TDD for Embedded C

Topics

• The why and what of TDD
• Embedded adaptation
• Abstracting HW and OS
• Mocking the silicon
• Test-double substitution options
• Function pointer substitution
• Preprocessor substitution
• TDD next to an RTOS
Why TDD?  
What is TDD?

This Work Flow is Designed to Allow Defects

Development

Test

Defects
Your program will have bugs. And they will surprise you when you find them.

The Physics of Debug Later Programming (DLP)

- As $T_d$ increases, $T_{find}$ increases dramatically
- $T_{fix}$ is usually short, but can increase with $T_d$
Edsger Dijkstra

Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result, the programming process will become cheaper. If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.
Can we Realize Dijkstra's Dream and Prevent Defects with Test Driven Development?

Being good at chasing bugs is not Technical Excellence
A Complex system that works is invariably found to have evolved from a simple system that worked.
The Physics of Test Driven Development

- When $T_d$ approaches zero, $T_{find}$ approaches zero
- In many cases, bugs are not around long enough to be considered bugs.
- See: http://www.renaissancesoftware.net/blog/archives/16

Why is it difficult to sustain?

TDD:
- Steady pace
- Speed limits
- No traffic lights (bugs)
- Might feel slow!!

Traditional:
- Spurts
- Fast when no problems!
- Debugging
- Feels fast! But often slow
Sustainability

Traditional development

Time development

Test-driven development

Time

Feels slower

Speculate  Code  Test  Debug

Feels slower

Speculate  Test and Code  Debug

Copyright © 2008-2012 James W. Grenning
All Rights Reserved. For use by training attendees.

Agile 2012, Grapevine, TX

www.renaissancesoftware.net
james@renaissancesoftware.net
basv@odd-e.com

Demo and Exercise
CppUTest Quick Intro

Cheat sheet

/* in CheatSheetTest.cpp */
#include "CppUTest/TestHarness.h"

/* Declare TestGroup with name CheatSheet */
TEST_GROUP(CheatSheet)
{
    /* declare a setup method for the test group. Optional. */
    void setup()
    {
        /* Set method real_one to stub. Automatically restore in teardown */
        UT_PTR_SET(real_one, stub);
    }

    /* Declare a teardown method for the test group. Optional */
    void teardown()
    {
    }
}; /* Do not forget semicolon */

/* Declare one test within the test group */
TEST(CheatSheet, TestName)
{
    /* Check two longs are equal */
    LONGS_EQUAL(1, 1);

    /* Check a condition */
    CHECK(true);

    /* Check a string */
    STRCMP_EQUAL("HelloWorld", "HelloWorld");
}

/* In allTest.cpp */
IMPORT_TEST_GROUP(CheatSheet);

/* In main.cpp */
#include "CppUTest/CommandLineTestRunner.h"
#include "AllTests.h"

int main(int ac, char ** av)
{
    #include "CppUTest/CommandLineTestRunner.h"
    #include "AllTests.h"

    int main(int ac, char ** av)
    {
        return CommandLineTestRunner::RunAllTests(ac, av);
    }
}
Test Fixture

```cpp
TEST_GROUP(TemplateEngineTest) {
    Template aTemplate;
    TemplatePlaceholderValues replacementValues;
    EmailFormatter *emailFormat;
    void setup() {
        emailFormat = createMockFormatter();
        aTemplate.setEmailFormat(emailFormat);
    }
    void teardown() {
        destroyFormatter(emailFormat);
    }
}
TEST(TemplateEngineTest,
templatesWithoutPlaceHoldersDoNotChange) {
    aTemplate.set("Nothing here");
    STRCMP_EQUAL("Nothing here",
                 aTemplate.replaceValues(replacementValues).c_str());
}
```

Run before each test

Run after each test

The data and setup / teardown is also sometimes called: test fixture

Data needed in each test.

