

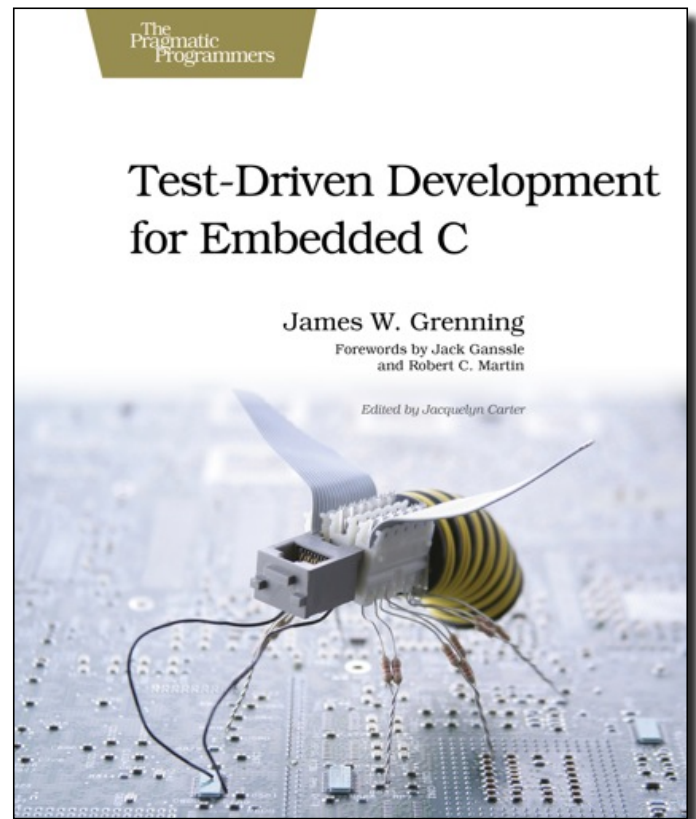


Test-Driven Development for Embedded C/C++

Web Delivered Live
March 24-26, 2020

by
James W Grenning
james@wingman-sw.com
www.wingman-sw.com
www.twitter.com/jwgrenning

For the Exclusive use of training attendees



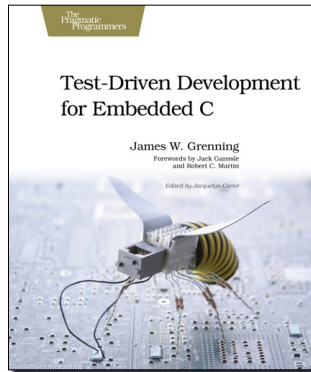


Test-Driven Development for Embedded C/C++

Web Delivered Live
March 24-26, 2020

by
James W Grenning
james@wingman-sw.com
www.wingman-sw.com
www.twitter.com/jwgrenning

For the Exclusive use of training attendees



WEB-24



Test-Driven Development for Embedded C/C++

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered

www.wingman-sw.com
james@wingman-sw.com

2

Test Driven Development Topics

➡ Formalities

- ➡ Test Driven Development
- ➡ TDD Adapted to Embedded
- ➡ TDD with Dependencies
- ➡ Clean Tests
- ➡ Mocks

- ➡ Refactoring Overview
- ➡ Legacy Code Overview

➡ The End

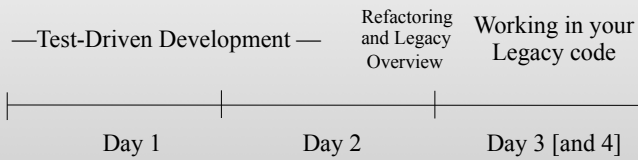
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered

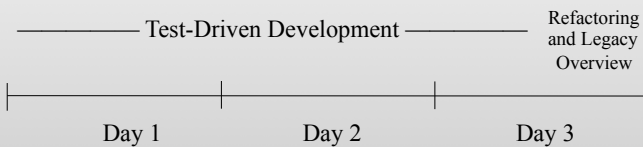
www.wingman-sw.com
james@wingman-sw.com

3

Topics and Timeline On-site TDD Courses



Topics and Timeline Remote Delivered TDD Courses

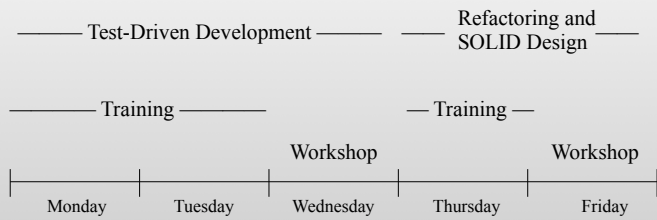


Three five-hour days.
Possible for delivery from the USA to Europe

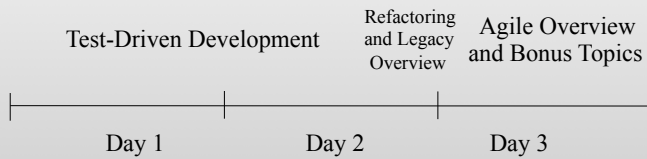
Topics and Timeline Design and Refactoring Courses



TDD and Refactoring



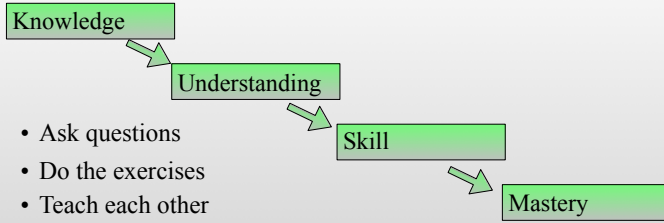
Topics and Timeline On-site Courses



Formalities

- Sign-in sheet
- Introductions
- Hours
- Breaks
- Expectations

Learning Model



- Ask questions
- Do the exercises
- Teach each other
- Get your own experience
- Build your skill

“If you don't want to be wrong, don't ask.
If you never want to be right, don't ask.”

Gabrielle Benefield



Test Driven Development

What do you
like about
development?

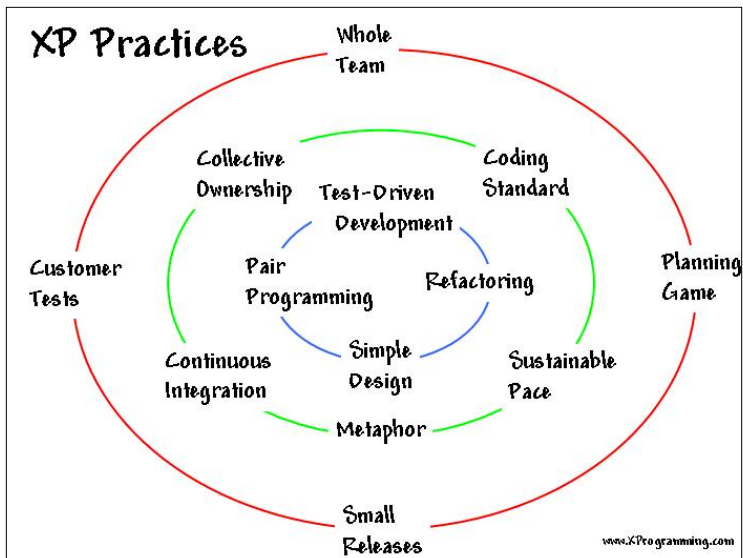
What is your
least favorite
development
activity?

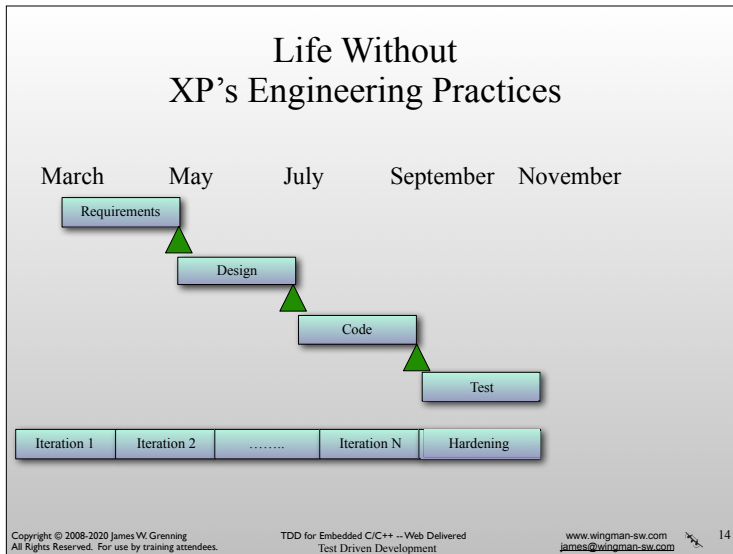
Why are you
here?

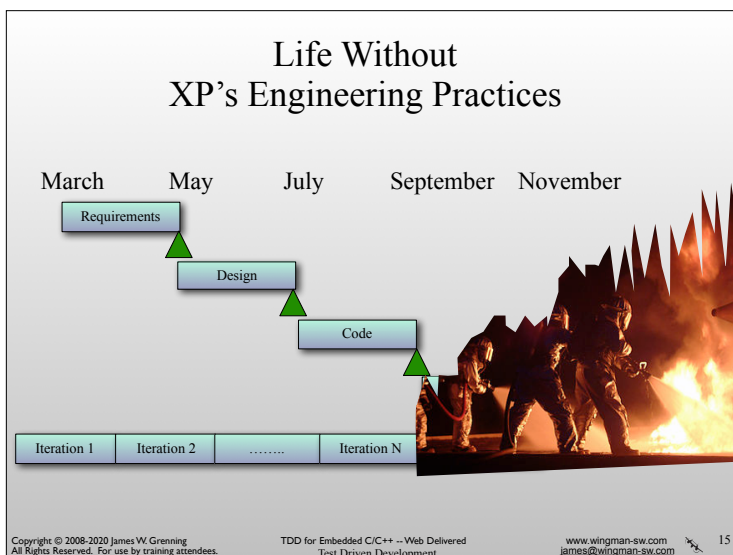
I like...

