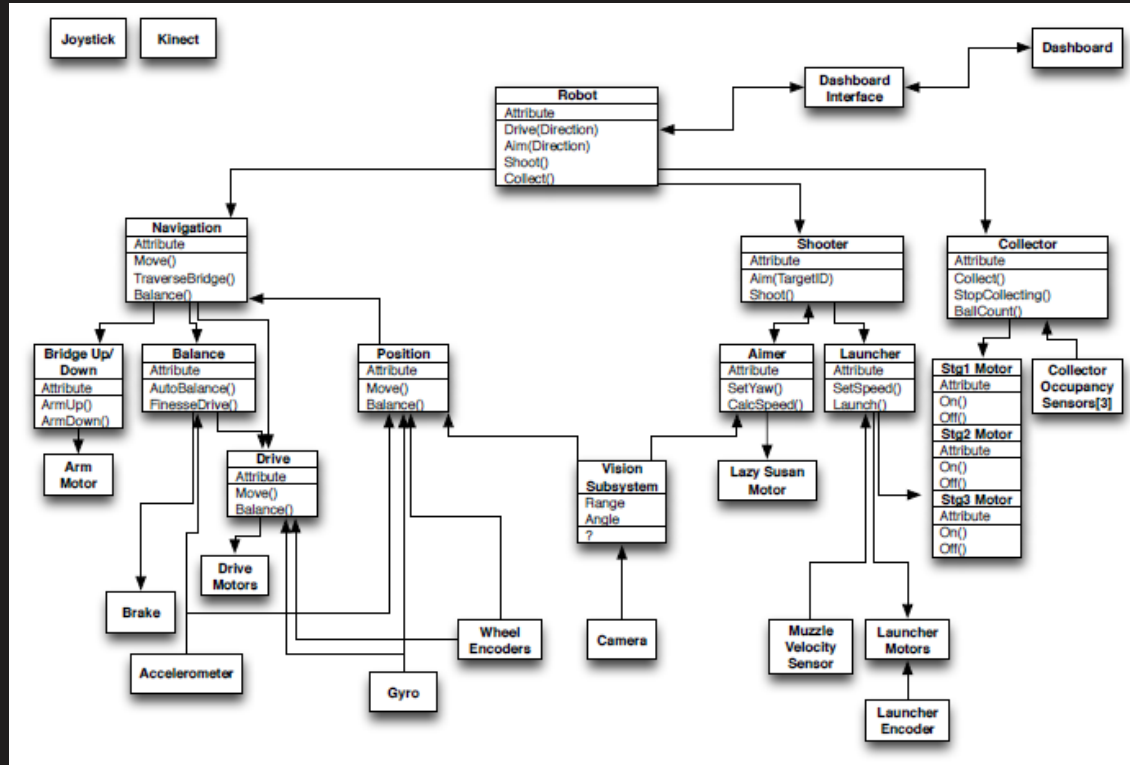
Robot.java Organization

FIRST "IterativeRobot" Main

FIRST "IterativeRobot" Extension Method Template
Class Declarations
- Declare and construct robot module objects
- Construct Object linkages if needed

Methods
robotinit()
   Team code to invoke robot modules
autonomousInit()
   Team code to invoke robot modules
autonomousPeriodic()
   Team code to invoke robot modules
teleoperatedInit()
   Team code to invoke robot modules
teleoperationPeriodic()
   Team code to invoke robot modules
TestPeriodic()
   Team code to invoke robot modules

Robot.java creates module objects and invokes module objects during competition phases

Robot.java also provides linkages between objects

Module.java Organization

Module programs contain all sequence logic for each competition phase and test code to verifiy it's I/O functionality

Module ClassTemplate
Class Declarations
   declare and construct module component I/O objects

Methods
robotInit()
  Team code for module components
autoInit()
  Team code for module autonomous initialization
autoUpdate()
  Team code for module autonomous logic
teleopnit()
  Team code for module teleoperation initialization
teleopUpdate()
  Team code for module teleoperation logic
testUpdate()
  Team code for module test logic

