

O'REILLY®

# OSCON™

Open Source Convention



## Improving the Embedded Development Process

Matt Fletcher, Developer

William Bereza, Co-founder

Atomic Object, LLC

{fletcher, bereza}@atomicobject.com

OSCON 2007 Room D137-138

# About this presentation

- Test-driven development of embedded systems
- Target audience is developers
- Audience is familiar with test-driven development (TDD)



# Atomic and Savant Automation

- Atomic Object develops custom software

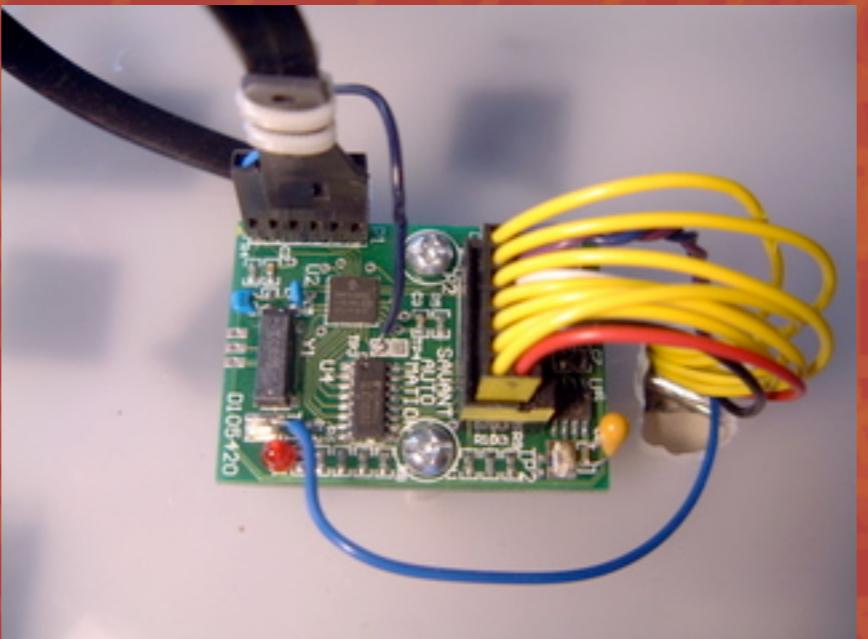


- Savant Automation produces automated-guided vehicles



# Battery monitor board

- Monitors the onboard battery's overall state of charge



- Vehicle's main computer decides when to charge or get back to work



# Feature development

Choose a new feature



# Feature development

Develop a system test for it



# Feature development

See the system test fail



# Feature development

Create unit tests for a module



# Feature development

See the unit tests fail



# Feature development

Make the tests pass



# Feature development

Develop modules as needed



# Feature development

See system test pass



# Feature development

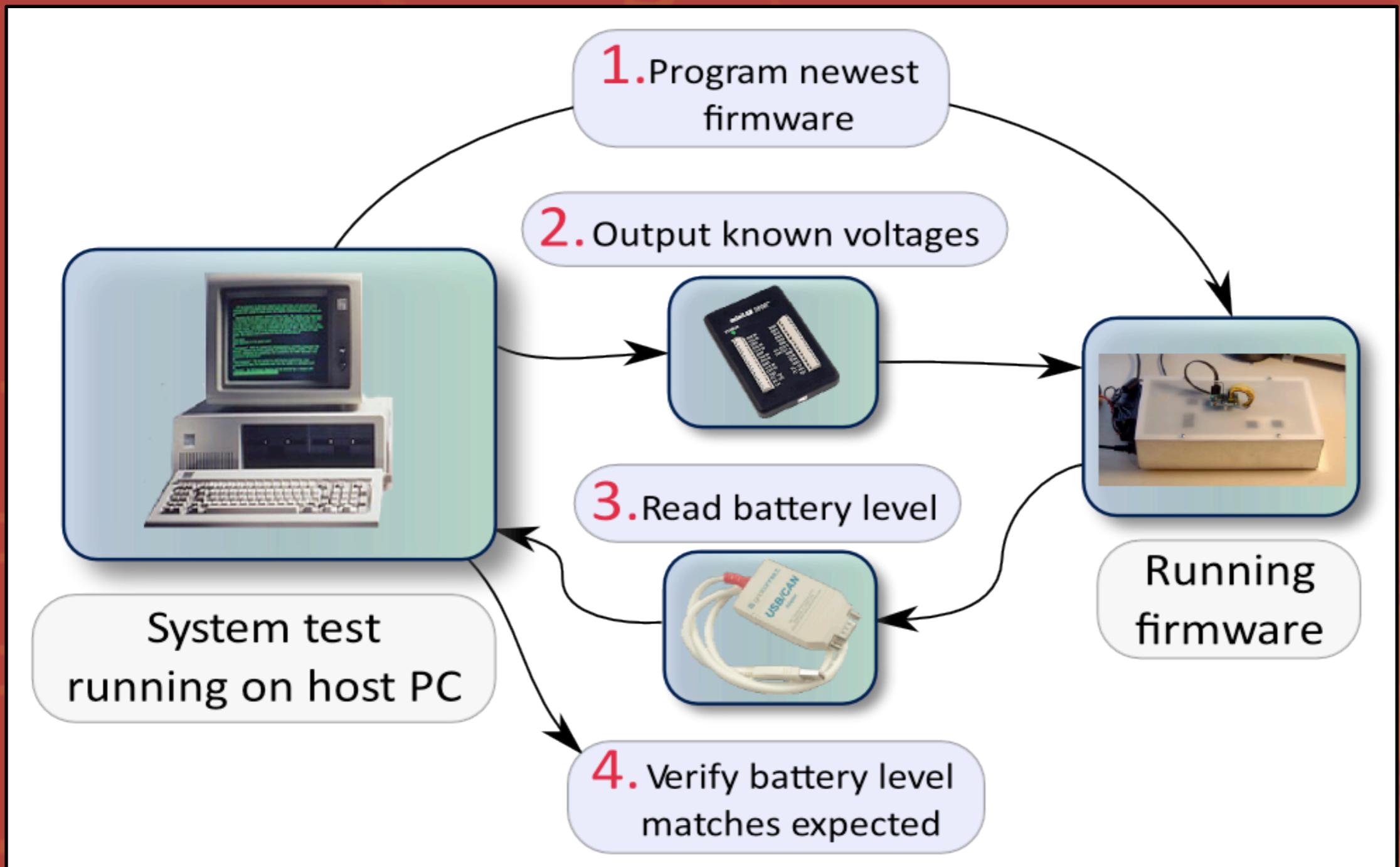
Choose the next feature...



