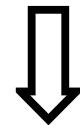


Feature Driven Design (Using TDD and Mocks)

ESC-427 Mark VanderVoord & Greg Williams

An Alternative To

Top



Down

Up



Bottom

Instead of Slicing

Application

SubSystems

Hardware Drivers

We Instead Slice

Application

SubSystems

Hardware Drivers

So

We Need

An Example

How About



Measure Battery Voltage

With Filtered Input



Output
Volts
Once
Per
Second

And Also, It Should

Send Wakeup Message Over Serial Port



So Where To Start?

Choose a Feature

Send Wakeup Message Over Serial Port



[Why?]

Let's Start with main

main.c

main.c

```
void main(void)
{
}
```

main.c

```
#include "Executor.h"

void main(void)
{
    while(Executor_Run());
}
```

main.c

```
#include "Executor.h"

void main(void)
{
    Executor_Init();

    while(Executor_Run());
}
```

Executor.c

Executor.c

```
#include "Executor.h"

bool Executor_Run(void)
{
    return TRUE;
}
```

Executor.c

```
#include "Executor.h"

void Executor_Init(void)
{
}

bool Executor_Run(void)
{
    return TRUE;
}
```

Executor.c

```
#include "Executor.h"
#include "UsartConductor.h"

void Executor_Init(void)
{
}

bool Executor_Run(void)
{
    UsartConductor_Run();
    return TRUE;
}
```

Executor.c

```
#include "Executor.h"
#include "UsartConductor.h"

void Executor_Init(void)
{
    UsartConductor_Init();
}

bool Executor_Run(void)
{
    UsartConductor_Run();
    return TRUE;
}
```

Wait!

Executor?

Conductor?

A Quick Aside

Design Pattern

MCH

Model Conductor Hardware

[Model]

Module's API

Only Public Interface

[Hardware]

Low-Level

Device-Driver

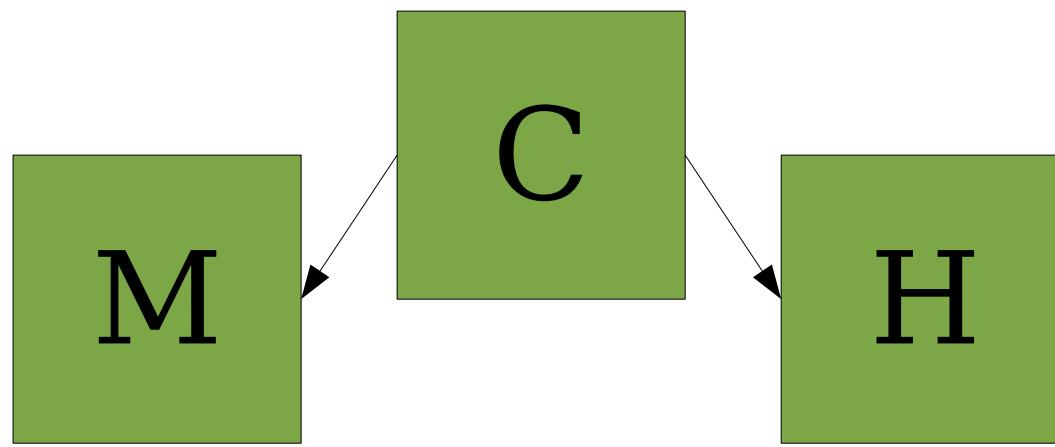
Only Interface to HW

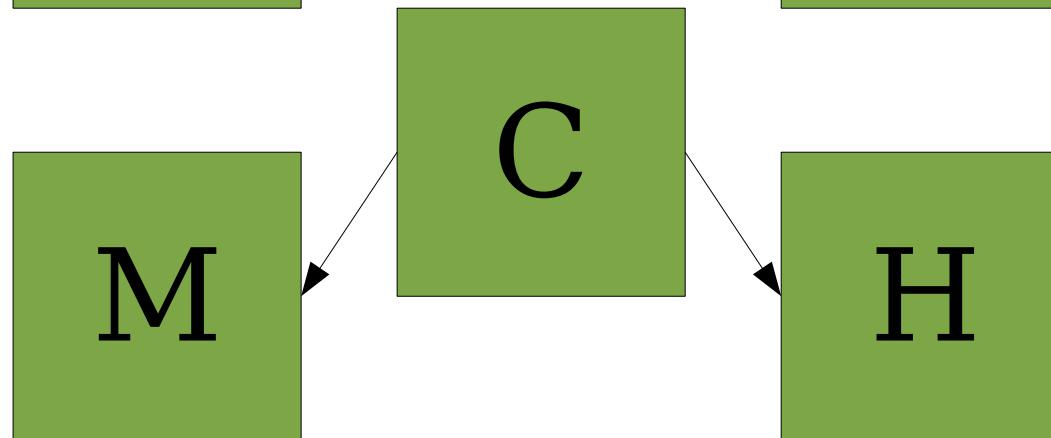
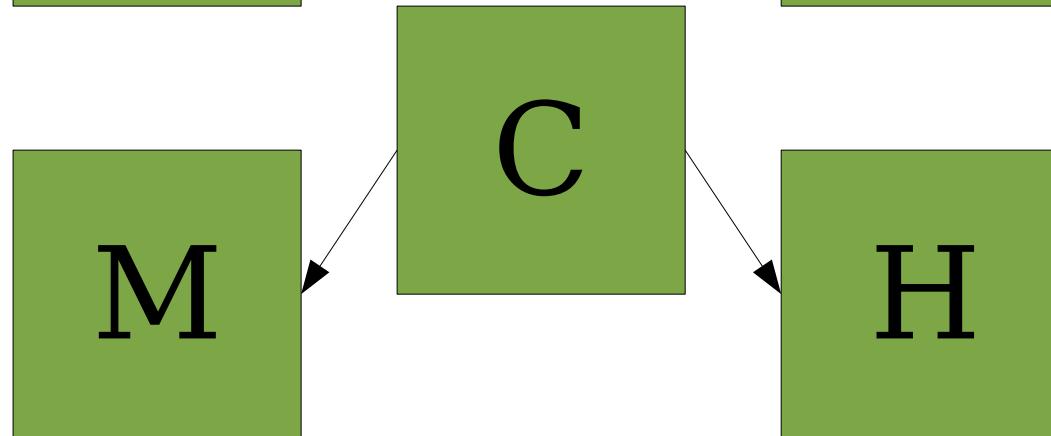
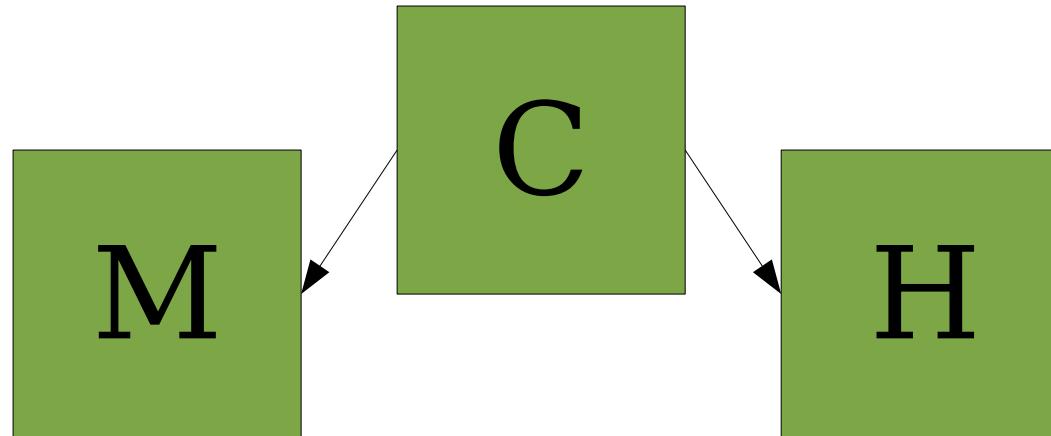
[Conductor]