Output Format

No news is good news

```
./TestFirst
OK (1 tests, 1 ran, 1 checks, 0 ignored, 0 filtered out, 0 ms)
```

Give precise information when fails

```
./TestFirst
TestFirst.cpp:10: error: Failure in TEST(FirstTest, First)
make: *** [all] Error 1
  expected <1 \0x1>
  but was <0 \0x0>

Errors (1 failures, 1 tests, 1 ran, 1 checks, 0 ignored, 0 filtered out, 0 ms)
```
More info

• CppUTest Github Project page:
  – http://cpputest.github.com/cpputest/

• The CppUTest Github repository:
  – https://github.com/cpputest/cpputest

• The CppUTest Man page:
  – http://www.cpputest.org/

Adaptation for Embedded Software
What are the special challenges you have over a non-embedded developer?

- Concurrent HW development
- HW bottleneck
- Cross compilation
- Limited memory and IO
- Target debug
- Architecture
Hardware is Scarce!

- It does not exist.
- It is being used by someone else.
- It has bugs of its own.
Minimize Debug On Hardware

Copyright Fox Broadcasting. Used under fair use.
Why Use Your Development System for a Test Bed?

• Waiting for hardware.
• Waiting for restart.
• Waiting for downloads.
• Waiting for long compiles.
• Debugging on the target.
• Target has bugs.
• (Re)Configuring the lab

Avoid wasteful practices

How to Use Your Development System for a Test Bed?

• Multi-targeted code.
• Must make code portable
• Must beware of hardware and OS dependencies.
• Object Oriented approach to Dependency Management.
But There are Risks with Development System Tests

- Architecture differences
  - Word size
  - Big-endian Little-endian
  - Alignment
- Compiler differences
- Library differences
- Execution differences

TDD Adaptation for Embedded Development

Stage 1

Write a Test
Make it Pass
Refactor

More Frequent

Less Frequent
TDD Adaptation for Embedded Development

Stage 1
- Write a Test
- Make it Pass
- Refactor

Stage 2
- Compile for Target Processor

Stage 3
- Run Tests in the Eval Hardware

Stage 4
- Run Tests in Target Hardware

Stage 5
- Acceptance Tests

More Frequent

Less Frequent

See: [http://renaissancesoftware.net/files/articles/ProgressBeforeHardware.pdf](http://renaissancesoftware.net/files/articles/ProgressBeforeHardware.pdf)

Continuous Integration - Embedded

1. Developer has all unit tests

2. Developer checks in work regularly.

3. CIS notices changes, refreshes its local copy, builds and runs host

4. After successful host build/test, CIS builds and hands images to TMS to deploy to simulator, eval/ reference boards or target system.

The SCR, CIS and TMS are not necessarily separate hardware systems.
Problems Found at Appropriate Stage

<table>
<thead>
<tr>
<th>Stage</th>
<th>Problems Likely to Find in Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Logic, design, modularity, interface, boundary conditions</td>
</tr>
</tbody>
</table>
| 2     | Compiler compatibility (language features)  
       | Library compatibility (header files, declarations) |
| 3     | Processor executions problems (compiler and library bugs)  
       | Portability problems (word size, alignment, endian) |
| 4     | Ditto stage 3  
       | Hardware integration problems  
       | Misunderstood hardware specifications |
| 5     | Ditto stage 4  
       | Misunderstood feature specification |

Hardware and OS Abstraction
Separation of Responsibilities

- Every minute, the RTOS wakes up the Light Scheduler.
- If it is time for one of the lights to be controlled, the LightController is told to turn on/off the light.

Light Scheduler Design

- Program to Interfaces
- Separate interface and implementation as separate entities.
- This design has good separation of responsibilities
LightScheduler Test Fixture Design

- Use the real collaborators if you can.
- Use fakes when you must.