# Example system tests

- Battery level output based on input voltages
- Battery level mode vs. diagnostic data mode
- Calibration commands adjust battery level values





# Battery voltage test

proves "the battery voltage reported via CAN is correct."

```
set_battery_voltage 51.0
verify_can_reports_battery_voltage 51.0
```

```
set_battery_voltage 2.0
verify_can_reports_battery_voltage 2.0
```



# Under the covers

```
set_battery_voltage 51.Q
verify_can_reports_battery_voltage 51.Q
```



```
def set_battery_voltage(voltage)
  @minilab.write_analog(VOLTAGE_OUTPUT_PIN,
                        voltage / BATTERY_VOLTAGE_DIVIDER)
  sleep 0.1
end
```



# Under the covers

```
set_battery_voltage 51.0
verify_can_reports_battery_voltage 51.0
```



```
def verify_can_reports_battery_voltage(voltage)
    last_message = @pcan.receive_message
    assert_in_delta(voltage / CAN_VOLTAGE_BIT_WEIGHT,
                    last_message.BatteryVoltage,
                    CAN_VOLTAGE_DELTA)
end
```



# Keys to system testing

- Test automation
- Choosing support hardware with command line and programmatic interfaces
- Diligence. System test-first!



# Develop code unit test-first

- Interaction-based testing
- Modules are composed with delegates
- Tests compose the target module with mocks
- Composition is done at link time



# Rake

- Ruby make
- Automates building units and executables
- Easy injection of Ruby scripts into the build system



# Argent

- Ruby inline code generation
- Finds and sets up tests
- Invoked against files by the build system



# Generated mocks

- Mocks created automatically
  - Ruby script scans header file
  - Creates mock header and source



# Generated mocks

- Regular expression used to parse functions

```
FUNC_MAGIC =/( \w*\s+)*(\w+)\s+(\w+)\s*\(((^\s)]*)*)\)\)/
```

- Naming conventions dictate how to declare functions and how to call mock functions



# Code to be mocked

MessageCreator.h

CANMessage MessageCreator\_GetNextMessage(void)

MessageSender.h

void MessageSender\_SendMessage(CANMessage message)

StatusOutputer.c

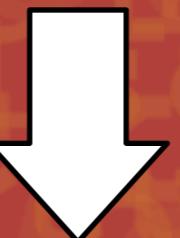
Needs to send the latest messages!



# Mock with return

MessageCreator.h

CANMessage MessageCreator\_GetNextMessage(void)



MockMessageCreator.h

CANMessage MessageCreator\_GetNextMessage(void)

void MessageCreator\_GetNextMessage\_Return(CANMessage toReturn)



# Mock expecting a value

```
MessageSender.h
```

```
void MessageSender_SendMessage(CANMessage message)
```



```
MockMessageSender.h
```

```
void MessageSender_SendMessage(CANMessage message)
```

```
void MessageSender_SendMessage_Expect(CANMessage message)
```



# The test!

```
static void testShouldSendTheNewestMessage(void)
{
    CANMessage someMessage = {0x0b, 0x0e, 0x0e, 0x0f};
    MessageCreator_GetNextMessage_Return(someMessage);
    MessageSender_SendMessage_Expect(someMessage);

    StatusOutputter_SendNewestMessage();
}
```

Mocks are automatically verified during tearDown



# The code!

```
void StatusOutputter_SendNewestMessage(void)
{
    CANMessage newestMessage;
    newestMessage = MessageCreator_GetNextMessage();
    MessageSender_SendMessage(newestMessage);
}
```



# Continuous integration

- Automatically builds the system and runs tests on check-in
- Build status is published to monitors throughout the office



# Summary

- Automated system testing
- Automated unit testing
- Automated build system
- Continuous integration
- Ruby scripts and extensions played a key role!



# Embedded development - fun

Imagine frustrated engineer vs. a productive pair



# Build your own tools

- Solve specific problems with custom tools
  - Automate tedious processes
  - Generate repetitive code



# Resources

- Rake
  - [rake.rubyforge.org](http://rake.rubyforge.org)
- Argent
  - [rubyforge.org/project/argent](http://rubyforge.org/project/argent)
- Minilab Ruby gem
  - [minilab.rubyforge.org](http://minilab.rubyforge.org)



# Resources

- Atomic Object embedded
  - [www.atomicobject.com/pages/Embedded+Software](http://www.atomicobject.com/pages/Embedded+Software)
  - papers
  - Embedded System Conference 2007 demo project
- Agile Embedded Yahoo! group:
  - [tech.groups.yahoo.com/group/AgileEmbedded](http://tech.groups.yahoo.com/group/AgileEmbedded)



# Questions?

