Growing Embedded Applications Organically with Ceedling and Friends

Greg Williams

ATOMIC EMBEDDED

Embedded Development...

- Limited Memory
- Limited Processing Power
- Language Limitations
- Short Timelines
- Growing Complexity
- Non-Standard Hardware
- Is HARD

What Can We Do?

- Give up?
- Pray?
- Complain?
- Get a new job?!?
- Get better!
- Be responsible!

NO SILVER BULLET

Testing Success

- Smart and Capable People
- Good Process
- Tools to Support Good People and Good Process

Validation

- Need to verify we have done the right thing
- What is the right thing?
- How do we check it?
- Will it always work?
- Is it worth it?
- But it takes to much F!$%#$G time!!!

Manual Validation

- Real Deal / Simulation
- Stimulate conditions / Modify state
- Run to breakpoint
- Check resultant state
- Tedious
- Painful
- Usually do it ONCE

Automation

- Buy a robot!!!
- Create test plans
- Buy fancy hardware
- Spend a lot of time figuring out what and how
- How can we automate everything??
- $$$$$$$

Unit Testing

- Focus on testing individual modules and functions
- Verify that a given scenario produces the correct result
- Ensures building blocks perform specifc operations according to their design
- Drives toward proper encapsulation
- Design can evolve naturally...

Unit Testing

- Automatable
- Provide living documentation of design
- Instant regression testing
- Facilitates refactoring
  - Eliminate dead code
  - Eliminates: “Don’t fix it if it ain’t broke” or “We’ll fix that next time we touch it”
  - Ahhhhhhhhhh.....

What is TDD?

- What do we mean by Test-Driven Development?
  - Mindset of Maximizing Testability
  - First executable code is TEST code
  - Write JUST ENOUGH code to satisfy tests

The TDD Cycle

- Select a Feature to Implement
- Write a Small Test
- Execute Test and Watch It Fail
- Implement source code to satisfy test
- Correct until the tests pass
- REPEAT, REPEAT, REPEAT...