TEST(LightScheduler, light_not_changed_at_the_wrong_time)
{
    LightScheduler_ScheduleTurnOn(3, EVERYDAY, 1200);
    FakeTimeService_SetMinute(1199);
    LightScheduler_Wakeup();
    LONGS_EQUAL(NO_LIGHT_ID, LightControllerSpy_GetLastId());
    LONGS_EQUAL(LIGHT_STATE_UNKNOWN, LightControllerSpy_GetLastState());
}

TEST(LightScheduler, light_changed_at_the_right_time)
{
    LightScheduler_ScheduleTurnOn(3, EVERYDAY, 1200);
    FakeTimeService_SetMinute(1200);
    LightScheduler_Wakeup();
    LONGS_EQUAL(3, LightControllerSpy_GetLastId());
    LONGS_EQUAL(LIGHT_ON, LightControllerSpy_GetLastState());
}
Light Scheduler Test List

Light Scheduler Tests

- Lights are not changed at initialization
- Time is wrong, day is wrong, no lights are changed
- Day is right, time is wrong, no lights are changed
- Day is wrong, time is right, no lights are changed
- Day is right, time is right, the right light is turned on
- Day is right, time is right, the right light is turned off
- Schedule every day
- Schedule a specific day
- Schedule all weekdays
- Schedule weekend days
- Remove scheduled event
- Remove non-existent event
- Multiple scheduled events at the same time
- Multiple scheduled events for the same light
- Remove non scheduled light schedule
- Schedule the maximum supported number of events (128)
- Schedule too many events

A Complex system that works is invariably found to have evolved from a simple system that worked.
Partial Skeleton Let’s Your Try the Design and Test Ideas Early

Partial Skeleton Let’s Your Try the Design and Test Ideas Early

Copy/Paste THINK

TEST(LightScheduler, light_not_changed_at_the_wrong_time)
{
    LightScheduler_ScheduleTurnOn(3, EVERYDAY, 1200);
    TransitionClockTo(SUNDAY, 1199);
    ExpectLightsUnchanged();
}

TEST(LightScheduler, light_changed_at_the_right_time)
{
    LightScheduler_ScheduleTurnOn(3, EVERYDAY, 1200);
    TransitionClockTo(SUNDAY, 1200);
    ExpectLightOn(3);
}
Test-Double Substitution Options

Replacing collaborators

- Linker-stubbing
- Function pointer
- Pre-processor
Linker Substitution

Linker Stubbing

#include "sqlite3.h"

int sqlite3_open(const char *filename, sqlite3 **ppDb)
{
    FAIL("sqlite3_open");
    return 0;
}

int sqlite3_step(sqlite3_stmt *stmt)
{
    FAIL("sqlite3_step");
    return 0;
}

int sqlite3_reset(sqlite3_stmt *pStmt)
{
    FAIL("sqlite3_reset");
    return 0;
}

sqlite3_int64 sqlite3_last_insert_rowid(sqlite3*)
{
    FAIL("sqlite3_last_insert_rowid");
    return 0;
}

int sqlite3_bind_int64(sqlite3_stmt*, int, sqlite3_int64)
{
    FAIL("sqlite3_bind_int64");
    return 0;
}

Don’t link the real library. Instead provide a fake that does something useful for the test
Use linker stubs across subsystems

- Use linker stubs on other subsystems
- Linker stubs requires no change in code!

Function Pointer Substitution
Reasons to Use Function Pointers for Test-Doubles

• When you already use function pointers for other purposes (you have a built-in test hook!)
  – Swappable device drivers
  – Swappable implementations
• You need the production implementation in the test build, and you must substitute a test double in the same test build
  – printf
• You don’t want to have too many test builds due to using link-time substitution

Example Usage - Printed Output

• Logs and printed output are helpful for checking program correctness and debugging
• But...
  – They require that you look at the output.
  – They doom you to a lifetime of manually verified tests
• So...
  – Design your code so that printed output can be captured and verified in your unit tests
Manual/Tedious/Error Prone Output Inspection

- Sometimes we want to use the real function.
- Sometimes we want to use a test version.
- Define a function pointer that has the same declaration as the function to override, in this case `printf()`

```c
#ifndef D_FormatOutput_H
#define D_FormatOutput_H
extern int (*FormatOutput)(const char *, ...);
#endif  // D_FormatOutput_H
```
By default, `FormatOutput` points to the `printf`

```c
#include "FormatOutput.h"
#include <stdio.h>

int (*FormatOutput)(const char* format, ...) = printf;
```