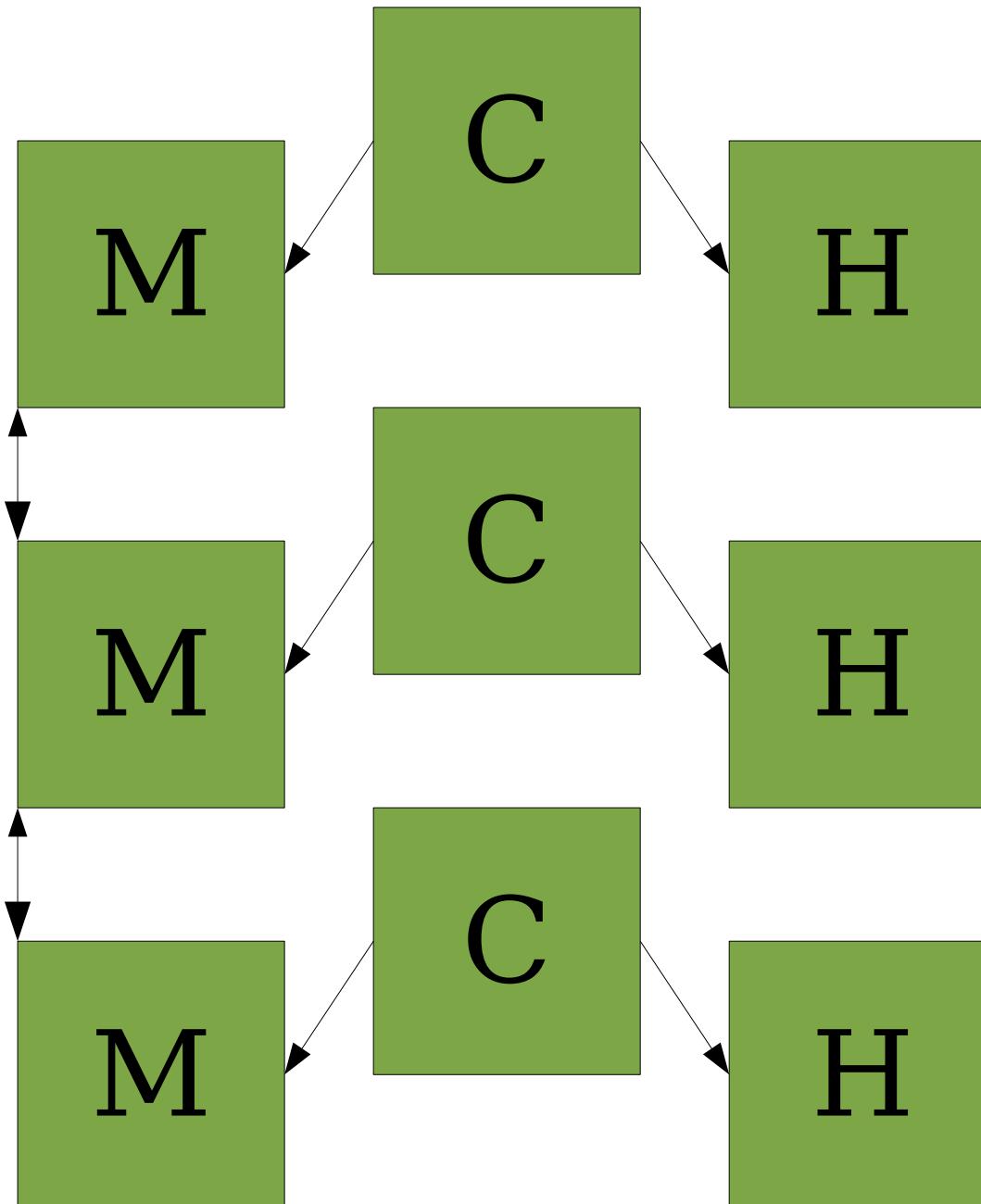
Sole Connection

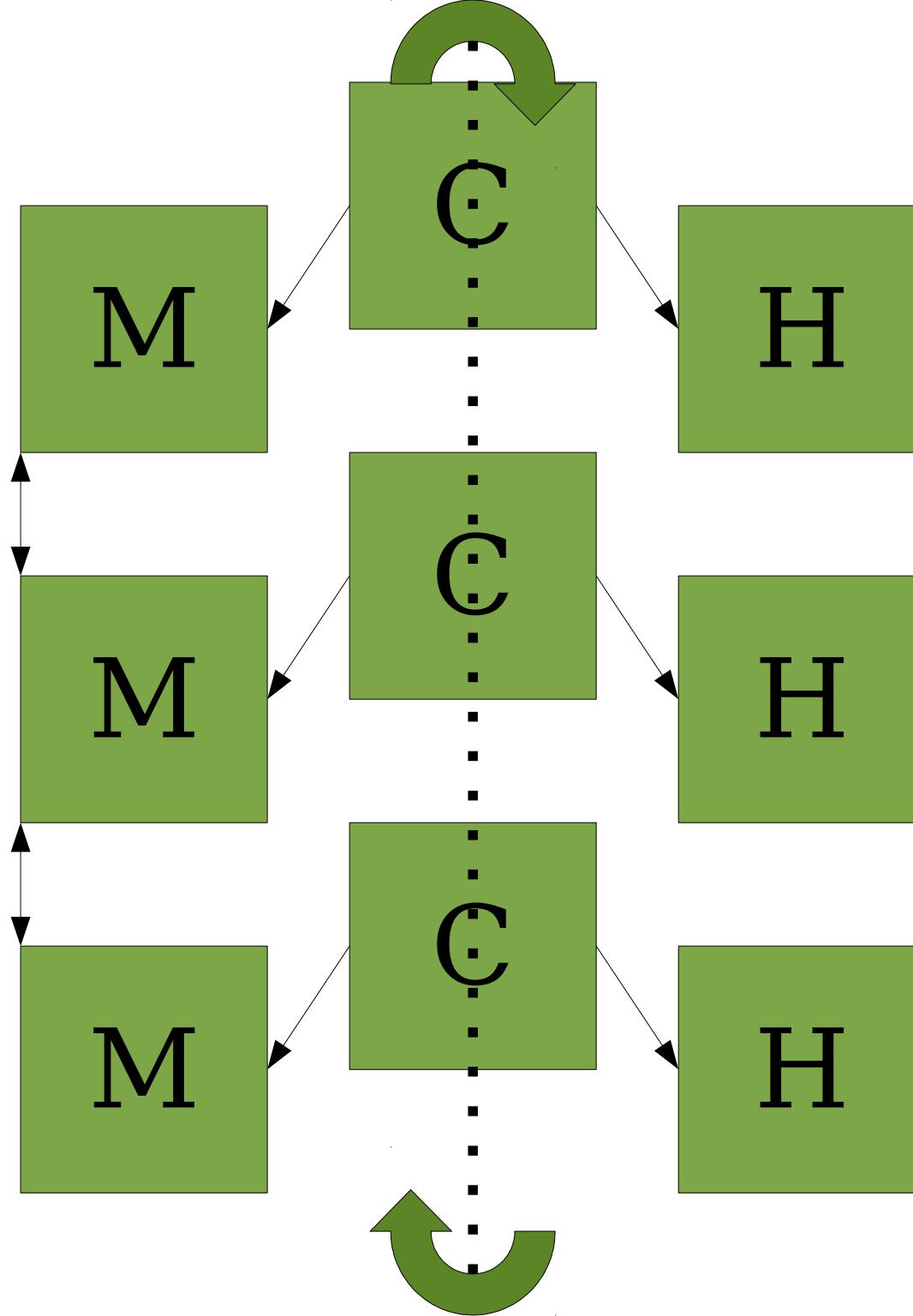
Calls Model

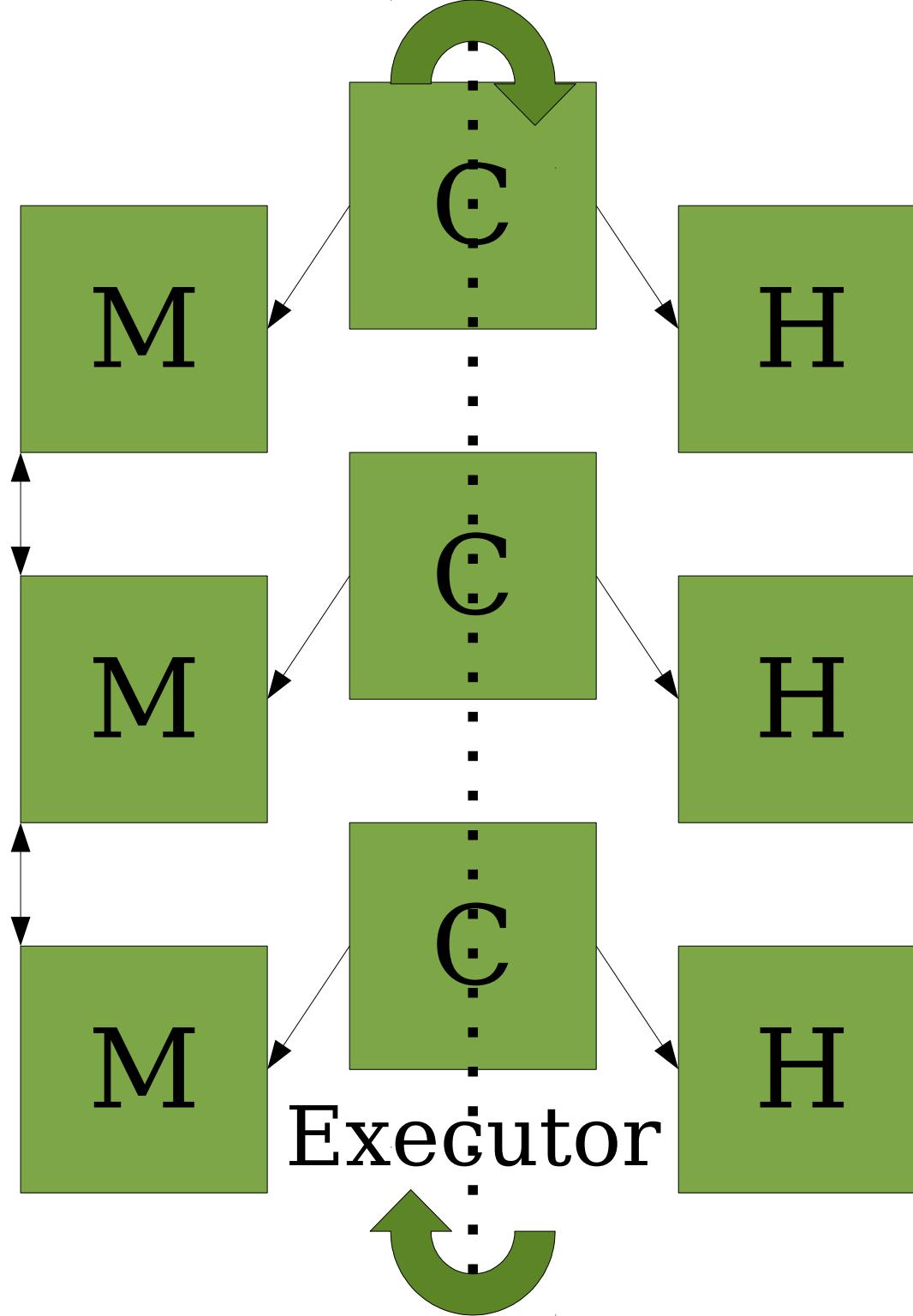
Calls Hardware

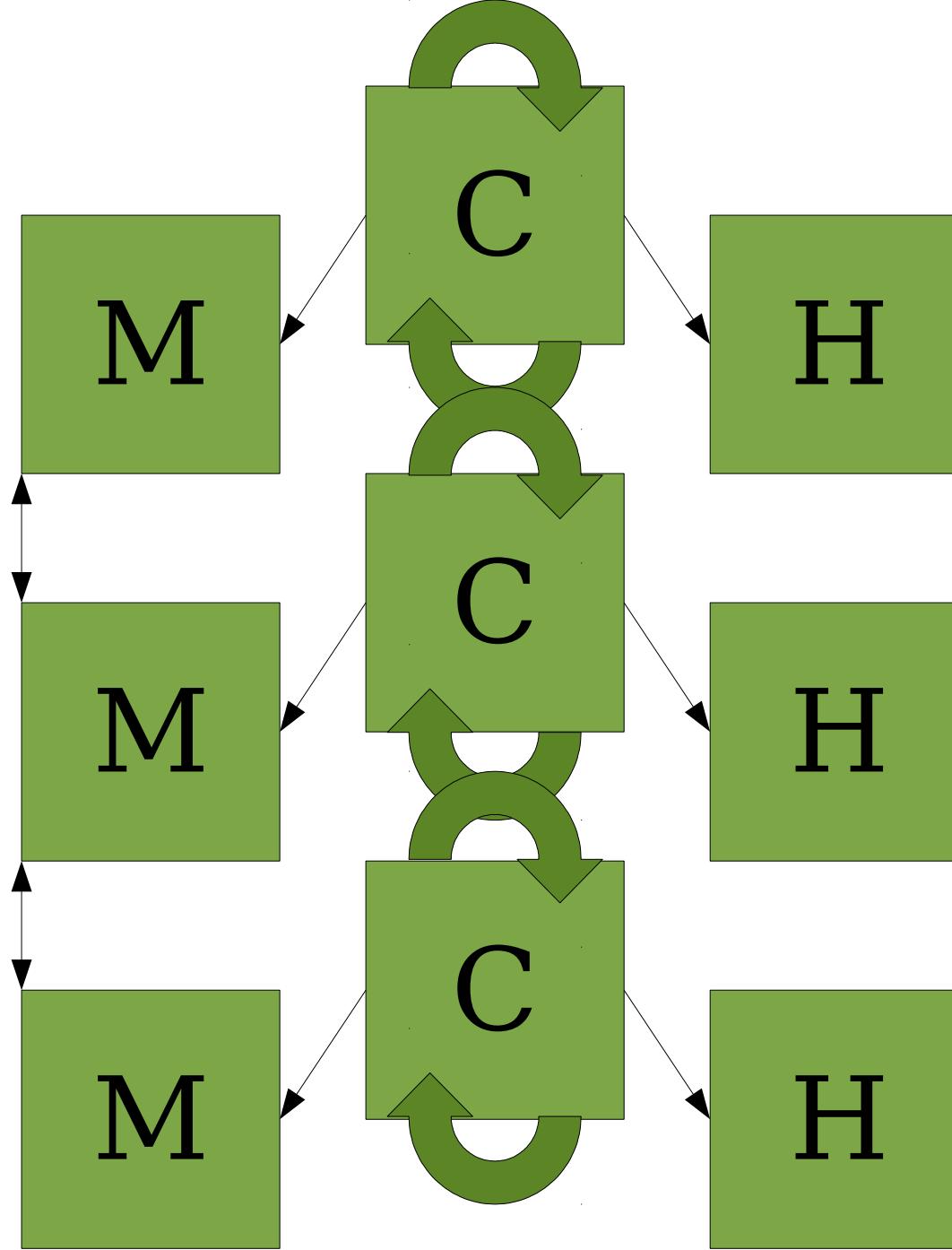












Back To
UsartConductor

UsartConductor.c

UsartConductor.c

```
#include "UsartConductor.h"
#include "UsartHardware.h"
#include "UsartModel.h"

void UsartConductor_Init(void)
{
}

}
```

UsartConductor.c

```
#include "UsartConductor.h"
#include "UsartHardware.h"
#include "UsartModel.h"

void UsartConductor_Init(void)
{
    UsartHardware_Init();

}

}
```

UsartConductor.c

```
#include "UsartConductor.h"
#include "UsartHardware.h"
#include "UsartModel.h"

void UsartConductor_Init(void)
{
    UsartHardware_Init();
    UsartHardware_TransmitString()

}
```

UsartConductor.c

```
#include "UsartConductor.h"