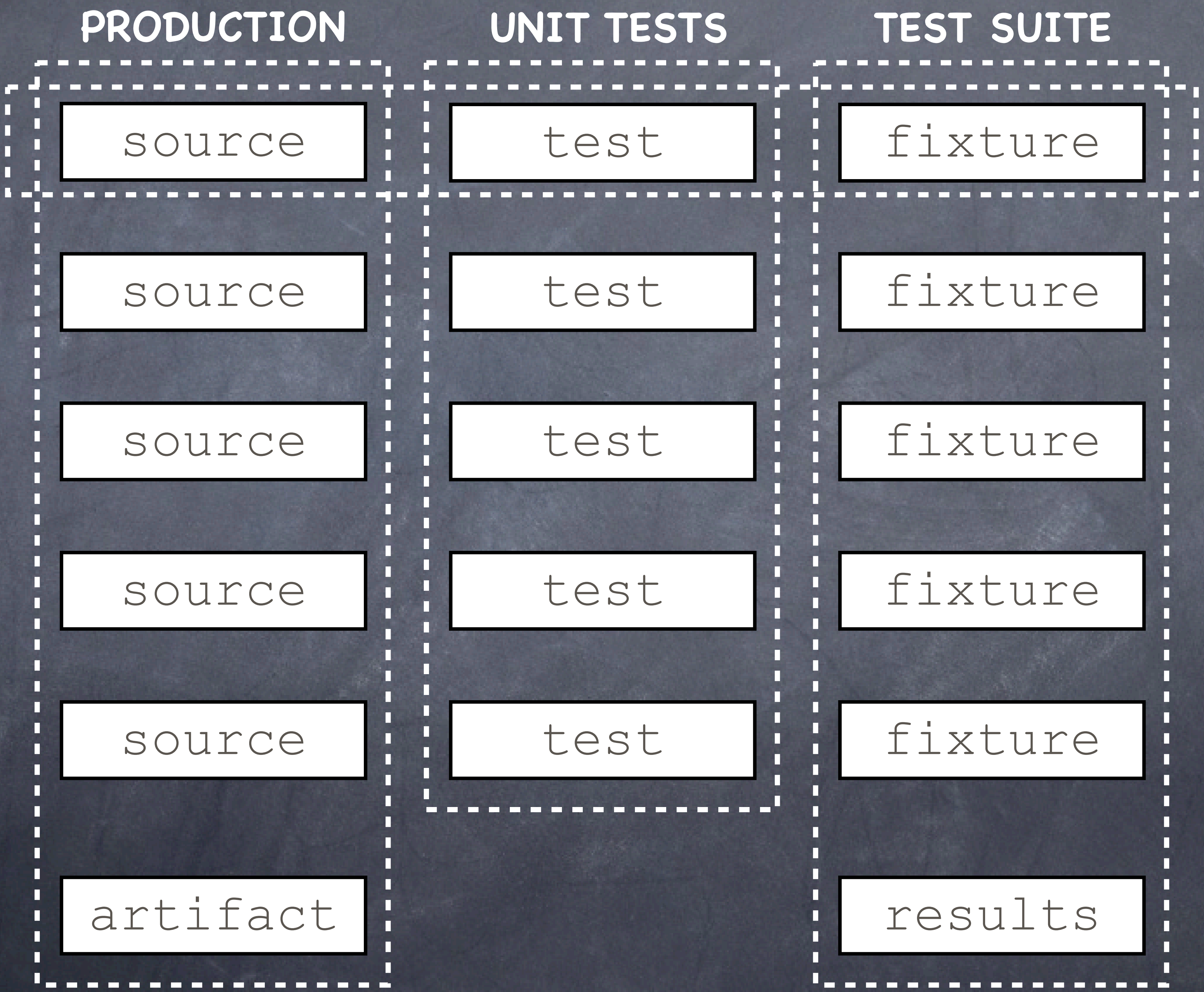
Types of Tests

- Unit Tests
  - Exercises a unit of source code
  - Executes unit in isolation from surroundings
- Integration Tests
  - Same fundamentals as unit tests, but exercises a group of modules / subsystem
- System Tests
  - Run against full application on target hardware
  - Exercises behavior that unit/integration tests cannot
  - Ideally targeted at validating FEATURES

Benefits of TDD

- Tested Software
  - High Level of Code Coverage
  - Full Coverage Measured by Coverage Tools
- Well-Designed Software
- Well-Documented Software
- Maintainable Software
- Sanity!!!
- Shiny Happy People...

Mechanics: Testable Project



Where to start...

Project Scope:

Measure temperature (thermistor voltage) and report degrees Celsius once per second over serial port

Time to plant a Ceedling...

```
> gem install ceedling
> ceedling new MyProject
Project 'MyProject' created!
 - Tool documentation is located in vendor/ceedling/docs
 - Execute 'rake -T' to view available test & build tasks
> cd MyProject
> rake test:delta
-------------------------
OVERALL UNIT TEST SUMMARY
-------------------------


No tests executed.
>
```

DONE!

Feature Request / Task

Read Analog to Digital Converter
X times per second

Let's GO!

- Dig through datasheet
- ADC_Init()
    - Setup ADC in proper mode
- ADC_Read()
    - Trigger a conversion and wait for completion
    - Return the results
- Wire it into the system
    - We still need to time the samples
- WAIT!!!
- Where do the results go? Hmmmm......

Feature-Driven Development

- **"We need to read something from and ADC converter, so let's write a driver!!"**
  - NOOOOOOOOOOOOOOOOOOOO
  - Leads to cluttered APIs and DEAD CODE!
- Focus on what is required NOW, and implement ONLY THAT
- Use a top-down approach, discovering needs along the way

Feature-Driven Development

(continued...)

- Software IS features
- Customers pay for features, NOT infrastructure
- Infrastructure-First => WASTE

- Feature-Driven Development
  - Minimally working system as soon as possible
  - Build towards feature completion
  - Tests are a safety net while refactoring
- Features yield meaningful progress metrics
  - Satisfied project managers AND developers

Top-Down Design

- Mocks to the rescue!
- CMock generates mocks using only header files (INTERFACES)
- The lower levels need not be implemented AT ALL!!
- Leads to easily refactored interfaces prior to implementation of underlying code

CMock

- Creates mocks of modules using only the header files (interfaces)
- Utilizes Ruby to make the magic happen
- Creates helper methods
- Verifies interactions with other modules and libraries