Production Code

**Before:**

```c
void someProductionCode()
{
    printf("hello %s\n", "world");
}
```

**After:**

```c
#include "FormatOutput.h"

void someProductionCode()
{
    FormatOutput("hello %s\n", "world");
}
```
Runtime Substitution
Test setup and teardown

• We can override the function by assigning the test version of the function during setup
• Don’t forget to restore the original
  • How should I do this more safely?

```cpp
void setup()
{
    FormatOutput = FormatOutputSpy;
}

void teardown()
{
    FormatOutput = printf;
}
```

• CppUTest has a mechanism for overriding and restoring pointers that must be restored after each test.
• UT_PTR_SET() assigns the pointer and restores the original function pointer after tearDown()

```cpp
void setup()
{
    UT_PTR_SET(FormatOutput, FormatOutputSpy);
}

void teardown()
{
}
```
Sometime we Write Tests for Test Code
We must be able to trust the spy.
Tests are documentation!

```c
TEST(FormatOutput, ATestThatPrintsThings)
{
    FormatOutputSpy_Create(20);
    FormatOutput("Hello, World\n");
    STRCMP_EQUAL("Hello, World\n",
                 FormatOutputSpy_GetOutput());
}
```

Spy Overflow

```c
TEST(FormatOutput, LimitTheOutputBufferSize)
{
    FormatOutputSpy_Create(4);
    FormatOutput("Hello, World\n");
    STRCMP_EQUAL("Hell", FormatOutputSpy_GetOutput());
}
```
More Checking on our Spy

TEST(FormatOutput, PrintMultipleTimes)
{
    FormatOutputSpy_Create(25);
    FormatOutput("Hello");
    FormatOutput("Hello, World\n");
    STRCMP_EQUAL("Hello, World\n", FormatOutputSpy_GetOutput());
}

Spying on the Output

TEST(CircularBufferPrint, PrintNotYetWrappedOrFull)
{
    CircularBuffer_Put(buffer, 10);
    CircularBuffer_Put(buffer, 20);
    CircularBuffer_Put(buffer, 30);
    CircularBuffer_Print(buffer);

    expectedOutput = "Circular buffer content: <10, 20, 30>\n";
    STRCMP_EQUAL(expectedOutput, FormatOutputSpy_GetOutput());
}
Test Fixture

```c
TEST_GROUP(CircularBufferPrint) {
    CircularBuffer buffer;
    const char * expectedOutput;
    const char * actualOutput;

    void setup()
    {
        UT_PTR_SET(FormatOutput, FormatOutputSpy);
        FormatOutputSpy_Create(100);
        buffer = CircularBuffer_Create(10);
    }

    void teardown()
    {
        CircularBuffer_Destroy(buffer);
        FormatOutputSpy_Destroy();
    }
}
```

Preprocessor Substitution
Preprocessor Substitution

• Is the least desirable form of substitution
• Though necessary when
  – Header files won’t compile off-target
  – Header files have are the start to a massive dependency chain
  – A direct function call API cannot be changed and you need to override the direct call and use the direct call in the implementation of the fake
e.g. `malloc()`, `free()`

Preprocessor Substitution Variants

• Header file test double
  – Change the include path during test builds
• Force includes
  – Force in a header file that substitutes problem dependencies
• Command line definition
  – Override a symbol one at a time
Processor Dependent Header File

```c
#if defined(_ACME_X42)
  typedef unsigned int        Uint_32;
  typedef unsigned short      Uint_16;
  typedef unsigned char       Uint_8;

  typedef int                 Int_32;
  typedef short               Int_16;
  typedef char                Int_8;
#elif defined(_ACME_A12)
  typedef unsigned long       Uint_32;
  typedef unsigned int        Uint_16;
  typedef unsigned char       Uint_8;

  typedef long                Int_32;
  typedef int                 Int_16;
  typedef char                Int_8;
#else
  #error <acmetypes.h> is not supported for this environment
#endif
```