#include "UsartHardware.h"
#include "UsartModel.h"

void UsartConductor_Init(void)
{
    UsartHardware_Init();
    UsartHardware_TransmitString(
        UsartModel_GetWakeupString()
    );
}
```

UsartModel.c

UsartModel.c

```
#include "UsartModel.h"

char* UsartModel_GetWakeupString(void)
{
    return wakeup;
}
```

UsartModel.c

```
#include "UsartModel.h"

char* wakeup = "It's Awesome Time!";

char* UsartModel_GetWakeupString(void)
{
    return wakeup;
}
```

UsartHardware.c

UsartHardware.c

```
#include "UsartHardware.h"

void UsartHardware_Init()
{
}
```

UsartHardware.c

```
#include "UsartHardware.h"

void UsartHardware_Init()
{
    Usart_ConfigureUsartI0();

}
```

UsartHardware.c

```
#include "UsartHardware.h"

void UsartHardware_Init()
{
    Usart_ConfigureUsartIO();
    Usart_EnablePeripheralClock();

}
```

UsartHardware.c

```
#include "UsartHardware.h"

void UsartHardware_Init()
{
    Usart_ConfigureUsartIO();
    Usart_EnablePeripheralClock();
    Usart_Reset();

}
```

UsartHardware.c

```
#include "UsartHardware.h"

void UsartHardware_Init()
{
    Usart_ConfigureUsartIO();
    Usart_EnablePeripheralClock();
    Usart_Reset();
    Usart_ConfigureMode();

}
```

UsartHardware.c

```
#include "UsartHardware.h"

void UsartHardware_Init()
{
    Usart_ConfigureUsartIO();
    Usart_EnablePeripheralClock();
    Usart_Reset();
    Usart_ConfigureMode();
    Usart_SetBaudRateRegister();

}
```

UsartHardware.c

```
#include "UsartHardware.h"

void UsartHardware_Init()
{
    Usart_ConfigureUsartIO();
    Usart_EnablePeripheralClock();
    Usart_Reset();
    Usart_ConfigureMode();
    Usart_SetBaudRateRegister();
    Usart_Enable();
}
```