"IterativeRobot" Extension Template

Robot.java

```java
package org.usfirst.Rush.team2228.Robot;
import edu.wpi.first.wpilibj.IterativeRobot;
public class MyRobot extends IterativeRobot {
    Public void robotInit(){
    }
    public void autonomousInit() {
    }
    public void autonomousPeriodic() {
    }
    public void teleoperatedInit() {
    }
    public void teleoperatedPeriodic() {
    }
    Public void testPeriodic(){
    }
}
```

Robot.java Declarations Example

```java
package org.usfirst.Rush.team2228.Robot;

//Import the other files needed by the Robot Class program
import edu.wpi.first.wpilibj.IterativeRobot;
Import org.usfirst.frc.team2228.modules.Aquisition;
Import org.usfirst.frc.team2228.modules.Orientation;
Import org.usfirst.frc.team2228.modules.Execution;

public class MyRobot extends IterativeRobot{

    //Define Constants just used by IterativeRobot
    final int CONSTANT1 = 0.5; //constant comment

    //Define the variables as members of our Robot class
    int variable1; //variable comment - UNITS!

    //Define robot module object variables as members of our Robot Class
    AcquisitionModule gather;
    OrientationModule elevator;
    ExecutionModule   shooter;
}
```

Robot.java Initialization Program Example

```java
//Imports the other files needed by the program
…
public class MyRobot extends IterativeRobot{

    //Define Constants
    …
    //Define the variables as members of our MyRobot class
    …
    //Define object variables as members of our MyRobot Class

    //Initializes the variables in the robotInit method,
    //this method is called when the robot is initializing
    public void robotInit() {
        _gather = new AcquisitionModule();
        _elevator = new OrientationModule();
        _shooter = new ExecutionModule();


        // create module component objects
        _gather.robotInit();
        _elevator.robotInit();
        _shooter.robotInit();
    }
}
```

Create module objects

Have robot modules create I/O objects

Robot.java Autonomous Program Example

```java
// This method is called once each time the robot enters autonomous mode
public void autonomousInit() {

    // Each robot module will initialize for autonomous mode
    _gather.autoInit();
    _elevator.autoInit();
    _shooter.autoInit();

}

// This method is called each time the robot receives a packet(approx. 20ms) instructing
// the robot to be in autonomous enabled mode
public void autonomousPeriodic() {

        // Each robot module will update its logic for autonomous operation
        _gather.autoUpdate();
        _elevator.autoUpdate();
        _shooter.autoUpdate();


}
```

Robot.javaTeleoperation (teleop) Program Example

```java
// This method is called once each time the robot enters teleoperation mode
public void teleoperatedInit() {

    // Each robot module will initialize for teleoperation mode
    _gather.teleopInit();
    _elevator.teleopInit();
    _shooter.teleInit();

}

// This method is called each time the robot receives a packet(approx. 20ms) instructing
// the robot to be in teleoperation enabled mode
public void teleoperatedPeriodic() {

    // Each robot module will update its logic for teleoperation
    _gather.teleopUpdate();
    _elevator.teleopUpdate();
    _shooter.teleopUpdate();
}
```