Adjust the Include Path So the `#include` Test-Double is Found First

```c
#ifndef ACMETYPES_H_
#define ACMETYPES_H_

#include <stdint.h>

typedef uint32_t Uint_32;
typedef uint16_t Uint_16;
typedef uint8_t  Uint_8;

typedef int32_t  Int_32;
typedef int16_t  Int_16;
typedef int8_t   Int_8;

#endif /* ACMETYPES_H_ */
```

Read about it at http://www.renaissancesoftware.net/blog/archives/231
Just to be Sure, Add this Test

```c
TEST(acmetypes, checkIntSizes)
{
    LONGS_EQUAL(1, sizeof(Uint_8));
    LONGS_EQUAL(1, sizeof(Int_8));
    LONGS_EQUAL(2, sizeof(Uint_16));
    LONGS_EQUAL(2, sizeof(Int_16));
    LONGS_EQUAL(4, sizeof(Uint_32));
    LONGS_EQUAL(4, sizeof(Int_32));
}
```

Ideally

- Use a portable types file, rather than vendor dependent file
- Limit the areas of your code that depend on problem vendor code
See the C that is NOT C

/* Chip Vendor Specific Header File */
...
extern cregister volatile unsigned int AMR;  /* Address Mode Register */
extern cregister volatile unsigned int CSR;  /* Control Status Register */
extern cregister volatile unsigned int IFR;  /* Interrupt Flag Register */
extern cregister volatile unsigned int ISR;  /* Interrupt Set Register */
extern cregister volatile unsigned int ICR;  /* Interrupt Clear Register */
extern cregister volatile unsigned int IER;  /* Interrupt Enable Register */
extern cregister volatile unsigned int ISTP; /* Interrupt Service Tbl Ptr */
extern cregister volatile unsigned int IRP;  /* Interrupt Return Pointer */
extern cregister volatile unsigned int NRP;  /* Non-maskable Int Return Ptr */
extern cregister volatile unsigned int IN;   /* General Purpose Input Reg */
extern cregister volatile unsigned int OUT;  /* General Purpose Output Reg */
...

Read about it at http://www.renaissancesoftware.net/blog/archives/249

Force Include

//OffTargetDefines.h
#define cregister
#define interrupt
...


Registers Become Global Variables

```c
// FakeRegisters.c
volatile unsigned int AMR;
volatile unsigned int CSR;
volatile unsigned int IFR;
volatile unsigned int ISR;
volatile unsigned int ICR;
volatile unsigned int IER;
volatile unsigned int ISTP;
volatile unsigned int IRP;
volatile unsigned int NRP;
volatile unsigned int IN;
volatile unsigned int OUT;
```

Ideally

- You should limit areas of your code that are vendor specific
- Make a Hardware Abstraction Layer
Command Line Definition

- Compilers all preprocessor symbols to be defined on the command line
- These gcc command line options changes all occurrences of `malloc()` to `cpputest_malloc()` and `free()` to `cpputest_free()`

```
-Dmalloc=cpputest_malloc
-Dfree=cpputest_free
```

Undefine the Override So the Original Can be Used

```
#undef malloc
#undef free

void * cpputest_malloc(size_t size)
{
  // do the malloc/free book keeping
  return malloc(size);
}

void cpputest_free(void * mem)
{
  // do the malloc/free book keeping
  return free(mem);
}
```
Problem - **asm**

The legacy code has **asm** instructions that won’t compile off-target

**Solution - 1**

Make **asm** go away using forced include

**Solution - 2**

Introduce an **AsmSpy** to capture the instruction stream and check it in a test case

- See http://www.renaissancesoftware.net/blog/archives/136

---

**What is an **AsmSpy**?**

```c
TEST(AsmSpy, captures_asm_text)
{
    AsmSpy("NOP");
    AsmSpy("GLOP");
    AsmSpy("SLOP");
    STRCMP_EQUAL("NOP;GLOP;SLOP;", AsmSpy_Debrief());
}
```

Read more:
http://www.renaissancesoftware.net/blog/archives/136
http://www.renaissancesoftware.net/blog/archives/143
Force Include AsmSpy.h

```c
#ifndef D_AsmSpy_H
#define D_AsmSpy_H

#define asm AsmSpy

void AsmSpy_Create(int size);
void AsmSpy_Destroy(void);
void AsmSpy(const char *);
const char * AsmSpy_Debrief(void);

#endif
```