UsartHardware.c

```
#include "UsartHardware.h"
#include "UsartConfigurator.h"

void UsartHardware_Init()
{
    Usart_ConfigureUsartIO();
    Usart_EnablePeripheralClock();
    Usart_Reset();
    Usart_ConfigureMode();
    Usart_SetBaudRateRegister();
    Usart_Enable();
}
```

UsartHardware.c

```
#include "UsartHardware.h"
#include “UsartConfigurator.h”

void UsartHardware_Init() ...

void UsartHardware_TransmitString(char*
data)
{
}

}
```

UsartHardware.c

```
#include "UsartHardware.h"
#include “UsartConfigurator.h”

void UsartHardware_Init() ...

void UsartHardware_TransmitString(char*
data)
{
    while(*data != NULL)
    {
        }

    }
}
```

UsartHardware.c

```
#include "UsartHardware.h"
#include "UsartConfigurator.h"

void UsartHardware_Init() ...

void UsartHardware_TransmitString(char*
data)
{
    while(*data != NULL)
    {
        Usart_PutChar(*data++);
    }
}
```

UsartHardware.c

```
#include "UsartHardware.h"
#include “UsartConfigurator.h”
#include "UsartPutChar.h"

void UsartHardware_Init() ...

void UsartHardware_TransmitString(char*
data)
{
    while(*data != NULL)
    {
        Usart_PutChar(*data++);
    }
}
```

Skeleton Forming

Functions
Added As Needed

Our Next Functions

Talk To HW Registers

Let's Look At One

UsartConfigurator.c

```
#include "UsartConfigurator.h"

void Usart_ConfigureUsartIO(void)
{
}

}
```

UsartConfigurator.c

```
#include "UsartConfigurator.h"
#include "AT91SAM7X256.h"

void Usart_ConfigureUsartIO(void)
{
}

}
```

UsartConfigurator.c

```
#include "UsartConfigurator.h"
#include "AT91SAM7X256.h"

void Usart_ConfigureUsartIO(void)
{
    AT91C_BASE_PIOA->PIO_ASR =
        USART_TX_PIN;
    AT91C_BASE_PIOA->PIO_BSR = 0;
    AT91C_BASE_PIOA->PIO_PDR =
        USART_TX_PIN;
}
```

Wait!

Where are the Tests?

Unit Tests

TestUsartConfigurator.c

```
#include "UsartConfigurator.h"
#include "AT91SAM7X256.h"
#include "unity.h"
```

TestUsartConfigurator.c

```
#include "UsartConfigurator.h"  
#include "AT91SAM7X256.h"  
#include "unity.h"
```

```
void testShouldConfigPioPinForUsartTx(void)
```

```
{
```

```
}
```

TestUsartConfigurator.c

```
#include "UsartConfigurator.h"
#include "AT91SAM7X256.h"
#include "unity.h"

void testShouldConfigPioPinForUsartTx(void)
{
    AT91C_BASE_PIOA->PIO_ASR = 0;
    AT91C_BASE_PIOA->PIO_BSR = 0xffffffff;
    AT91C_BASE_PIOA->PIO_PDR = 0;

}
```

TestUsartConfigurator.c

```
#include "UsartConfigurator.h"
#include "AT91SAM7X256.h"
#include "unity.h"

void testShouldConfigPioPinForUsartTx(void)
{
    AT91C_BASE_PIOA->PIO_ASR = 0;
    AT91C_BASE_PIOA->PIO_BSR = 0xffffffff;
    AT91C_BASE_PIOA->PIO_PDR = 0;
    Usart_ConfigureUsartI0();

}

}
```

TestUsartConfigurator.c

```
#include "UsartConfigurator.h"
#include "AT91SAM7X256.h"
#include "unity.h"

void testShouldConfigPioPinForUsartTx(void)
{
    AT91C_BASE_PIOA->PIO_ASR = 0;
    AT91C_BASE_PIOA->PIO_BSR = 0xffffffff;
    AT91C_BASE_PIOA->PIO_PDR = 0;
    Usart_ConfigureUsartIO();
    TEST_ASSERT_EQUAL(USART_TX_PIN,
                      AT91C_BASE_PIOA->PIO_ASR);
    TEST_ASSERT_EQUAL(0,
                      AT91C_BASE_PIOA->PIO_BSR);
    TEST_ASSERT_EQUAL(USART_TX_PIN,
                      AT91C_BASE_PIOA->PIO_PDR);
}
```

Wait!

TEST_ASSERT_EQUAL
?



Unit Test Framework

Sets Up

Runs

Collects Results

Unit Tests

Collection

Assertions

Describing

WHAT

Should Happen

[Assertions Like What?]

So, Does It Work?



Wait!

How Did It Run?

Two Tools

Make the Magic

1

Simulator

Address-Specific Code

Same Compiler

2

Scripting Language



1950s
1960s
1970s
1980s

1990s
2000s

2010s
2020s



Discovers Tests

Builds Runners

Executes Tests

Returns Results

Making Testing

Painless



So We're Ducky

Wait!

Why Not Test-Driven?

We Should Have

Feature 2



Measure Battery Voltage



Output
Volts
Once
Per
Second

Remember main() ?

main.c

```
#include "Executor.h"

void main(void)
{
    Executor_Init();

    while(Executor_Run());
}
```

so main's test

Testmain.c

```
void testMainShouldCallExecutorInit  
AndCallExecutorRunUntilFalse(void)  
{  
    Executor_Init_Expect();  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(FALSE);  
  
    main();  
}
```

let's start with
Executor_Run

TestExecutor.c

```
void testRunShouldCallRunForEachConductor()
{
}

}
```

TestExecutor.c

```
void testRunShouldCallRunForEachConductor()
{
    TEST_ASSERT_EQUAL(TRUE, Executor_Run());
}
```

TestExecutor.c

```
void testRunShouldCallRunForEachConductor()
{
    UsartConductor_Run_Expect();
    TimerConductor_Run_Expect();
    AdcConductor_Run_Expect();

    TEST_ASSERT_EQUAL(TRUE, Executor_Run());
}
```

wait!

where do these

_Expect

functions come from?