"RobotMap" Class Example

The RobotMap class defines the addressing of all I/O on the RoboRio

```java
package org.usfirst.frc.team2228.robot;
/**
 * The RobotMap is a mapping from the ports sensors and actuators
 * are wired into to a variable name. This provides flexibility changing
 * wiring, makes checking the wiring easier
 */
public class RobotMap {
    //---Make RobotMap a singleton object – see software handbook

    // ---DIGITAL I/O---
    final int DIGITAL_IO_CHANNEL0 = 0;

    // For example
    final int ELEVATOR_AT_BOTTOM_SWITCH_PORT = 1;

    final int DIGITAL_IO_CHANNEL2 = 2;
    final int DIGITAL_IO_CHANNEL3 = 3;
    final int DIGITAL_IO_CHANNEL4 = 4;
    final int DIGITAL_IO_CHANNEL5 = 5;
    final int DIGITAL_IO_CHANNEL6 = 6;
}
```
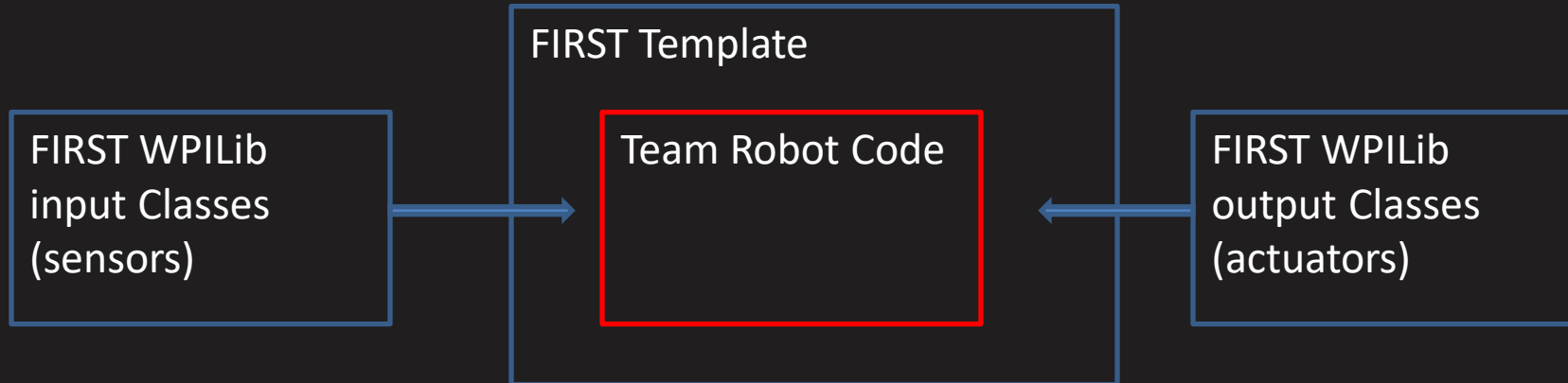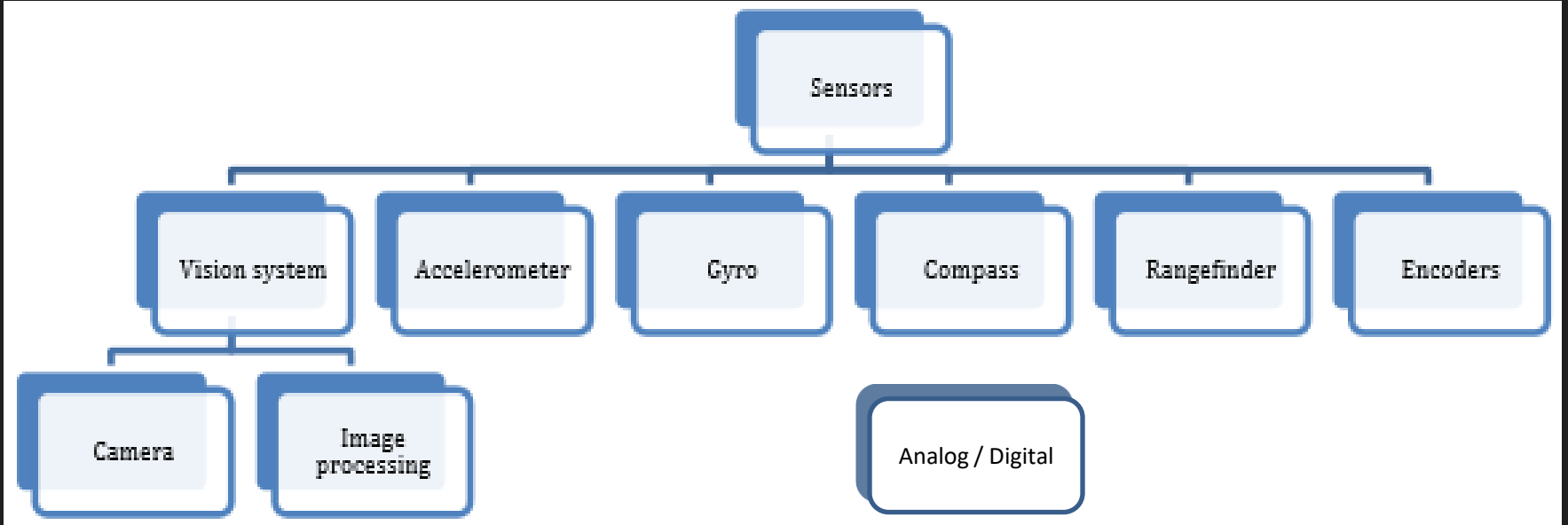
Format in Directory style:
(Constants -> in capital letters)
MODULE_DEVICE_LOCATION
and/or ACTION