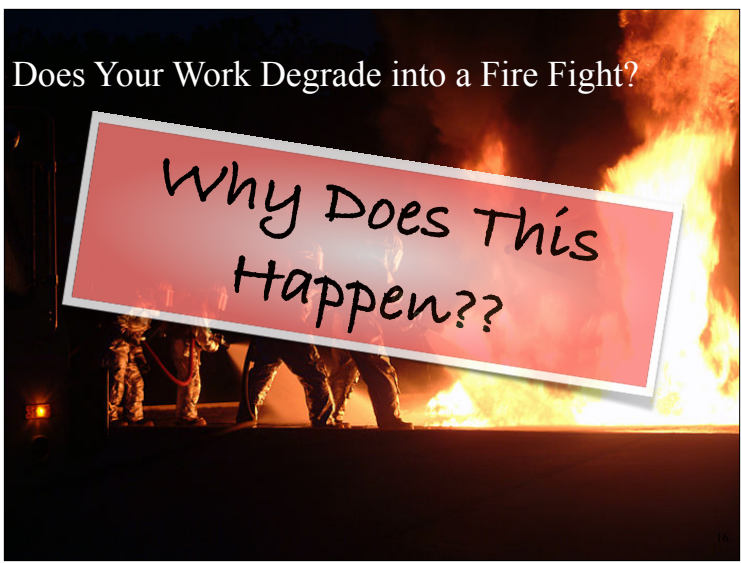
I don't like...

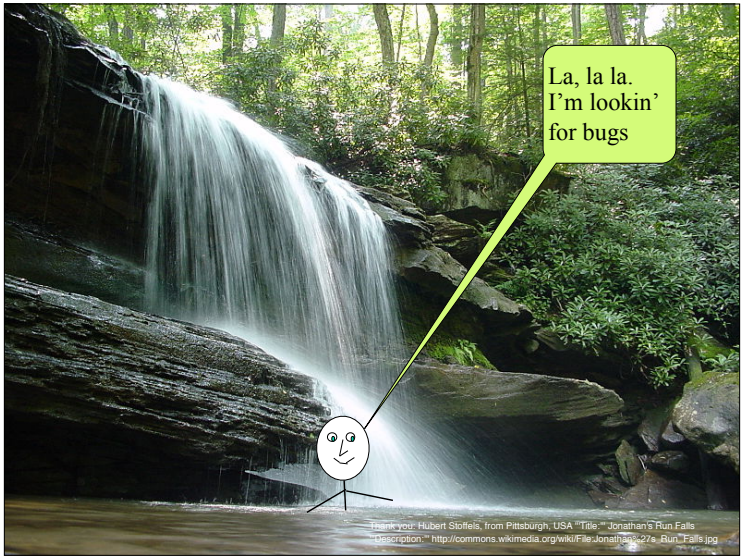
I'm here because ...





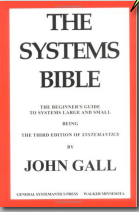








Your program will have bugs. And they will surprise you when you find them.



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Driven Development

www.wingman-sw.com
james@wingman-sw.com

19

The Last Bug



Here's to a project finished at last!

To 1293 fixed bugs!

Hey wait, what's that?

Oh no, a bug!

He must have managed to pass QA checks

Not for long!

There you go, dead as a doornail

Phew

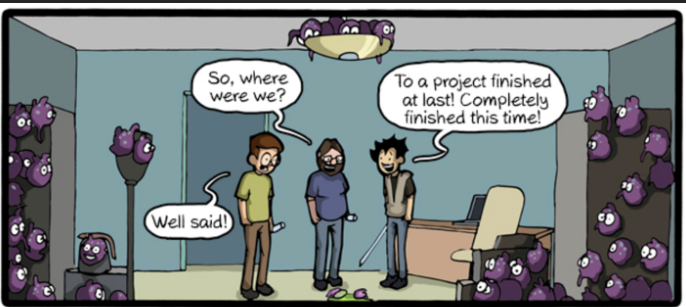
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Driven Development

www.wingman-sw.com
james@wingman-sw.com

20

Finished at Last



So, where were we?

Well said!

To a project finished at last! Completely finished this time!

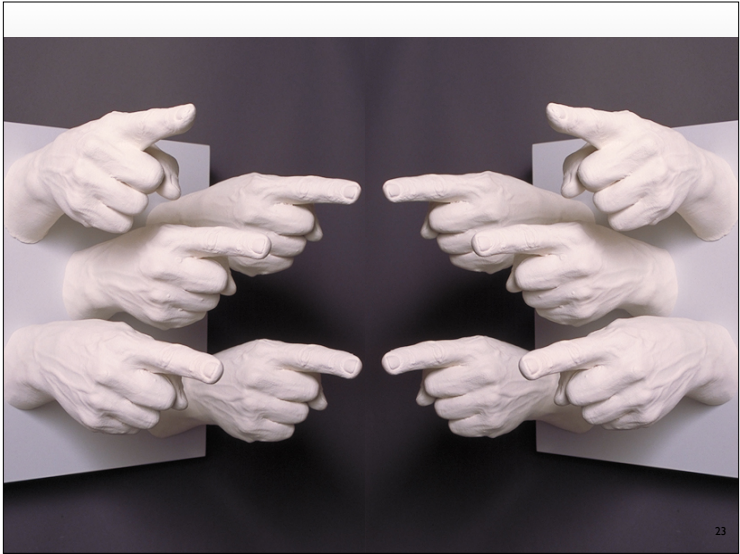
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

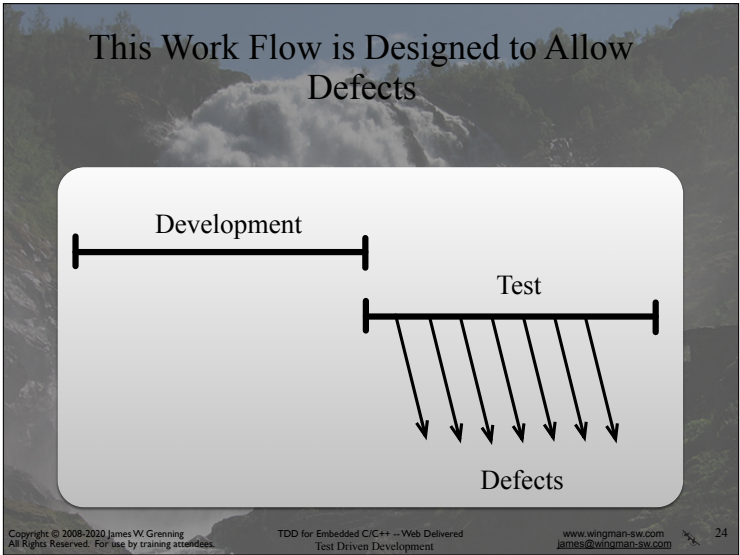
TDD for Embedded C/C++ -- Web Delivered
Test Driven Development

www.wingman-sw.com
james@wingman-sw.com

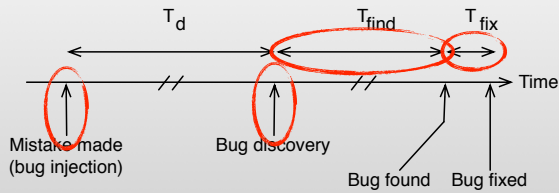
21





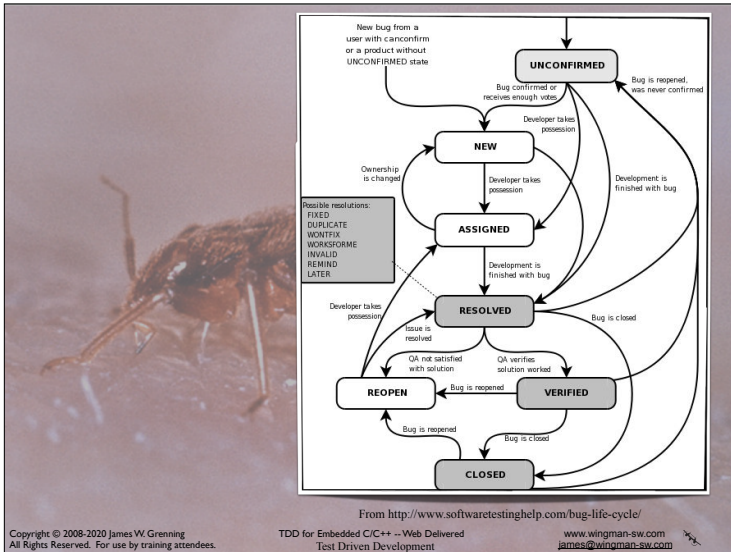


The Physics of Debug Later Programming (DLP)



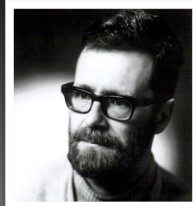
- As T_d increases, T_{find} increases dramatically
- T_{fix} is usually short, but can increase with T_d

<https://wingman-sw.com/articles/the-physics-of-test-driven-development>



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.



This Test Could Have Prevented The Zune Bug

```
TEST(Rtc, check20081231)
{
    days = daysSince1980(2008, 366);
    CHECK(ConvertDays(days, &time));
    assertDate(WED, 2008, 12, 31);
}
```



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

THE
SYSTEMS
BIBLE
THE BEGINNER'S GUIDE
TO SYSTEMS ANALYSIS AND DESIGN
BY
THE THIRD EDITION OF JEFFREY GALL
BY
JOHN GALL
LONDON: JOHN WILEY & SONS, INC. 1992

Bugs Can Hide Anywhere. What has to be tested?