Mock

TestExecutor.c

```
#include "Unity.h"
#include "CMock.h"
#include "Executor.h"
#include "MockModel.h"
#include "MockUsartConductor.h"
#include "MockAdcConductor.h"
#include "MockTimerConductor.h"
```

run the test

it can't find the mocks!

weren't they magic?

a little about CMock

Searches Source Files

Creates Mocks

All Public Functions

Simple Rules

Functions We Expect To Be Called

void BeAwesome(void)



void BeAwesome_Expect(void)

Functions We Expect To Be Called With Specific Arguments

```
void BeSwank(int Percent,  
             char* Name)
```



```
void BeSwank_Expect(int Percent,  
                     char* Name)
```

Functions We Expect To Be Called And Return A Specific Value

```
int BeSpiffy(void)
```



```
void BeSpiffy_ExpectAndReturn(int  
    toReturn)
```

Functions We Expect To Be Called With Arguments and Return a Val

```
int BeMe(char* Name)
```



```
void BeMe_ExpectAndReturn(  
    char* Name, int toReturn)
```

Re-Look at TestMain

Testmain.c

```
void testMainShouldCallExecutorInit  
AndCallExecutorRunUntilFalse(void)  
{  
    Executor_Init_Expect();  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(TRUE);  
    Executor_Run_ExpectandReturn(FALSE);  
  
    main();  
}
```

And TestExecutor

TestExecutor.c

```
void testRunShouldCallRunForEachConductor()
{
    UsartConductor_Run_Expect();
    TimerConductor_Run_Expect();
    AdcConductor_Run_Expect();

    TEST_ASSERT_EQUAL(TRUE, Executor_Run());
}
```

[so why no mocks?]

we need those
conductors

don't implement now

TestUsartConductor.c

```
void testNeedToImplementUsartConductor_Run()
{
    TEST_IGNORE();
}
```

UsartConductor.c

```
void UsartConductor_Run(void)
{
}
```

TEST_IGNORE ?

shows up

test summary

like a trail of breadcrumbs



12/27/2006

“sweet”

rerun tests



what happened?

exactly what should

`Executor_Run` just
returns `TRUE`

Executor.c

```
bool Executor_Run(void)
{
    return TRUE;
}
```

Executor.c

```
bool Executor_Run(void)
{
    UsartConductor_Run();
    TimerConductor_Run();
    AdcConductor_Run();
    return TRUE;
}
```

rerun tests



notice output

lists ignores

lets choose ADC

TestAdcConductor.c

```
void  
testRunShouldNotDoAnythingIfItIsNotTime( )  
{  
    AdcConductor_Run();  
}
```

TestAdcConductor.c

```
void
testRunShouldNotDoAnythingIfItIsNotTime( )
{
    AdcModel_DoGetSample_ExpectAndReturn(
        FALSE);

    AdcConductor_Run();
}
```

again, we're calling

non-existent functions

TestAdcModel.c

```
void  
testNeedToImplementAdcModel_DoGetSample()  
{  
    TEST_IGNORE();  
}
```

AdcModel.c

```
bool AdcModel_DoGetSample(void)  
{  
    return FALSE;  
}
```

does it work?



implement it!

AdcConductor.c

```
void AdcConductor_Run(void)
{
    if (AdcModel_DoGetSample())
    {
    }
}
```

does it work?



AdcModel has IGNORE

so continue Conductor

TestAdcConductor.c

```
void
testShouldNotGetAdcResultIfSampleNotComplete(
)
{
    AdcModel_DoGetSample_Return(TRUE);

    AdcHardware_GetSampleComplete_Return(FALSE);

    AdcConductor_Run();
}
```

which requires

TestAdcHardware.c

```
void testNeedAdcHardware_GetSampleComplete()
{
    TEST_IGNORE();
}
```

AdcHardware.c

```
bool AdcHardware_GetSampleComplete(void)
{
    return FALSE;
}
```

and of course fails

until we write

AdcConductor.c

```
void AdcConductor_Run(void)
{
    if (AdcModel_DoGetSample() &&
        AdcHardware_GetSampleComplete())
    {
    }
}
```

does it work?



so next test in
conductor

TestAdcConductor.c

```
void
testRunShouldGetLatestSampleFromAdcAndPassItToModelAndStartNewConversionWhenItIsTime()
{
    AdcModel_DoGetSample_ExpectAndReturn(TRUE);
    AdcHardware_GetSampleComplete_Return(TRUE);
    AdcHardware_GetSample_ExpectAndReturn(293);
    AdcModel_ProcessInput_Expect(293);
    AdcHardware_StartConversion_Expect();

    AdcConductor_Run();
}
```

We Don't Want To
Forget To Start First
Conversion

So Use Ignores Again

TestAdcConductor.c

```
void
testAdcConductorInitShouldStartConversion()
{
    TEST_IGNORE();
}
```

also,

placeholders for

AdcHardware_GetSample()
AdcModel_ProcessInput()
AdcHardware_StartConversion()

to make it pass

AdcConductor.c

```
void AdcConductor_Run(void)
{
    if (AdcModel_DoGetSample() &&
        AdcHardware_GetSampleComplete())
    {
        AdcModel_ProcessInput(
            AdcHardware_GetSample());
        AdcHardware_StartConversion();
    }
}
```

Conductor's Done

A large, ripe peach is centered against a dark blue background. It rests on a vibrant red, textured surface that appears to be a cloth or a textured rug. The peach's skin has a smooth, yellowish-orange gradient with some darker orange and red hues near the bottom right. A small, shallow depression is visible at the top center where the stem was attached.

So
That's Peachy

When We Run Tests

Get List Of Ignores

AdcConductor_Init()

AdcHardware_GetSampleComplete()

AdcHardware_GetSample()

AdcHardware_StartConversion()

AdcModel_DoGetSample()

AdcModel_ProcessInput()

Remember These Are

breadcrumbs



12/27/2006

So Pick One,

Start Implementing

Repeat Until No
Ignores

Feature Should Be
Complete

Questions Before We
Start Talking About
Specific Tricks?