Problem - `#pragma`

The legacy code has `#pragma` instructions that won’t compile off-target

Solution

Adjust the compiler settings to ignore unknown `#pragmas`
Types of Test-doubles

Test-double types

- Exploding stubs
- Dynamic
- Recording - Mock
- Generic
Exploding stubs

```c
sqlite3_int64 sqlite3_last_insert_rowid(sqlite3*)
{
    FAIL("sqlite3_last_insert_rowid");
    return 0;
}
```

Easy and you know when it is used.
Delays more implementation...

Dynamic stub

```c
stubSqlite.cpp

sqlite3_int64 sqlite3_last_insert_rowid(sqlite3* lite)
{
    if (sqlite3_last_insert_rowid_stub)
        return sqlite3_last_insert_rowid_stub(lite);
    return 0;
}
```

```
stubSqlite.h
extern sqlite3_int64 (*sqlite3_last_insert_rowid_stub)(sqlite3*);
```

“Sensible default”
**Recording ... mock**

---

**stubSqlite.cpp**

```cpp
generic stub
sqlite3_int64 sqlite3_last_insert_rowid(sqlite3* lite)
{
    if (sqlite3_last_insert_rowid_stub)
        return sqlite3_last_insert_rowid_stub(lite);
    mock("Sqlite").actualCall("sqlite3_last_insert_rowid").withParameter("lite", lite);
    if (mock("Sqlite").hasReturnValue())
        return (sqlite3_int64) mock("Sqlite").returnValue().getIntValue();
    return 0;
}
```

---

**testUsingSQL.cpp**

```cpp
Dynamic when set

```

---

**stubSqlite.cpp**

```
Generic stub

```

---

```
Mock by default
With a “sensible default”
```
CppUMock MockPlugin

MockPlugin does checking expectations and cleanup automatically

```cpp
int main(int ac, char** av)
{
    MockSupportPlugin plugin;
    TestRegistry::getCurrentRegistry() -> installPlugin(&plugin);
    return CommandLineTestRunner::RunAllTests(ac, av);
}
```

Install it!

Mocking the Silicon
Why a Mock Object?

- Problem - Complex collaborator interactions cannot be captured with a simple spy.
- Solution - Mock Object
  - A Mock Object is a Test Double that verifies that the code being tested interacts with its collaborator properly.
  - The test tells the mock
    - The expected calls
    - In the expected order
    - What to return.

Message Flow for Flash Memory Block Erase with Error
Flash Program Flow Chart

How Many Tests are Needed?

Start
Program Command Write 0x40 to 0x0
Write data to address
Read status register
b7 == 1
Wait for ready loop
NO
YES

b1 == 0
NO
YES
Protected Block Error
Clear status Write 0xFF to 0x0
b3 == 0
YES
NO
Vpp Error
b4 == 0
YES
NO
Program Error

Flash Driver Test Fixture

FlashDriverTest

FlashDriver
+ Program(addr, data)
+ ProtectBlock(block)
+ EraseBlock(block)
 //etc

<<interface>>
IO
+ Read(addr) : data
+ Write(addr, data)

<<implements>>
MockIO
+ Expect_Read(addr, data)
+ Expect_Write(addr, data)

<<implementation>>
IO
<<hardware>>
Flash Driver Test

TEST(Flash, ProgramSucceedsReadyImmediately)
{
    int result = 0;
    mock("IO").strictOrder();
    mock("IO").expectOneCall("IOWrite").withParameter("addr", CommandRegister).withParameter("value", 0x40);
    mock("IO").expectOneCall("IOWrite").withParameter("addr", (int)0x1000).withParameter("value", 0xBEEF);
    mock("IO").expectOneCall("IORead").withParameter("addr", (int)0).andReturnValue(1<<7);
    mock("IO").expectOneCall("IORead").withParameter("addr", (int)0x1000).andReturnValue(0xBEEF);
    result = Flash_Program(0x1000, 0xBEEF);
    LONGS_EQUAL(0, result);
    mock().checkExpectations();
    mock().clear();
}

Mock Flash Write / Read

void IOWrite(ioAddress_t addr, ioData_t value)
{
    mock_scope_c("IO")->actualCall("IOWrite")->withIntParameters("addr", addr)->withIntParameters("value", value);
}

ioData_t IORead(ioAddress_t addr)
{
    mock_scope_c("IO")->actualCall("IORead")->withIntParameters("addr", addr);
    return mock_scope_c("IO")->returnValue().value.intValue;
}
What do the Tests Mean?