FIRST JAVA Class Library

- FIRST has also done most of the "Class" work for us in a library of Classes called "WPILib"

- Team Rule: You must use FIRST Classes for I/O functions. New I/O class should be reviewed by Team leader and Mentor before developing
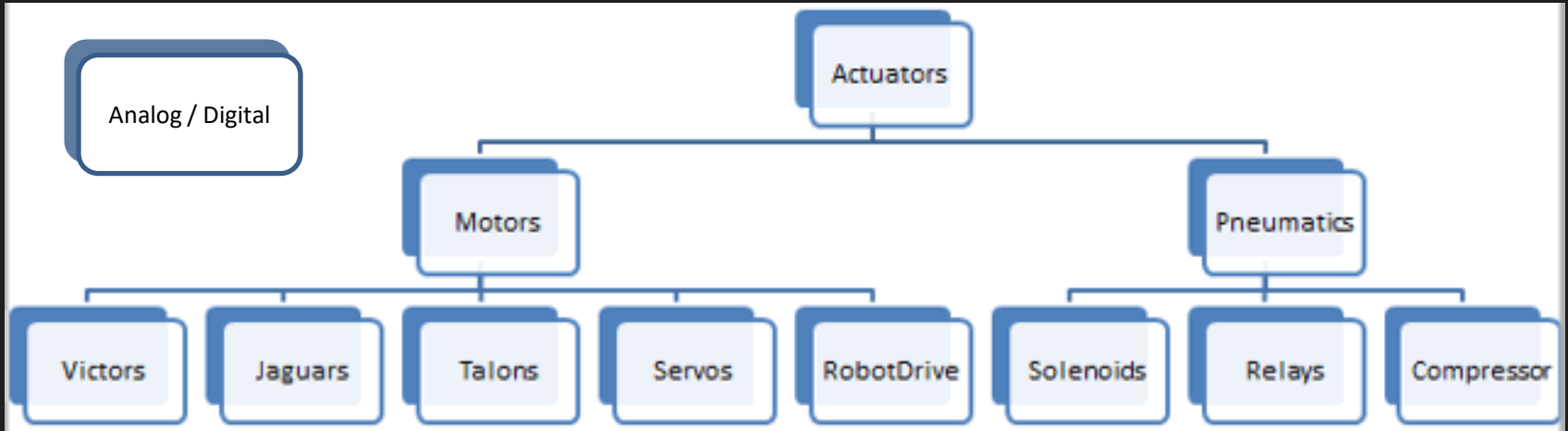
FIRST Template

FIRST WPILib input Classes (sensors) → Team Robot Code ← FIRST WPILib output Classes (actuators)

FIRST JAVA Class Library: Sensors

See CTSoft-FIRST_2015_FRC_Java_Programming.pdf

FIRST JAVA Class Library: Actuators

See CTSoft-FIRST_2015_FRC_Java_Programming.pdf

FIRST JAVA Class Library: CAN

CAN(Controller Area Network) is a communications system to pass data between control components

There are a number of CAN devices supported in the FRC control system:
- Jaguar speed controllers
- CAN-Talon speed controllers
- The Power Distribution Panel (PDP)
- The Pneumatics Control Module (PCM)

- Device status is returned every 20ms for each device automatically - the program does not need to request those updates. Whenever the program requests status from a CAN device it will be no sooner than 20ms.

- If no information is received within 50ms, the device creation will fail (either an exception in Java or an error status in C++).