Function
Under Test
Returns An
Array



No Problem

If

char*

or
const char*

TEST_ASSERT_EQUAL_STRING

otherwise

can check pointer
(if important to us)

TEST_ASSERT_EQUAL

and / or

roll a helper function

```
void AssertEqualIntArray(int* expected,
                        int* actual,
                        unsigned int length)
{
    unsigned int i;
    for (i = 0; i < length; i++)
    {
        sprintf(message,
                "Array failed at index %u.", i);
        TEST_ASSERT_EQUAL_INT_MESSAGE(
            expected[i],
            actual[i],
            message);
    }
}
```

this (and others)

included in
UnityHelper.c

Can I Verify
I've Tested
All Options
In An Array
Of Function
Pointers



Of Course

Let's Say You Have
Something Like This

```
YUMMY_T Yummies[] = {
    {"sandwich", Sandwich_Make},
    {"pizza", Pizza_Make},
    {"donut", Donut_Make},
    {"cookie", Cookie_Make}
};
```

```
void* Yummy_Make(char* snacktype)
{
    int i;
    for (i=0; i < 4; i++) {
        if (strcmp(Yummies[i].str,snacktype)==0)
            return (Yummies[i].func());
    }
    return NULL;
}
```

we have tests like this

```
void test_VerifyCookiesCanBeMade( )
{
    void* FakeCookie = (void*)0x5a5a;
    void* Retval;

    Cookie_Make_ExpectAndReturn(FakeCookie);

    Retval = Yummy_Make("cookie");
    TEST_ASSERT_EQUAL(FakeCookie, Retval);
}
```

but how do we know

that we tested each condition?

globals aren't a sin
(in the tests)

```
static int FunctionPointerCalls = 0;

void test_VerifyCookiesCanBeMade( )
{
    void* FakeCookie = (void*)0x5a5a;
    void* Retval;

    Cookie_Make_ExpectAndReturn(FakeCookie);

    Retval = Yummy_Make("cookie");
    TEST_ASSERT_EQUAL(FakeCookie, Retval);

    FunctionPointerCalls++;
}
```

```
static int FunctionPointerCalls = 0;

void test_VerifyCookiesCanBeMade() {...}
void test_VerifyPizzasCanBeMade() {...}
void test_VerifyDonutsCanBeMade() {...}
void test_VerifySandwichesCanBeMade() {...}

void test_VerifyAllFunctionPointersTested()
{
    TEST_ASSERT_EQUAL(
        ( sizeof(Yummies[]) / sizeof(YUMMY_T) ),
        FunctionPointerCalls);
}
```

Or If Not Practical

Leave Yourself A
Reminder

```
void test_VerifyAllFunctionPointersTested()
{
    TEST_ASSERT_EQUAL(4,
        ( sizeof(Yummies[] ) / sizeof(YUMMY_T) ) );
}
```



I Have
Similar
Tests. Can I
Save Some
Work?

yes

Tests are C Code

If You Had Repeat Code

Make A Function

Same Here, Make A
Helper Function

Look At Last Example

```
void test_VerifyCookiesCanBeMade( )
{
    void* FakeCookie = (void*)0x5a5a;
    void* Retval;

    Cookie_Make_ExpectAndReturn(FakeCookie);

    Retval = Yummy_Make("cookie");
    TEST_ASSERT_EQUAL(FakeCookie, Retval);
}
```

Break Out Helper

```
void HelperVerifyYummyCanBeMade(  
                                char* NameOfYummy,  
                                EXPECT* ExpectPtr)  
{  
    void* FakeYummy = (void*)0x5a5a;  
    void* Retval;  
    ExpectPtr(FakeYummy);  
    Retval = Yummy_Make(NameOfYummy);  
    TEST_ASSERT_EQUAL(FakeYummy, Retval);  
}
```

```
void testVerifyCookiesCanBeMade()  
{  
    HelperVerifyYummyCanBeMade(  
        "cookie",  
        Cookie_Make_ExpectAndReturn(FakeYummy));  
}
```

A Few Things To Note

```
void HelperVerifyYummyCanBeMade(  
                                char* NameOfYummy,  
                                EXPECT* ExpectPtr)  
{  
    void* FakeYummy = (void*)0x5a5a;  
    void* Retval;  
    ExpectPtr(FakeYummy);  
    Retval = Yummy_Make(NameOfYummy);  
    TEST_ASSERT_EQUAL(FakeYummy, Retval);  
}  
  
void testVerifyCookiesCanBeMade()  
{  
    HelperVerifyYummyCanBeMade(  
        "cookie",  
        Cookie_Make_ExpectAndReturn(FakeYummy));  
}
```

Named To Auto-
Recognize Tests

```
void HelperVerifyYummyCanBeMade(
    char* NameOfYummy,
    EXPECT* ExpectPtr)
{
    void* FakeYummy = (void*)0x5a5a;
    void* Retval;
    ExpectPtr(FakeYummy);
    Retval = Yummy_Make(NameOfYummy);
    TEST_ASSERT_EQUAL(FakeYummy, Retval);
}

void testVerifyCookiesCanBeMade()
{
    HelperVerifyYummyCanBeMade(
        "cookie",
        Cookie_Make_ExpectAndReturn(FakeYummy));
}
```

(Would Not Exist)

```
void HelperVerifyYummyCanBeMade(
    char* NameOfYummy,
    EXPECT* ExpectPtr)
{
    void* FakeYummy = (void*)0x5a5a;
    void* Retval;
    ExpectPtr(FakeYummy);
    Retval = Yummy_Make(NameOfYummy);
    TEST_ASSERT_EQUAL(FakeYummy, Retval);
}

void testVerifyCookiesCanBeMade()
{
    HelperVerifyYummyCanBeMade(
        "cookie",
        Cookie_Make_ExpectAndReturn(FakeYummy));
}
```

Could Be Called

Outside Helper

Avoid Custom Type

```
void HelperVerifyYummyCanBeMade( char* NameOfYummy,
                                  void* FakeYummy )
{
    void* Retval;
    ExpectPtr(FakeYummy);
    Retval = Yummy_Make(NameOfYummy);
    TEST_ASSERT_EQUAL(FakeYummy, Retval);
}
```

```
void testVerifyCookiesCanBeMade()
{
    void* FakeYummy = (void*)0x5a5a;
    Cookie_Make_ExpectAndReturn(FakeYummy) )
    HelperVerifyYummyCanBeMade(
        "cookie",FakeYummy);
}
```



I Don't Care
How Often
FuncA Gets
Called... Just
FuncB

Or Similarly

I Don't Care
What Was
Passed Into
The Mock...
Just What It
Returns!