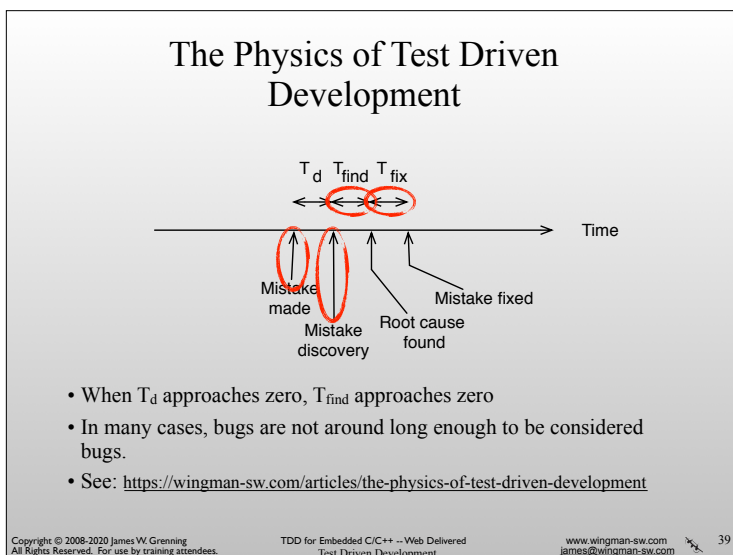
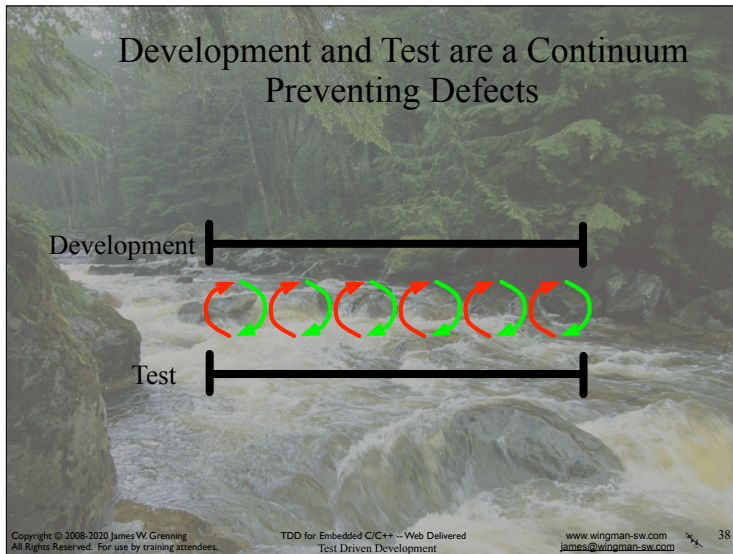
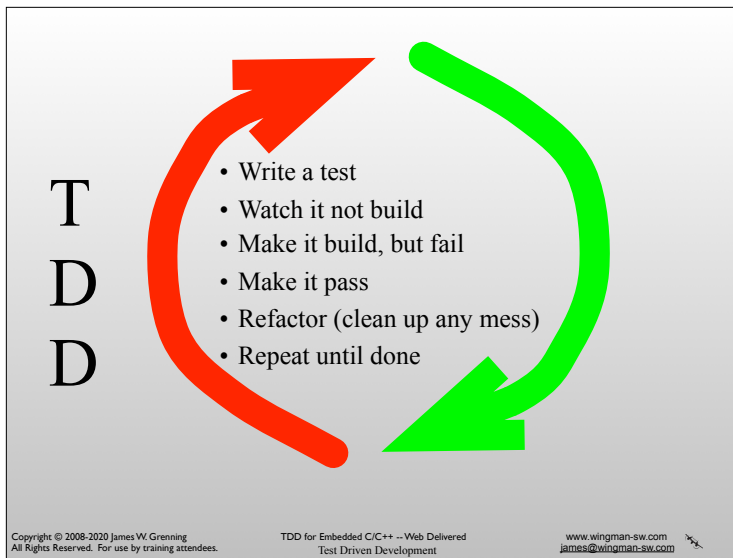
CommitStrip.com

Can we Realize Dijkstra's Vision?

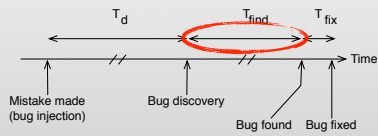
Can we Prevent Defects
with Test-Driven
Development?

Being good at chasing bugs
is not Technical Excellence

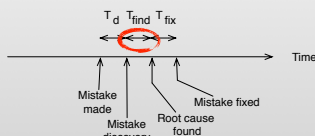




The Physics of Debug Later Programming



The Physics of Test-Driven Development



TDD Demo and Exercise

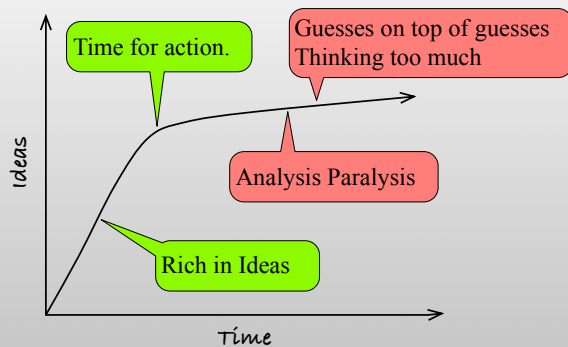
The Need

- The Circular Buffer holds a series of integers.
- Create a Module that manages the usage and integrity of a Circular Buffer.
- A full Circular Buffer rejects new values
- An empty buffer returns a default value
- No concurrent access - populated and drained from the main loop.

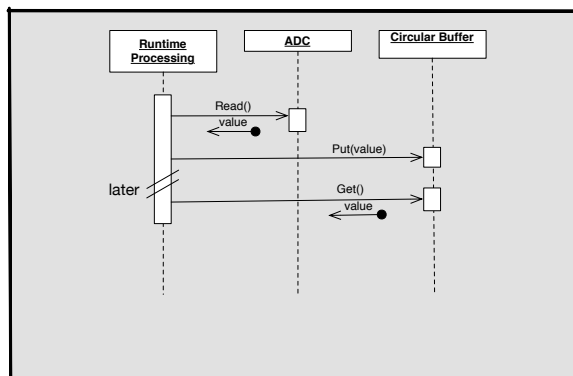
Before We Start Test Driving

- What does the API look like?
 - Name the functions
- How would you test it?
 - Describe in a few sentences
- What test cases do you need?
 - Make a list
- What does its internal data structure look like?
 - Identify the fields you anticipate

Avoid Analysis Paralysis



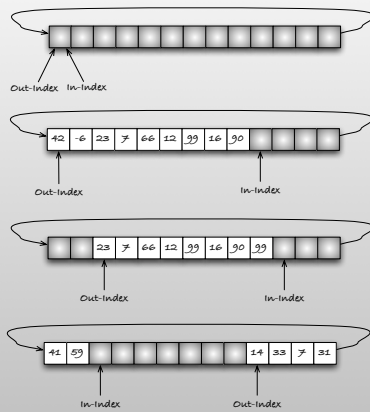
No Concurrency Concerns for v1



The Anticipated Interface

```
CircularBuffer
+ Put
+ Get
+ IsEmpty
+ IsFull
//etc
```

Create a Circular Buffer Module

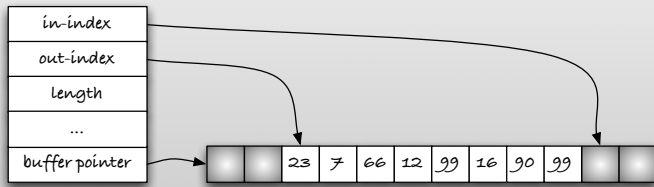


The Initial Test-List

Circular Buffer Tests

- Initially Empty
- Transition to empty
- Transition to Full
- Transition from Full
- Put-Get FIFO
- Put to full
- Get from empty
- Filled but not wrapped around
- Wrap around

Anticipated Internal Structure



Demo

```
TEST(CircularBuffer, empty_after_creation)
{
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
}
```



```
bool CircularBuffer_IsEmpty(CircularBuffer * self)
{
    return false;
}
```

The Simplest Implementation
that Passes the compiler/linker
and fails the Test

```
bool CircularBuffer_IsEmpty(CircularBuffer * self)
{
    return true;
}
```

The Simplest Implementation
That Passes All Tests

```
TEST(CircularBuffer, empty_after_creation)
{
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
}

TEST(CircularBuffer, not_empty_after_put)
{
    CircularBuffer_Put(buffer, 42);
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
}
```

```
bool CircularBuffer_IsEmpty(CircularBuffer * self)
{
    return true;
}

void CircularBuffer_Put(CircularBuffer * self, int value)
{
}
```

The Simplest Implementation
That Passes the prior test and
fails the new test

```
bool CircularBuffer_IsEmpty(CircularBuffer * self)
{
    return self->index == 0;
}

void CircularBuffer_Put(CircularBuffer * self, int value)
{
    self->index++;
}
```

The Simplest Implementation
That Passes All Tests

```
TEST(CircularBuffer, empty_after_creation)
{
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
}

TEST(CircularBuffer, not_empty_after_put)
{
    CircularBuffer_Put(buffer, 42);
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
}

TEST(CircularBuffer, empty_after_removing_the_last_item)
{
    CircularBuffer_Put(buffer, 42);
    CircularBuffer_Get(buffer);
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
}
```

```
bool CircularBuffer_IsEmpty(CircularBuffer * self)
{
    return self->index == 0;
}

void CircularBuffer_Put(CircularBuffer * self, int value)
{
    self->index++;
}

int CircularBuffer_Get(CircularBuffer * self)
{
    return -1;
}
```

The Simplest Implementation
That Passes All Tests

```
bool CircularBuffer_IsEmpty(CircularBuffer * self)
{
    return self->index == self->outputIndex;
}

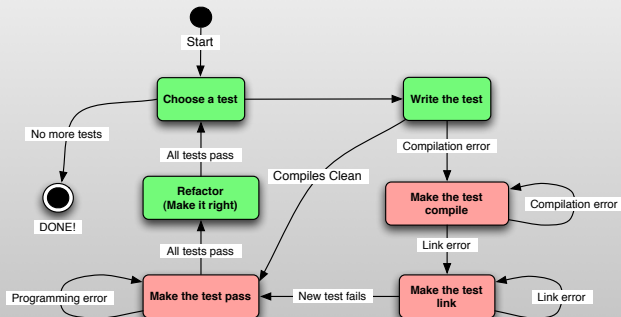
void CircularBuffer_Put(CircularBuffer * self, int value)
{
    self->index++;
}

int CircularBuffer_Get(CircularBuffer * self)
{
    self->outputIndex++;
    return -1;
}
```

The Simplest Implementation
That Passes All Tests

Continue Until You Can't Think of Any More Tests

When your tests are done and your code is done.



Exercise

Exercise objectives

- Experience the cadence of TDD
- Work in small steps, solving one problem at a time
- Get more experience with CppUTest
- Experience pair programming
 - Take turns at the keyboard
 - Discuss the tests and the needed production code
 - Keep the programmer at the keyboard from coding ahead of the tests
 - “Do you have a test for that?”

Pair Programming Guidelines

For Both:

- Ask: do you have a test for that??
- Stay humble (or at least pretend to be)

For the "more experienced":

- Stay humble (or at least pretend to be)
- Explain what you are doing (and why, if you are knowledgeable, not just years in the seat)
- Pass the keyboard/mouse to the other.
- Tell him/her what to type if you have to...
- Ask the other what to do next...