CMock Example

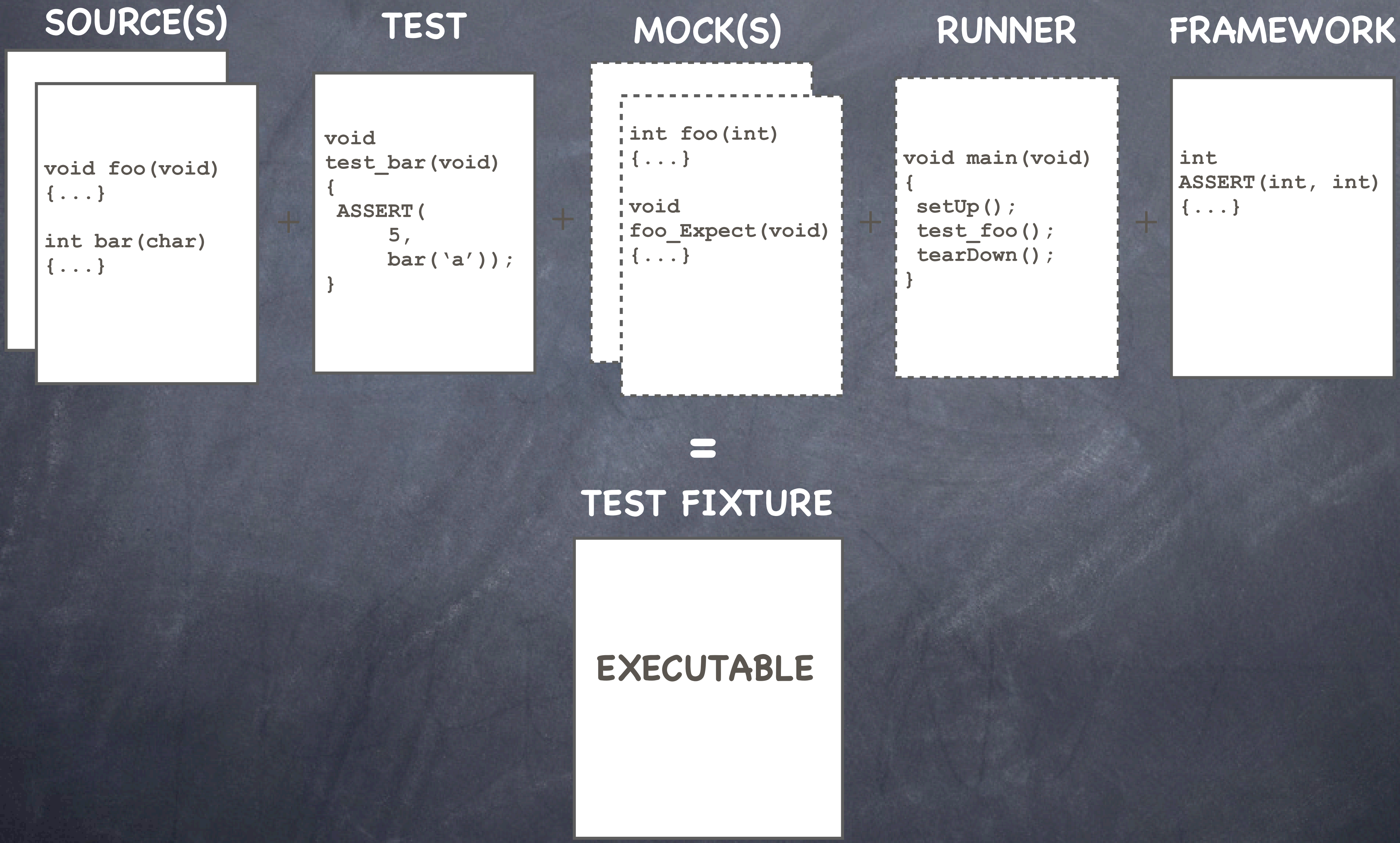
```
ARGS* ParseStuff(char* Cmd);
void  HandleNeatFeatures(NEAT_FEATURE_T NeatFeature);
```

```
int  ParseStuff(char* Cmd);
void ParseStuff_ExpectAndReturn(char* Cmd, int toReturn);
void ParseStuff_IgnoreAndReturn(int toReturn);
void ParseStuff_StubAndCallback(CMOCK_ParseStuff_CALLBACK Callback);

void HandleNeatFeatures(NEAT_FEATURE_T* NeatFeature);
void HandleNeatFeatures_Expect(NEAT_FEATURE_T* NeatFeature);
void HandleNeatFeatures_ExpectWithArrays(NEAT_FEATURE_T* NeatFeature,
                                         int NeatFeature_Depth);
void HandleNeatFeatures_Ignore(void);
void HandleNeatFeatures_StubAndCallback(CMOCK_HandleNeatFeatures_CALLBACK
                                        Callback);
```

```
void test_MyFunc_should_ParseStuffAndCallTheHandlerForNeatFeatures(void)
{
    NEAT_FEATURES_T ExpectedFeatures = { 1, "NeatStuff" };

    ParseStuff_ExpectAndReturn("NeatStuff", 1);
    HandleNeatFeatures_Expect(ExpectedFeatures);

    //Run Actual Function Under Test
    MyFunc("NeatStuff");
}
```

```
void MyFunc(char* Command)
{
  int ID;
  NEAT_FEATURES_T Neat;

  ID = ParseStuff(Command);
  switch(ID)
  {
    case 0:
      HandleStupidFeatures();
      break;
    case 1:
      Neat.id = 1;
      Neat.cmd = Command;
      HandleNeatFeatures(Neat);
      break;
    default:
      break;
  }
}
```

Anatomy of a Ceedling Test



```
void foo(void)
{...}

int bar(char)
{...}
```

```
void
test_bar(void)
{
 ASSERT(
     5,
     bar('a'));
}
```

```
int foo(int)
{...}

void
foo_Expect(void)
{...}
```

```
void main(void)
{
 setUp();
 test_foo();
 tearDown();
}
```

```
int
ASSERT(int, int)
{...}
```

Ceedling Quick Ref

- rake -T
  - List all tasks
- rake test:my_module
  - Test the specified module
  - Also can specify test, header or source
- rake test:all
  - Test all modules
- rake test:delta
  - Test changes (incremental)

Questions?