No Problem

CMock Has Plugins

Enable Ignores

Get Additional Options

Functions We Ignore When Called

```
void BeSweet(char* OptionalParams)
```



```
void BeSweet_Ignore(void)
```

Functions We Ignore Parameters But Still Return Values

```
int BeGroovy(char* OptionalParams)
```



```
void BeGroovy_IgnoreAndReturn(int  
    toReturn)
```

Return Can Still Be
Ordered

But Extras Will Repeat
Last Value

```
Cleanify_IgnoreAndReturn(0);  
Cleanify_IgnoreAndReturn(1);  
Cleanify_IgnoreAndReturn(2);
```

...

```
Cleanify() => 0  
Cleanify() => 1  
Cleanify() => 2  
Cleanify() => 2  
Cleanify() => 2
```

Must Enable
Ignore Plugin



The Mock
Needs To
Take A
Reference
To A Local
Variable

For Example

How Do You Test This?

```
int A = 2;

void FunctionToBeTested(int B)
{
    int C;

    Mock(A, B, &C);
}
```

```
int A = 2;

void FunctionToBeTested(int B)
{
    int C;

    Mock(A, B, &C);
}
```

```
//A is Easy, in the test file:
extern int A;
```

```
...
Mock_Expect(A, ...);
```

```
int A = 2;

void FunctionToBeTested(int B)
{
    int C;

    Mock(A, B, &C);
}
```

```
//B is also Easy. We passed it in:
extern int A;
int B = 6;
...
Mock_Expect(A, B, ...);

FunctionToBeTested(B);
```

```
int A = 2;

void FunctionToBeTested(int B)
{
    int C;

    Mock(A, B, &C);
}

//C???
```

Unity To The Rescue

It #defines TEST

```
int A = 2;  
#ifdef TEST  
    int C;  
#end
```

```
void FunctionToBeTested(int B)  
{  
#ifndef TEST  
    int C;  
#end
```

```
    Mock(A, B, &C);  
}
```

```
//Now We Can Use C Just Like A
```

But Usually

We Just Avoid Doing It



I'm Using
Someone
Else's
Library

Just Mock It

Set Up A Header

MyLib.h

Contains Declarations

For API We Care About

Then, #Include
In Test Files

```
#include “MockMyLib.h”
```

Real Code Links To
Real Library

Test Code Links To
Mocked Version

Avoids Nasty Chains Of
Includes

Avoids “Getting All The
Defines Right”

Serves As
Documentation To
What API Functions
Are Being Used

Small Variation

Instead of 1 Lib Header

Organize Functions

Into Multiple

API_Tasks.h

API_Semaphores.h

API_Queues.h

API_Timers.h

Faster Mock Generation

Further Documents

If Using MCH with OS

Conductor
Become Tasks

Conductor
Can Wait For Events

Events Fired From Interrupts In Hardware

Or Events Fired From
Model Calls

Events Are Faked

By Mocking Receive
Functions

Is Your Conductor
Periodic?

Mock The Task Wait Function To Test

Mocking Other People's
Work Is Always Fun

My Function
Returns An
Ugly Struct.
Is There An
Assert For
That?



Not Exactly

Helper Functions!

If Struct Is Compacted

Pretend It's
Array Of Bytes

```
void AssertEqualMyStruct(MYSTRUCT* expected,
                         MYSTRUCT* actual)
{
    unsigned int i;
    char* e = (char*)expected;
    char* a = (char*)actual;
    for (i = 0; i < length; i++)
    {
        sprintf(message,
                "Struct failed at index %u.", i);
        TEST_ASSERT_EQUAL_INT_MESSAGE(
            e[i],
            a[i],
            message);
    }
}
```

Otherwise

Check Each Element

```
void AssertEqualMyStruct(MYSTRUCT* e,
                         MYSTRUCT* a)
{
    TEST_ASSERT_EQUAL_INT_MESSAGE(
        e->MyInt, a->MyInt, "MyInt Fails");
    TEST_ASSERT_EQUAL_HEX16_MESSAGE(
        e->MyHex, a->MyHex, "MyHex Fails");
    TEST_ASSERT_EQUAL_FLOAT_WITHIN_MESSAGE(
        0.2, //Float Tolerance
        e->MyFloat,
        a->MyFloat,
        "MyInt Fails");
}
```



My Error
Handling
Manipulates
The Stack

CException

Light Wrapper

`setjmp / longjmp`

```
void func A() //I do nothing except complain
{
    Throw(0x1234);
}
```

```
int func B() //I clean Up A's Mess
{
    Try
    {
        A();
        C(); //I never get called
    }
    Catch
    {
        return EXCEPTION_ID;
    }
    return 0;
}
```

Again

CMock Plugin

Adds New Option

If We Want To Pretend To Throw An Error When Our Mock Is Called

```
void BeBest(int Param)
```



```
void BeBest_ExpectAndThrow(  
    int Param, int IdToThrow)
```

Which Works With
Normal Expects

```
Swankify_Expect(20);  
Swankify_Expect(18);  
Swankify_ExpectAndThrow(16, 0x9876);
```

...

```
Swankify(20) => validates param was 20  
Swankify(18) => validates param was 18  
Swankify(16) => validates param was 16,  
                  then Throws ID 0x9876
```

...

```
Swankify(20) => validates param was 20  
Swankify(16) => test fails! should match 18
```

We Can Also

Verify Catches

```
void test_VerifyMonkeyShouldThrowIfGetsATwo()
{
    Try
    {
        Monkey(2);
        TEST_FAIL("Should Have Thrown");
    }
    Catch
    {
        TEST_ASSERT_EQUAL(2, EXCEPTION_ID);
    }
}
```

Ok, Nice...

What If Not
CException?

Directly using
setjmp / longjmp

Co-Routines

etc.

CMock is Extensible

Write a New Plugin

Or

More Helper Functions

So,
That's
Swank



We've

Feature-Driven A Project

Mocked Others

Asserted Our Way To
Success

and

Learned Some New
Tricks

Not Bad For
Nearly 400 Slides

All That's Left

Free Stuff:

CMock

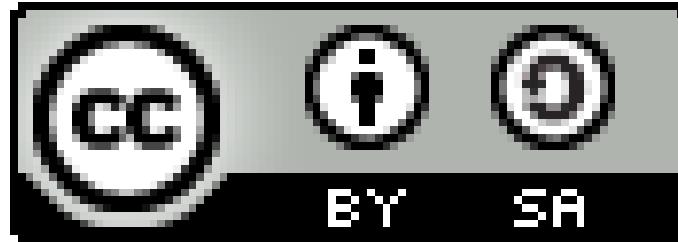
<http://cmock.sourceforge.net/>

Unity

<http://embunity.sourceforge.net/>

CException

<http://sourceforge.net/projects/cexception>



This Presentation is Licensed Under
a Creative Commons 3.0 Attribution,
Share-Alike License.

you can use all or part of this for whatever you want,
as long as you

- (1) credit Mark VanderVoord and Greg Williams
- (2) release derived works under a similar license.)

<http://www.creativecommons.org/>

Images By *(thanks!)*:



Question Mark - Stefan Baudy
Ginger Bread - Kevin "KB35"



Clock - Christina Castro
Glue - Jenny "mennyj"
Batteries - Ian Britton



Traffic Lights (2) - "flrnt"
Peach - Rick Harris



Duck - "LavenderLady"
Version - Travis Forden
Ruby - "Afternoon Sunlight"
Hilary Swank - Bruno Chatelin