For the "less experienced:

- Stay humble (or at least pretend to be) you knew I was going to do that, didn't you?
- Ask questions.
- Don't let anything get by you don't understand.
- Catch typos.
- Think.
- Pay attention.
- If you start to zone out, do "something" to zone in.
- Request to "drive" even if the "more experienced has to "dictate" code.

thanks to Kay Pentecost

Test Driven Development Debrief

Is TDD Software Engineering?



Kent Beck @KentBeck

37s

tdd is a great excuse to think about the problem before you think about the solution

Engineering combines the fields of science and maths to solve real world problems that improve the world around us. What really distinguishes an engineer is his ability to implement ideas in a cost effective and practical approach. This ability to take a thought, or abstract idea, and translate it into reality is what separates an engineer from other fields of science and mathematics.

from <http://whatisengineering.com/>

Coding is Dangerous Work Carefully

- TDD is solving one problem at a time.
- At first we focus on the tests, with incomplete code. Defining one behavior at a time.
- Once all the tests are done, the code is ready for integration.



Photo by Andrew Bost (Own work) [CC-BY-SA-2.5 (www.creativecommons.org/licenses/by-sa/2.5)], via Wikimedia Commons

Some of the Common Concerns

Debrief

TDD Will Take Too Long!



But that is so
Inefficient!
So Slow!
Show us the fast way!

How Would You Have Tested a CircularBuffer?



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

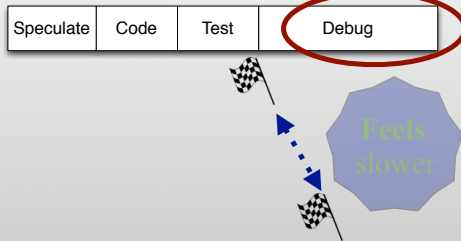
www.wingman-sw.com
james@wingman-sw.com

70

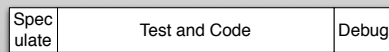
Misconception and Sustainability

Time developers
do not notice nor
plan for

Traditional
development



Test-driven development



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

www.wingman-sw.com
james@wingman-sw.com

71

How are you Spending your Time When your Compiler is Busy?

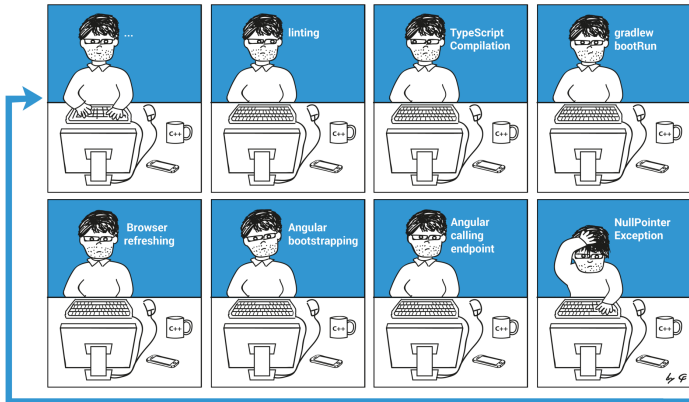


Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

www.wingman-sw.com
james@wingman-sw.com

72



<https://www.rainerhahnekamp.com/en/why-we-test-do-things-faster-with-test-driven-development/>

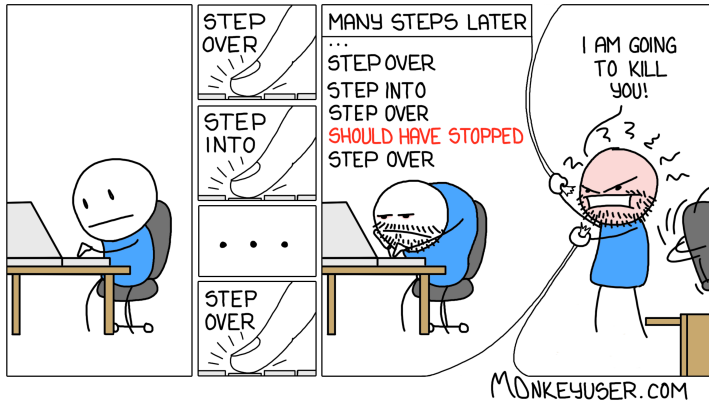
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

www.wingman-sw.com
james@wingman-sw.com

73

STEP BY STEP DEBUGGING



<http://www.monkeyuser.com/2017/step-by-step-debugging/>

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

www.wingman-sw.com
james@wingman-sw.com

74

Too Many Lines of Code

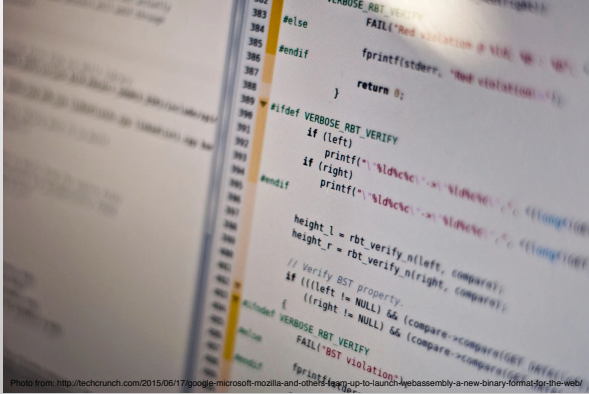
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

www.wingman-sw.com
james@wingman-sw.com

75

TDD Means Too Many Lines of Code? Think Again.



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

www.wingman-sw.com
james@wingman-sw.com

76

We Keep Jumping Around

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

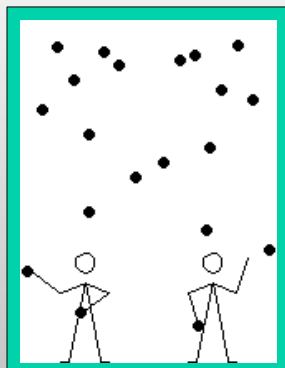
TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

www.wingman-sw.com
james@wingman-sw.com

77

Debug Later Programming is Like Juggling

- TDD helps to manage complexity
 - Records progress one behavior at a time
 - Modify one variant at a time
 - Regression tests warn when an existing behavior is compromised



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test-Driven Development Debrief

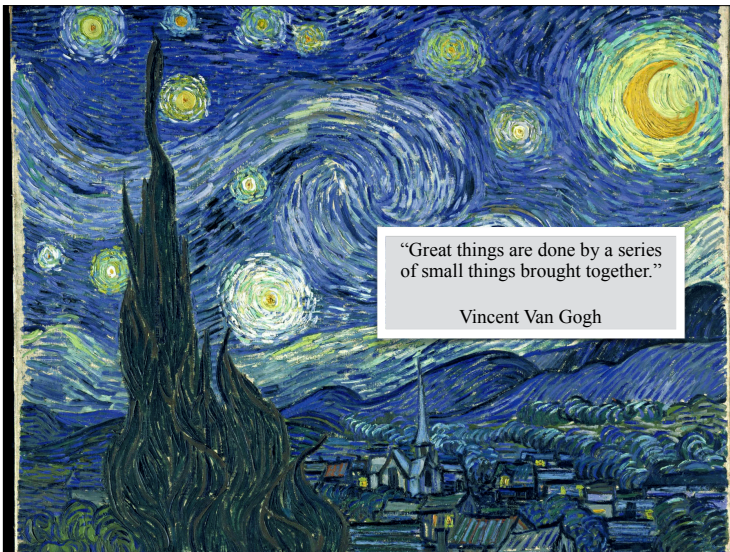
www.wingman-sw.com
james@wingman-sw.com

78

Small Deliberate and Verified Steps

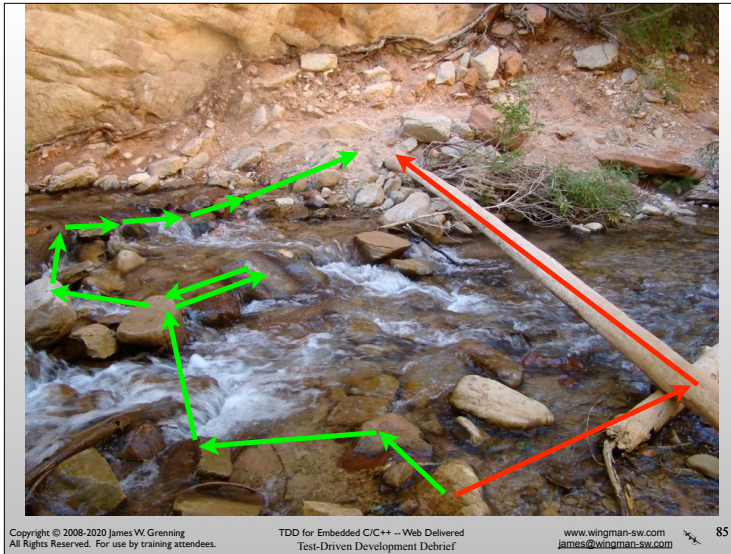
- Have a hypothesis of what will happen when you press test.
- Have your code work on purpose.
- Each change is verified
 - Maybe by seeing the test output
 - Maybe by seeing how the tools react
 - Maybe with a failing test
 - Maybe with a passing test

The Steps are So Small!

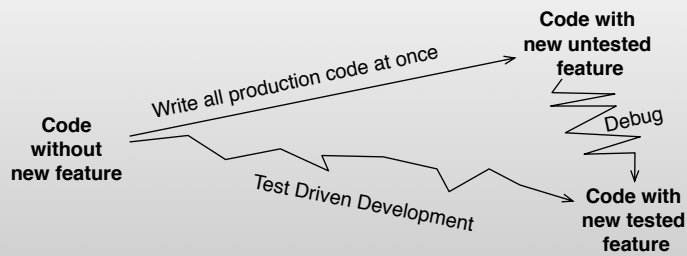


“Great things are done by a series
of small things brought together.”

Vincent Van Gogh



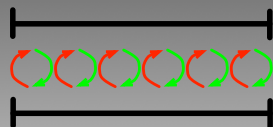
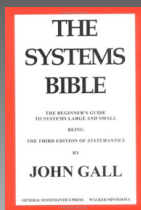
TDD Progress in the Right Direction



- See: <https://wingman-sw.com/articles/tdd-stepping-stones>

Grow Full Capability One Behavior at a Time

A Complex system that works is invariably found to have evolved from a simple system



I Don't Know What Tests to Write!