- Vendor’s driver did not pass my tests
- Undocumented operations (resets) were added in silicon vendor’s solution.
- My driver functions met the spec, but may have encountered integration problems
  – Were the extra resets really needed?
TDD Next to an RTOS

Intro to the Fake Function Framework

Choices When Faking the OS

• For new code, you might choose to create an OS abstraction layer.
• For existing code, or where the layer is not desired, you can create an RTOS test double
How Do I Test OS Dependent Code Like This?

```c
int MessageProcessor_WaitForMessage(char * buffer, size_t length) {
    INT8U error = 0;
    SerialInterrupt_WaitForString(buffer, length, int_sync);
    OSSemPend(int_sync, 1000, &error);
    return error == OS_ERR_NONE;
}
```

I may just fake out `WaitForMessage()` and manually test the OS dependent code.

But that may not be an option in legacy code where OS primitives are used more liberally.

The ISR Has to Satisfy the Request
Let the Fake Do the Work That Would Happen Concurrently

Fakes are Created With the Fake Function Framework

```c
#ifndef D_uCosII_TestDouble_H
#define D_uCosII_TestDouble_H

#include "fff2.h"
extern "C" {
#include "ucos_ii.h"
}

FAKE_VALUE_FUNCTION(OS_EVENT *, OSSemCreate, INT16U)

FAKE_VOID_FUNCTION(OSSemPend, OS_EVENT *,		INT32U,\t	INT8U *)
```
Basic fff Features

TEST(uCosII_TestDouble, OSSemPend_basics)
{
    OS_EVENT event;
    INT8U error;
    OSSemPend(&event, 0, &error);
    LONGS_EQUAL(1, OSSemPend_fake.call_count);
    POINTERS_EQUAL(&event, OSSemPend_fake.arg0_val);
    LONGS_EQUAL(0, OSSemPend_fake.arg1_val);
    POINTERS_EQUAL(&error, OSSemPend_fake.arg2_val);
    POINTERS_EQUAL(&event, OSSemPend_fake.arg0_history[0]);
    LONGS_EQUAL(0, OSSemPend_fake.arg1_history[0]);
}

The Test Case

TEST(MessageProcessor, WaitForMessage_succeeds)
{
    fakeInput = "sched lightOn 5 Monday 20:00";
    CHECK_TRUE(        
        MessageProcessor_WaitForMessage(      
            (char*)&receiveBuffer, sizeof(buffer))      
    );
    LONGS_EQUAL(1, OSSemPend_fake.call_count);
    STRCMP_EQUAL(fakeInput, receiveBuffer);
}
Its Setup and Teardown

TEST_GROUP(MessageProcessor)
{
    void setup()
    {
        OSSemPend_reset();
        OSSemPend_fake.custom_fake = MyOSSemPend;
        MessageProcessor_Create();
    }

    void teardown()
    {
        MessageProcessor_Destroy();
    }
}

Its Custom Implementation Satisfies the Scenario

static const char * fakeInput;
static char receiveBuffer[100];
void MyOSSemPend(OS_EVENT *event, INT32U timeout, INT8U *error)
{
    memcpy(receiveBuffer, fakeInput, strlen(fakeInput));
    *error = OS_ERR_NONE;
}
Scaling Lean & Agile Development
Thinking and Organizational Tools for Large Scale Scrum
Craig Larman
Bas Vodde

Practices for Scaling Lean & Agile Development
Large, Multisite, and Offshore Products with Large-Scale Scrum
Craig Larman
Bas Vodde