See CTSoft-FIRST_2015_FRC_Java_Programming.pdf

Software Documentation

<u>Software Specification:</u>

- ❑ Definition of I/O and I/O type for each control module
  (RobotMap.java from CIM document)
- ❑ List of functions that need to be done for each control module
  (Class model)

<u>Class header:</u>

- ❑ List of all methods, how a method is called and method description
- ❑ Description of all methods
  (See Software Handbook)

<u>Method Code:</u>

- ❑ COMMENT-COMMENT-COMMENT (Why not How)
- ❑ JAVA Style Guide (syntax and readability guide – see Software Handbook)

Program Development Best Practices

Code Review:

• All code should be reviewed by software sub-team and sub-team mentor. More than the author needs to understand the code.

• Code review consists of checking the following:
- ✓ Input range check
- ✓ output range check
- ✓ Check for magic numbers (Documented? Better: Constant variable?)
- ✓ Check that any code written is already done in FIRST library
- ✓ Comments document why the program is written as it is
- ✓ Is the code backed up
- ✓ Does the code meet the robot function requirements!!!!!

Remember : There are many ways to solve a problem – The is only one question to ask: Does the code meet the requirements and comply with Team 2228 style guide?!!!

Programming Best Practices

Sensor Inputs: All sensor inputs should be checked for range values.

Actuator outputs: All method outputs should be checked for range as not to exceed the range input to actuators

Comments: There should be enough comments such that if they were grouped together anyone would understand what the program was supposed to do

Failures: All code should consider what to do in a failure condition. For example: variable out of range, stuck in a loop, external processes take too long due to failure, communications errors, etc

NO MAGIC NUMBERS: All constant numbers should be associated with a constant name

Programming Style: Follow the programming style guide in the handbook..

A consistent style is beneficial when other members of our team and other teams read your code

Program Execution Safety

- On powering up the robot check that the robot software mode is **"DISABLED"**

- Check to see that the robot wheels are off the ground

- On enabling the robot software, the person enabling the robot software needs to **HOLLER** – **"ROBOT ENABLING"** and then **WAIT** for a **"CLEAR"** response from the assigned robot safety observer. The person enabling then responds: **"ROBOT ACTIVE"**

- Someone needs to be within reach of the robot **"DISABLE"** button to kill the robot program.