TDD Guided by ZOMBIES

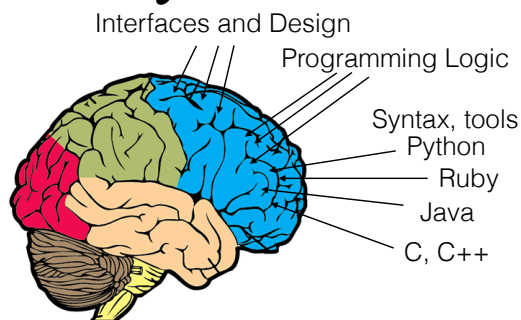


Zero	Simple Scenarios Simple Solutions		
One			
Many			
	B o u n d a r i e s	I n t e r f a c e s	E x c e p t i o n s

Zombies photo thanks to Joel Friesen

copyright (c) 2016 James W. Grenning

Exercising Different parts of your brain



Transformation Priority Premise

“As the tests get more specific,
the code gets more generic.”

Further Reading

<https://8thlight.com/blog/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html>

Thanks Robert Martin

The Transforms

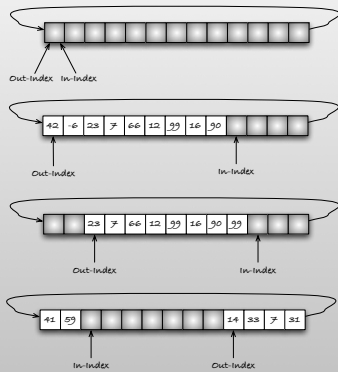
- ({}->nil) no code at all->code that employs nil
- (nil->constant)
- (constant->constant+) a simple constant to a more complex constant
- (constant->scalar) replacing a constant with a variable or an argument
- (statement->statements) adding more unconditional statements.
- (unconditional->if) splitting the execution path
- (scalar->array)
- (array->container)
- (statement->recursion)
- (if->while)
- (expression->function) replacing an expression with a function or algorithm
- (variable->assignment) replacing the value of a variable.

I'll have to maintain the tests!

The Tests Support Design Changes

- The original design did not use an interrupt to collect ADV values.
- Values were read on one pass through the main loop and consumed later.
- The new design uses interrupts instead of a polling loop!
- Every 10 milliseconds, get an ADC value.
- Save it in a CircularBuffer
- During the main loop process the ADC values.

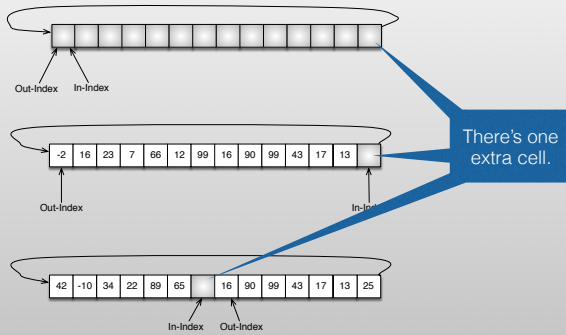
Initial Design Uses a counter or a flag for IsFull



We have a race condition!

- CircularBuffer could use the extra-cell design for single-producer and single-consumer to avoid the race.
- What would be your retest time if you did not have thorough automated tests for your CircularBuffer?

The buffer is full if
`next(in_index) == out_index`



Experience How Automated Tests Support Design Changes

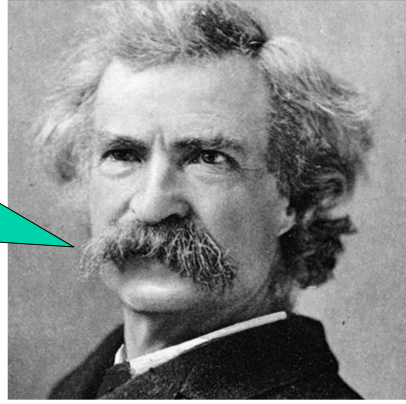
- For practice refactor your design to use the extra cell method.
- Do you need any other tests?
- You can also watch a video of James modifying the design with the protection of the tests.
 - <https://wingman-sw.com/courses/demos/c/circular-buffer#alloc>

It Feels Like a Lot More Work

Maybe it's Just Uncomfortable

Mark Twain

"It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so."



Productivity and Predictability

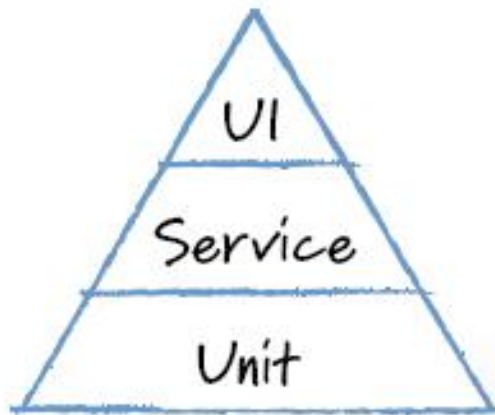
- Defects kill predictability:
 - Cost of fixing is not predictable
 - When they materialize is not predictable
- Test-driven is predictable:
 - Working at a steady pace
 - Results in fewer bugs
 - More productive than “*debug-later programming*”
- Test-driven programmers rarely need the debugger



Unit Tests are Necessary but Not Sufficient

There are Different kinds of Automated Tests

- Unit Test
 - Feedback to the developer that the code does what is expected
 - Written using a Unit Test Harness (e.g. unity, CppUTest, ...)
- Story Tests - Executable use cases - Service Level Tests
 - Feedback to the *Customer* that the code meets the requirements
 - Used at many levels
 - Component
 - Groups of integrated components
 - System
 - Written in a domain specific language (e.g. FitNesse) [FITNESSE]
- Integration, System, Load Tests, Exploratory, Usability...

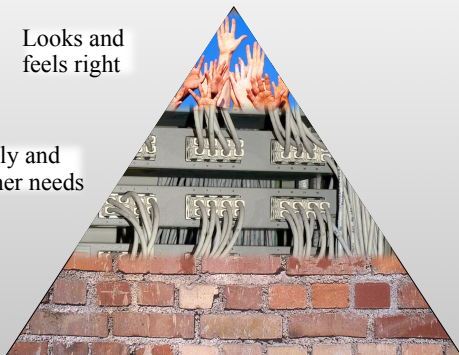


<http://blog.mountaingoatsoftware.com/the-forgotten-layer-of-the-test-automation-pyramid>

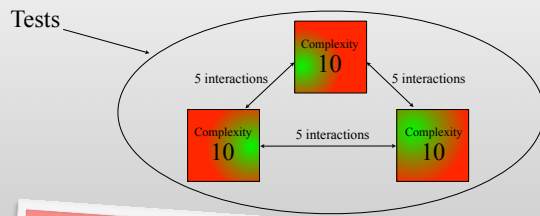
Looks and
feels right

Wired properly and
meets customer needs

Solid
foundation



Higher Level Tests Needed for Thorough Testing

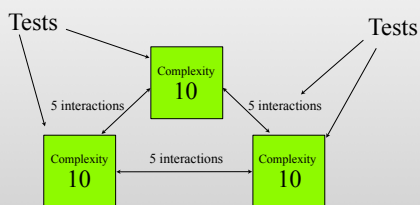


1000 (or more) tests are needed to test this simple system

Higher Level Tests are Needed to Verify Functionality and Interconnections



Unit Tests Can Be Thorough



As few as 30 unit tests and 15 integration test when tested as units

A Few Other Things to Know

Tests are FIRST

F_{ast}

I_{ndependent}

R_{epeatable}

S_{elf-validating}

T_{imely}

Thank you: Tim Ottinger

Tests Should Be Quiet Unless Something is Wrong

- By default, tests are silent unless there is an error.
- Use CppUTest executable switches for more feedback
 - Verbose (-v)
 - Filter by TEST_GROUP (-g substring)
 - Filter by TEST name (-n substring)
- Verbose output helps when tracking down
 - test case crashes
 - static initialization problems

Adaptation for Embedded Software

What is Special About Embedded Software?

Limited memory

Hard to debug

HW has bugs

Concurrent HW/SW development

Primitive UI

Special tools

No HW until late in cycle

Real time debugging



How Long is Your Edit, Build, Run Cycle?

- Do you lose focus while you wait?



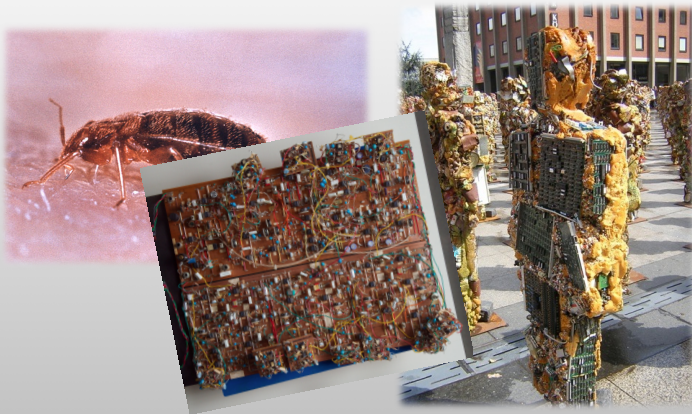
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Driven Development - Embedded

www.wingman-sw.com
james@wingman-sw.com

115

Hardware has its own Defects



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Driven Development - Embedded

www.wingman-sw.com
james@wingman-sw.com

116



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Driven Development - Embedded

www.wingman-sw.com
james@wingman-sw.com

117



Minimize DoH!



Debug
On
Hardware

Copyright Fox Broadcasting. Used under fair use.

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Driven Development - Embedded

www.wingman-sw.com
james@wingman-sw.com

119

Target Hardware Bottleneck

- Concurrent HW development
- Cross compilation
- Limited memory and IO
- Target debug
- Hardware is scarce
- Hardware has its own defects



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Driven Development - Embedded

www.wingman-sw.com
james@wingman-sw.com

120

Seems Embedded Engineers Think They Must Live with These Problems

- See survey replies from people about to take my training course

– <https://wingman-sw.com/attendee-preparations/current-practice>

- How did you answer those questions?
- Are you living with problems you don't even consider problems?
- Do you think that is just how it has to be?

Use Your Development System for a Test Bed!

Avoid Wasteful Activities

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

New Considerations When Using Your Development System for a Test Bed

Multi-target your code

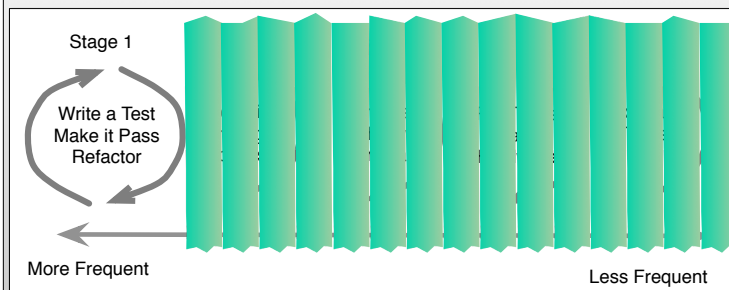
- Must make code portable
- Must beware of hardware and OS dependencies.
- Object Oriented approach to Dependency Management.

New Risks When Using Your Development System for a Test Bed

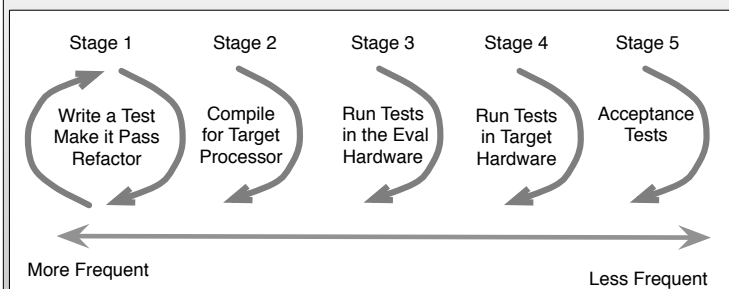
Beware of the Risks

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

TDD Adaptation for Embedded Development



TDD Adaptation for Embedded Development

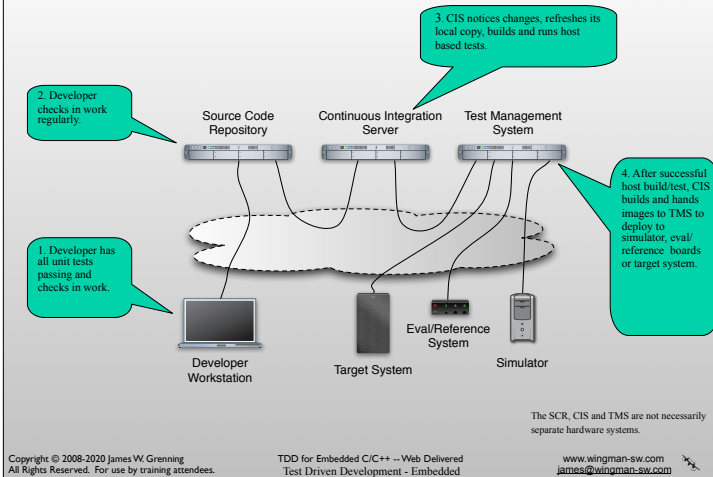


See : <https://wingman-sw.com/articles/progress-before-hardware>

Problems Found at Appropriate Stage

Stage	Problems Likely to Find in Stage
1	Logic, interface, boundary conditions, design, modularity
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

Continuous Integration - Embedded



Continuous Integration

- The practice
 - Merges are risky
 - Merging is not fun, do it more often
 - Check in frequently, whenever all tests are passing
- The tools
 - Source repository (svn, git, ...)
 - CI server (Jenkins, CruiseControl, ...)

“Nightly Builds are for Wimps” -- Martin Fowler

What you need for Continuous Integration

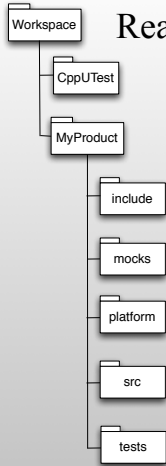
- Single command to build and test
- In embedded
 - Stages 1-4 are automated
 - They are run in order automatically upon checkin
 - Stop on failure
- That means
 - Automated build
 - Automated test run
 - Automate deploy to target for stages 3-4
- Notify the team when the build is broken

Discuss

- How would you do this in your team?

Dual-Target Build Approach

- One workspace containing test code and production code
- All code is under version control
- There is one source tree
- One or more test builds
- One or more production builds
- Builds and test runs should be automated



Reasonable Workspace Structure

- **MyProduct** might be a whole system or a subsystem. You may have multiple products
- **include** contains header files that define public interfaces
- **mocks** contains test stubs
- **platform** contains platform specific production code
- **src** contains production code
- **tests** contains test cases and maybe some stubs too
- There is usually a directory structure under **include**, **mocks**, **src** and **tests**, depending on your needs

Embedded is Special

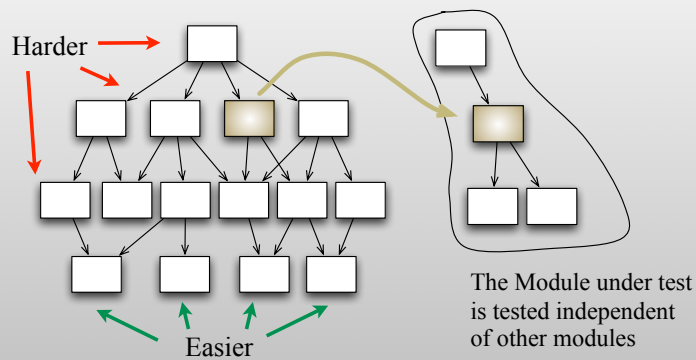
- The Target Hardware Bottleneck is a Problem
- Test-Driven Development is an additional tool to help you avoid the Target Hardware Bottleneck.



TDD and the Code in the Middle

(most of the code!)

Testing a Module in the Middle



Testing a Module in the Middle

The test case takes the
role of the client



The Code Under Test (CUT)



The servers might need
to be stubbed out



Design for Test

Object Oriented Concepts

Loose Coupling and High Cohesion

Using OO concepts

- Interfaces
 - Use header files to define the interface.
- Data hiding
 - Use file scope and discipline to hide internal data structures.
- Implementation hiding
 - Try not to let implementation details become visible through the public interface.

Design Principles

- Single Responsibility Principle [RCM]
 - Each module should have a single responsibility.
 - The responsibility can be described in one to three sentences.
- Separation of Concerns
- Substitutability

Group Exercise

Home Automation System Light Scheduling Requirements

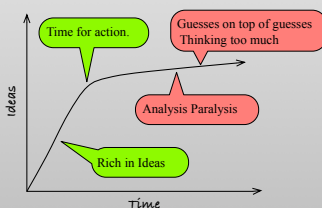
- The light scheduler maintains a schedule for lights events.
- A light can be scheduled to turn on.
- A light can be scheduled to turn off.
- 32 lights can be addressed by their integer ID.
- It must support 100 separate scheduled events
- Actions can be scheduled by
 - a specific day of the week (M Tu W Th F Sa Su), everyday, weekday and weekend day.
 - a specific minute of the day
- Lights can also be controlled though a UI on a front panel, iPad, iPhone, Android device, or web enabled device.

Group Exercise

1. Partition the problem into several areas of responsibility - High Level Design
 - Make a block diagram of this part of the system
 - UML if you like
2. How would you structure the internals of the scheduler?
3. How would you test it?
4. What test cases will you need?

Watch for Analysis Paralysis

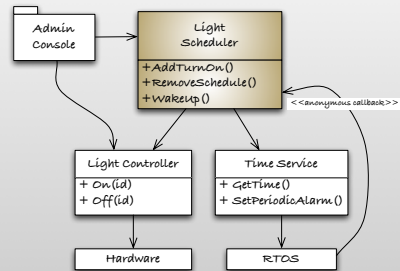
- Beware of diminishing returns on thinking ahead
- Think of the big picture, but skillfully procrastinate



Here's My Design

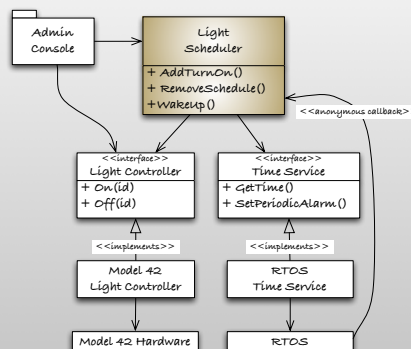
Separation of Responsibilities

- 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.



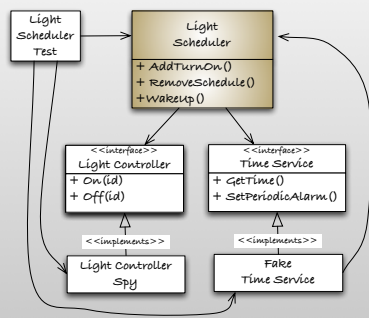
Light Scheduler Design

- 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.



Light Controller Spy



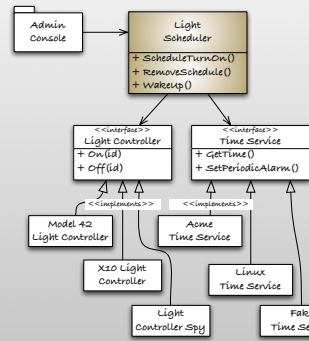
The Light Scheduler thinks it is talking to a real light controller.
Little does it know that it is being spied upon.

Typical Scheduler Test Pseudo code

- Set up a scheduled light
- Set the fake time
- Simulate the callback from OS to the Light Scheduler
- Check if the Light Scheduler turns on (or off, or not) the light only at the right time

TDD Help Break Problem Dependencies

- Must be testable
- The scheduler will manage a table of scheduled events
- Designs with partitioned responsibilities are flexible



Light Scheduler Test List

Light Scheduler Tests

Lights are not changed at initialization
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 everyday
Schedule a specific days
Remove scheduled event
Remove non-existent event
Schedule a light to turn on then later turn off
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



TDD and Collaborating Modules in C

Let's Make One Thin Slice of Functionality

Guiding Test

```
TEST(LightScheduler, lights_turned_on_at_the_right_time_scheduled_everyday)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    FakeTimeService_SetDay(SUNDAY);
    FakeTimeService_SetMinute(1200);
    LightScheduler_Wakeup();
    LONGS_EQUAL(3, LightControllerSpy_GetLastId());
    LONGS_EQUAL(LIGHT_ON, LightControllerSpy_GetLastState());
}
```

- I'd like to start with this test, but this is a big step!
- We need the scheduler interface, we need a schedule table, we need a spy light controller, we need ...
- It's time to procrastinate!
- Pick an easy place to start.