- Safety glasses must be worn whenever the battery is connected to the robot or any electronic devices.
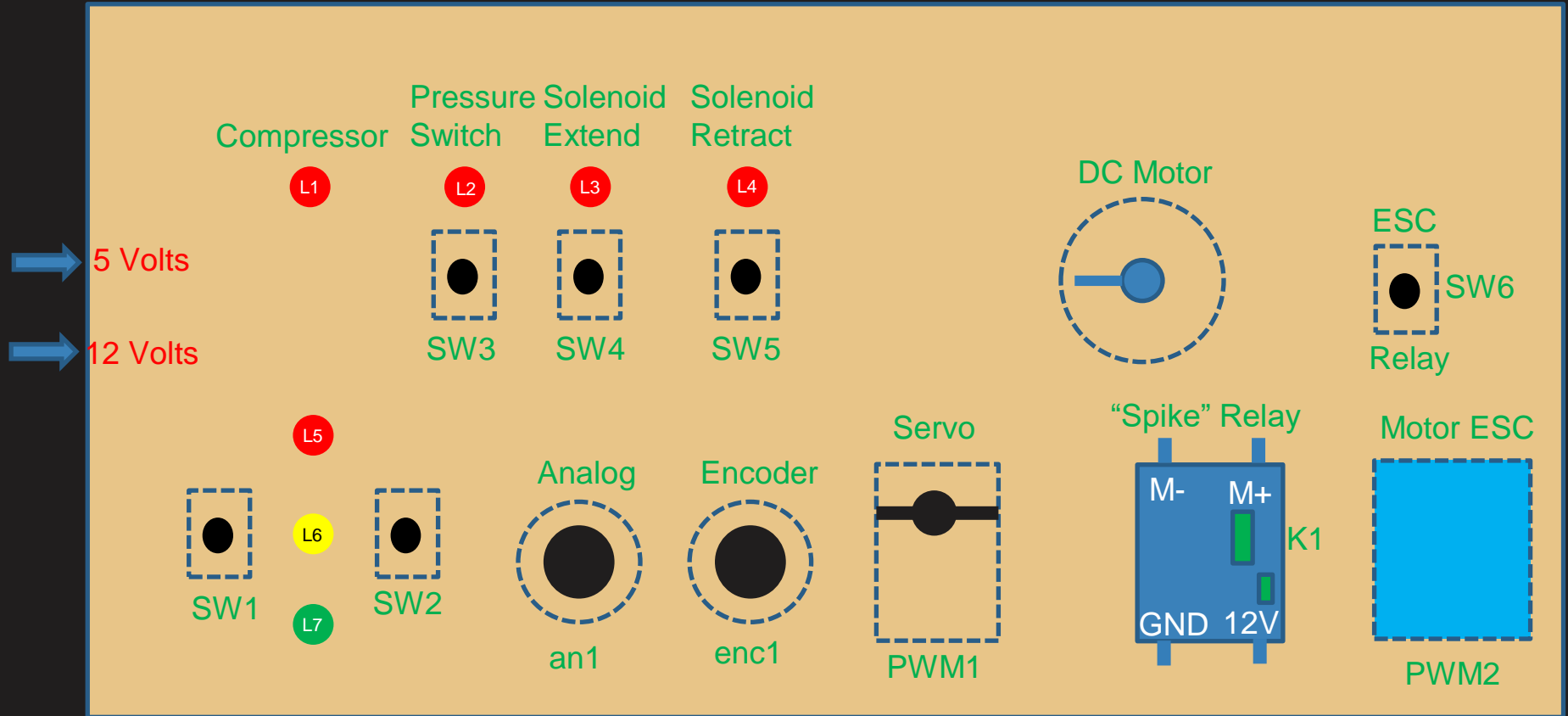
Part III RoboRio Programming

- Understand how to develop and execute a program

- Understand how to program I/O via a I/O test fixture

RoboRio Code Development and Execution

- Open software handbook and go to section:
  "Creating and Running Robot Programs"

- Link to FIRST documents and follow instructions

RoboRio I/O Test Fixture

FIRST JAVA Robot Program Lab

➢ Program Digital I/O

❑ Develop class module to read test panel traffic light switch SW1 and turn on LED "L6" (Red) on Test Panel

❑ Develop class module to turn on traffic light LEDs on Test Panel, then sequence LEDs with a delay between each LED activation

➢ Program Encoder Digital Input

❑ Develop class module to light LED "L6" (Red) for an "A" input and light LED "L8" (Green) for a "B" input.

➢ Program Analog I/O

❑ Develop class module to read "Analog" potentiometer from Test Panel

FIRST JAVA Robot Program Lab

➢ Program Relay I/O
- ❑ Develop class module to activate "Spike" relay forward and reverse on Test Panel. Put switch SW6 in "ESC" mode(up).
- ❑ Develop class module to run motor in forward and reverse. Put switch SW6 in "Relay" Mode(down)

➢ Program PWM I/O
- ❑ Develop class module to set speed to "DC motor" from Test Panel using "Motor ESC". Put switch SW6 in "ESC" mode(up).

➢ Program PWM I/O
- ❑ Develop class module to move "Servo" left and right on Test Panel

FIRST JAVA Robot Program Lab

- Program Traffic Light State Machine
  - x
- Program ramp up – ramp down speed control
  - x
- Program speed control via joystick
  - x
- Program LEDs via joystick push buttons
  - x

FIRST JAVA Robot Program Lab

- ➢ Program SRX speed controller
    - ❏ x
- ➢ Program SRX encoder input
    - ❏ x

Revisions

V161208 – RJV; added robot class dia example

V160902 – RJV; updated test IO panel layout, RoboRio code exercises

V160503- RJV, updated with robot.java program construction

V151018 – RJV, Review update

V151001 – RJV, updated JAVA program to match car example, added software review

V150918 – RJV –re-wrote class-object discussion, renamed training module

V150914 – RJV-added software documentation slide

V150906 - Original