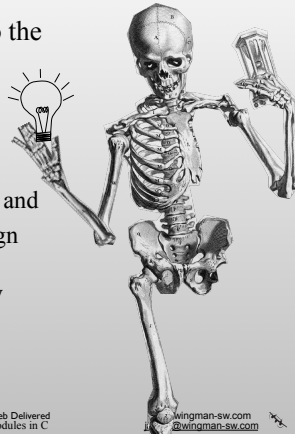
First Test-Drive the Spy



Test-Drive the Spy
keep it simple, we can add more to it later

Create a Walking Skeleton of the Interacting Objects

- Add one test for each call into the Light Scheduler
- These initial tests
 - are easy to make pass,
 - define a boundary condition,
 - provide a first use of key APIs, and
 - provide a safety net as the design evolves
- Try the design and test ideas early
- Integrate early



Test-Drive Spy's Initial Condition

LightControllerSpyTest.cpp

```
TEST(LightControllerSpy, state_unknown_after_LightController_Create)
{
    LONGS_EQUAL(NO_LIGHT_ID, LightControllerSpy_GetLastId());
    LONGS_EQUAL(NO_LIGHT_STATE, LightControllerSpy_GetLastState());
}
```

This drives the spy's initialization and its accessor functions

Test-Drive the On () API

LightControllerSpyTest.cpp

```
TEST(LightControllerSpy, On_remembers_id_and_state)
{
    LightController_On(10);
    LONGS_EQUAL(10, LightControllerSpy_GetLastId());
    LONGS_EQUAL(LIGHT_ON, LightControllerSpy_GetLastState());
}
```

This drives the spy's On () API and stub implementation

Test-Drive the Off () API

LightControllerSpyTest.cpp

```
TEST(LightControllerSpy, Off_remembers_id_and_state)
{
    LightController_Off(14);
    LONGS_EQUAL(14, LightControllerSpy_GetLastId());
    LONGS_EQUAL(LIGHT_OFF, LightControllerSpy_GetLastState());
}
```

This drives the spy's Off () API and stub implementation

Light Controller Interface

LightController.h

```
#ifndef LIGHT_CONTROLLER_INCLUDED
#define LIGHT_CONTROLLER_INCLUDED

void LightController_Create();
void LightController_Destroy();
void LightController_On(int id);
void LightController_Off(int id);

#endif
```

Incrementally developed -- Test-Driven

Light Controller Spy Interface

LightControllerSpy.h

```
#ifndef LIGHT_CONTROLLER_SPY_INCLUDED
#define LIGHT_CONTROLLER_SPY_INCLUDED

#include "LightController.h"

enum
{
    NO_LIGHT_ID = -1, NO_LIGHT_STATE = -1,
    LIGHT_OFF = 0, LIGHT_ON = 1
};

int LightControllerSpy_GetLastId();
int LightControllerSpy_GetLastState();

#endif
```

Incrementally developed -- Test-Driven

Light Controller Spy Implementation

LightControllerSpy.c

```
#include "LightControllerSpy.h"

static int lastId;
static int lastState;

void LightController_Create(void)
{
    lastId = NO_LIGHT_ID;
    lastState = NO_LIGHT_STATE;
}

void LightController_Destroy()
{
}
```

Light Controller Spy Implementation

LightControllerSpy.c - part 2

```
int LightControllerSpy_GetLastId() {  
    return lastId;  
}  
  
int LightControllerSpy_GetLastState() {  
    return lastState;  
}  
  
void LightController_On(int id) {  
    lastId = id;  
    lastState = LIGHT_ON;  
}  
  
void LightController_Off(int id) {  
    lastId = id;  
    lastState = LIGHT_OFF;  
}
```

Why Does the Spy Need Tests?

- The spy is so simple, tests seem like a waste
- But, the tests document the spy
 - and they don't take much effort to create
 - and the spy may get more complicated later
 - and, it's good practice.

With the Spy Ready for the Mission, Test-Drive the Scheduler

Each initial test, provides a first use of an API and defines a boundary condition

LightScheduler - No Lights Controlled During Initialization

```
TEST(LightScheduler, no_lights_controlled_during_creation)
{
    LONGS_EQUAL(NO_LIGHT_ID, LightControllerSpy_GetLastId());
    LONGS_EQUAL(NO_LIGHT_STATE, LightControllerSpy_GetLastState());
}
```

- This test drives
 - the skeleton of the LightScheduler
 - the initialization decisions
 - What is responsible for creating the LightController?

No Lights Controlled When Nothing is Scheduled

```
TEST(LightScheduler, no_lights_controlled_when_none_scheduled)
{
    LightScheduler_Wakeup();
    LONGS_EQUAL(NO_LIGHT_ID, LightControllerSpy_GetLastId());
    LONGS_EQUAL(NO_LIGHT_STATE, LightControllerSpy_GetLastState());
}
```

- This test drives the definition of the periodic wakeup callback function.

No Lights Controlled at the Wrong Time

```
TEST(LightScheduler, no_lights_controlled_when_its_not_the_scheduled_time)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    FakeTimeService_SetDay(SUNDAY);
    FakeTimeService_SetMinute(1199);
    LightScheduler_Wakeup();
    LONGS_EQUAL(NO_LIGHT_ID, LightControllerSpy_GetLastId());
    LONGS_EQUAL(NO_LIGHT_STATE, LightControllerSpy_GetLastState());
}
```

- This test drives the schedule a turn on event

WARNING!

- Up to this point, you don't need any scheduler implementation
- Only the wiring is needed.
- The next test drives a little implementation

Schedule a light for **EVERYDAY** Turns On at the Right Time

```
TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    FakeTimeService_SetDay(SUNDAY);
    FakeTimeService_SetMinute(1200);
    LightScheduler_Wakeup();
    LONGS_EQUAL(3, LightControllerSpy_GetLastId());
    LONGS_EQUAL(LIGHT_ON, LightControllerSpy_GetLastState());
}
```

- Once this test passes, you have
 - Your design and test fixture in place
 - You have proven you can test the scheduler off the target

Test-Driven LightScheduler.h at this point

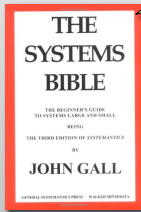
```
#ifndef D_LightScheduler_H
#define D_LightScheduler_H

void LightScheduler_Create(void);
void LightScheduler_Destroy(void);
void LightScheduler_Wakeup(void);
void LightScheduler_AddTurnOn(int id, int day, int minute);

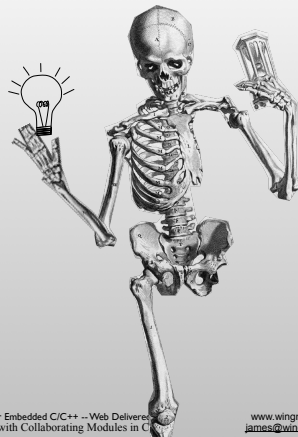
#endif
```

- It's a work in progress

A Complex system that works is invariably found to have evolved from a simple system that worked.



The Walking Skeleton Let's Try the Design and Test Ideas Early



Test-Driven LightScheduler.c at this point

```
#include "LightScheduler.h"
#include "LightController.h"
#include "TimeService.h"

typedef struct {
    int minute;
} ScheduledLightEvent;

static ScheduledLightEvent event;

void LightScheduler_Create(void)
{
    event.minute = -1;
}

void LightScheduler_Destroy(void)
{
}
```

Test-Driven LightScheduler.c at this point

```
void LightScheduler_AddTurnOn(int id, int day, int minute)
{
    event.minute = minute;
}

void LightScheduler_Wakeup(void)
{
    if (TimeService_GetMinute() != event.minute)
        return;

    LightController_On(3);
}
```

Clean Your Tests

Refer to these slides during the Cleanup Tests exercise

Don't
Repeat
Yourself

- Is this about code duplication?
- It's about idea representation duplication!

C LightScheduler TEST Before Refactoring

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    FakeTimeService_SetDay(SUNDAY);
    FakeTimeService_SetMinute(1199);
    LightScheduler_Wakeup();
    LONGS_EQUAL(NO_LIGHT_ID, LightControllerSpy_GetLastId());
    LONGS_EQUAL(NO_LIGHT_STATE, LightControllerSpy_GetLastState());
}

TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    FakeTimeService_SetDay(SUNDAY);
    FakeTimeService_SetMinute(1200);
    LightScheduler_Wakeup();
    LONGS_EQUAL(3, LightControllerSpy_GetLastId());
    LONGS_EQUAL(LIGHT_ON, LightControllerSpy_GetLastState());
}
```

Each test has to be inspected carefully to understand it.

Commonality is there though differences not obvious at a glance.

No Duplication, but not DRY

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    ScheduleLightOnThenSetTimeAndCheckLightState(3, EVERYDAY, 1200,
    SUNDAY, 1199,
    NO_LIGHT_ID, NO_LIGHT_STATE);
}

TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    ScheduleLightOnThenSetTimeAndCheckLightState(3, EVERYDAY, 1200,
    SUNDAY, 1200,
    3, LIGHT_ON);
}
```

Tests should follow the Four-Phase Test Pattern

Gerard Meszaros - xUnit Testing Patterns [XUNIT]

Setup - Establish the preconditions.
Exercise - Do something to the system.
Verify - Check the outcome.
Cleanup - Restore the system to its initial state.

a.k.a.

- Arrange, Act, Assert
- Given, When, Then

Descriptive And Meaningful Phrases

C LightScheduler TEST Refactored -- Copy/Paste/Think --

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    SetDayMinuteAndWakeUp(SUNDAY, 1199);
    CHECK_LIGHT_ID_STATE(NO_LIGHT_ID, NO_LIGHT_STATE);
}

TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    SetDayMinuteAndWakeUp(SUNDAY, 1200);
    CHECK_LIGHT_ID_STATE(3, LIGHT_ON);
}
```

Less duplication. Better
but needs work.

Extracted names tell the truth,
but not so helpful to the reader.

C LightScheduler TEST Refactored -- Copy/Paste/Think --

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    TransitionClockTo(SUNDAY, 1199);
    LIGHTS_ARE_UNCHANGED;
}

TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    LightScheduler_AddTurnOn(3, EVERYDAY, 1200);
    TransitionClockTo(SUNDAY, 1200);
    THEN_LIGHT_IS_ON(3);
}
```

Each test reads like a
story.

Many of the details have
been extracted to one
place, making changes
easier.

Extracting CHECKs in CppUTest Using Macros

```
#define LIGHTS_ARE_UNCHANGED\  
LONGS_EQUAL(NO_LIGHT_ID, LightControllerSpy_GetLastId());\  
LONGS_EQUAL(NO_LIGHT_STATE, LightControllerSpy_GetLastState());  
  
#define THEN_LIGHT_IS_ON(id)\  
LONGS_EQUAL(id, LightControllerSpy_GetLastId());\  
LONGS_EQUAL(LIGHT_ON, LightControllerSpy_GetLastState());
```

A failure in a macro will expand where used, and cause the test failure to report the line of the failed check.

A failure in a helper function will point to the line of the failed check in the helper function. — Not as helpful —

Extracting Setup Helpers in the TEST_GROUP

```
TEST_GROUP(LightScheduler)  
{  
    void setup()  
    {  
        LightController_Create();  
        LightScheduler_Create();  
    }  
    void teardown()  
    {  
        LightScheduler_Destroy();  
    }  
    void TransitionClockTo(int day, int minute)  
    {  
        FakeTimeService_SetDay(day);  
        FakeTimeService_SetMinute(minute);  
        LightScheduler_Wakeup();  
    }  
};
```

TransitionClockTo() does not generate any failures, so use a helper function rather than a macro.

Keep Tests Clean and Expressive

- Organize tests around common setup data and functions.
- if (Duplication between TEST cases)
 - Create helper routines in the TEST_GROUP.
- if (Duplication between TEST_GROUPS)
 - Create a helper classes that can be used for multiple TEST_GROUPS.
- Make the differences between tests obvious.

Example (real) Bad Test - Ramble On

```
TEST(CircularBuffer, test_it)
{
    CHECK(CircularBuffer_IsEmpty(buffer));
    CHECK_FALSE(CircularBuffer_IsFull(buffer));
    CircularBuffer_Put(buffer, 42);
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(42, CircularBuffer_Get(buffer));
    CircularBuffer_Put(buffer, 42);
    CircularBuffer_Put(buffer, 43);
    CircularBuffer_Put(buffer, 44);
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(42, CircularBuffer_Get(buffer));
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(43, CircularBuffer_Get(buffer));
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(44, CircularBuffer_Get(buffer));
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
    for (int i=0; i < CircularBuffer_Capacity(buffer); i++)
        CircularBuffer_Put(buffer, i);
    CHECK_TRUE(CircularBuffer_IsFull(buffer));
    for (int i=0; i < CircularBuffer_Capacity(buffer); i++)
        LONGS_EQUAL(i, CircularBuffer_Get(buffer));
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
    for (int i=0; i < CircularBuffer_Capacity(buffer); i++)
        CircularBuffer_Put(buffer, i);
    CHECK_TRUE(CircularBuffer_IsFull(buffer));
    LONGS_EQUAL(0, CircularBuffer_Get(buffer));
    CircularBuffer_Put(buffer, 999);
    for (int i=1; i < CircularBuffer_Capacity(buffer); i++)
        LONGS_EQUAL(i, CircularBuffer_Get(buffer));
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(999, CircularBuffer_Get(buffer));
}
```

Refactor Tests and Code as You Go

Test-case Uses

- Get the code working the 1st time
- Keep it working during changes
 - regression test safety net
- Documentation
 - Shows what the code does, unambiguously
- Examples
 - Ditto
- Helps see interface problems early
 - The test is the first user of the test case
- Helps see design problems
 - More on this later

TDD and Collaborating Objects in C++

Let's Make One Thin Slice of Functionality

Guiding Test

```
TEST(LightScheduler, lights_turned_on_at_the_right_time_scheduled_everyday)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    fakeTime.setDay(Time::MONDAY);
    fakeTime.setMinute(1200);
    lightScheduler.wakeup();
    LONGS_EQUAL(LightControllerSpy::ON, lightSpy.getLastState())
    LONGS_EQUAL(13, lightSpy.getLastId())
}
```

- I'd like to start with this test, but this is a big step!
- We need the scheduler interface, we need a schedule table, we need a spy light controller, we need ...
- It's time to procrastinate!
- Pick an easy place to start.

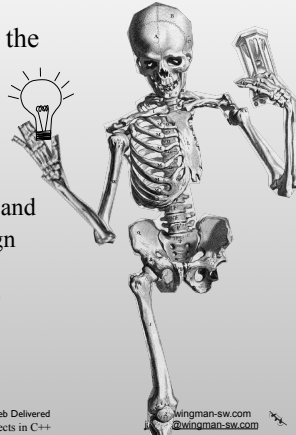
First Test-Drive the Spy



Test-Drive the Spy
keep it simple, we can add more to it later

Then a Walking Skeleton - try the design and test ideas early -

- Add one test for each call into the Light Scheduler
- These initial tests
 - are easy to make pass,
 - define a boundary condition,
 - provide a first use of key APIs, and
 - provide a safety net as the design evolves
- Try the design and test ideas early
- Integrate early



Test-Drive Spy's Initial Condition

LightControllerSpyTest.cpp

```
TEST(LightControllerSpy, light_state_unknown_after_create)
{
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastState())
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastId())
}
```

This drives the spy's initialization and its accessor functions

Test-Drive the On () API

LightControllerSpyTest.cpp

```
TEST(LightControllerSpy, light_state_on)
{
    light.on(3);
    LONGS_EQUAL(LightControllerSpy::ON, lightSpy.getLastState())
    LONGS_EQUAL(3, lightSpy.getLastId())
}
```

This drives the spy's On () API and stub implementation

Test-Drive the `Off()` API

LightControllerSpyTest.cpp

```
TEST(LightControllerSpy, light_state_off)
{
    light.off(7);
    LONGS_EQUAL(LightControllerSpy::OFF, lightSpy.getLastState())
    LONGS_EQUAL(7, lightSpy.getLastId())
}
```

This drives the spy's `Off()` API and stub implementation

Light Controller Interface

LightController.h

```
class LightController
{
public:
    explicit LightController();
    virtual ~LightController();

    virtual void on(int id) = 0;
    virtual void off(int id) = 0;

private:
    LightController(const LightController&);
    LightController& operator=(const LightController&);
};
```

Incrementally developed -- Test-Driven

Light Controller Spy Interface

LightControllerSpy.h - Part 1

```
class LightControllerSpy : public LightController
{
public:
    enum { NONE=-1, OFF, ON};

    explicit LightControllerSpy()
        : lastId(NONE), lastState(NONE)
    {}
    virtual ~LightControllerSpy()
    {}
}
```

Incrementally developed -- Test-Driven

Light Controller Spy Interface

LightControllerSpy.h - Part 2

```
virtual void on(int id)
{
    lastId = id;
    lastState = ON;
}
virtual void off(int id)
{
    lastId = id;
    lastState = OFF;
}

int GetLastState() { return lastState; }
int GetLastId() { return lastId; }
```

Incrementally developed -- Test-Driven

Light Controller Spy Interface

LightControllerSpy.h - Part 3

```
private:
    int lastId;
    int lastState;

    LightControllerSpy(const LightControllerSpy&);
    LightControllerSpy& operator=(const LightControllerSpy&);
};
```

Incrementally developed -- Test-Driven

Why Does the Spy Need Tests?

- The spy is so simple, tests seem like a waste
- But, the tests document the spy
 - and they don't take much effort to create
 - and the spy may get more complicated later
 - and, it's good practice.

With the Spy Ready for the Mission, Test-Drive the Scheduler

Each initial test, provides a first use of an interface and specifies a boundary condition

LightScheduler - No Lights Controlled During Initialization

```
TEST(LightScheduler, no_lights_controlled_during_initilization)
{
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastState())
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastId())
}
```

- This test drives
 - the skeleton of the LightScheduler
 - the initialization decisions
 - What is responsible for creating the LightController?

No Lights Controlled When Nothing is Scheduled

```
TEST(LightScheduler, no_lights_controlled_when_none_scheduled)
{
    fakeTime.setDay(Time::MONDAY);
    fakeTime.setMinute(100);
    lightScheduler.wakeup();
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastState())
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastId())
}
```

- This test drives the definition of the periodic wakeup callback function.

No Lights Controlled at the Wrong Time

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    fakeTime.setDay(Time::MONDAY);
    fakeTime.setMinute(1199);
    lightScheduler.wakeup();
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastState())
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastId())
}
```

- This test drives the AddTurnOn API

WARNING!

- Up to this point, you don't need any code in the scheduler functions added to keep the compiler and linker happy.
- Only the wiring is needed (function declarations and empty definitions)
- The next test drives a little implementation

Schedule a light for **EVERYDAY**, it Turns On at the Right Time

```
TEST(LightScheduler, lights_turned_on_at_the_right_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    fakeTime.setDay(Time::MONDAY);
    fakeTime.setMinute(1200);
    lightScheduler.wakeup();
    LONGS_EQUAL(LightControllerSpy::ON, lightSpy.getLastState())
    LONGS_EQUAL(13, lightSpy.getLastId())
}
```

- Once this test passes, you have
 - Your design and test fixture in place
 - You have proven you can test the scheduler off the target

Test-Driven LightScheduler.h at this point

```
#ifndef LIGHT_SCHEDULER_INCLUDED
#define LIGHT_SCHEDULER_INCLUDED

class TimeService;
class LightController;

class LightScheduler
{
public:
    explicit LightScheduler(TimeService&, LightController&);
    virtual ~LightScheduler();

    void wakeup();
    void addTurnOn(int lightId, int day, int minute);

private:
    TimeService& timeService;
    LightController& lightController;

    LightScheduler(const LightScheduler&);
    LightScheduler& operator=(const LightScheduler&);
};
#endif
```

- It's a work in progress

LightSchedulerTest Pre-Modern C++

```
TEST_GROUP(LightScheduler)
{
    FakeTimeService fakeTime;
    LightControllerSpy lightSpy;
    LightScheduler* lightScheduler;

    void setup()
    {
        lightScheduler = new LightScheduler(fakeTime, lightSpy);
    }

    void teardown()
    {
        delete lightScheduler;
    }
};
```

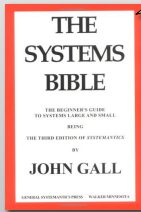
LightSchedulerTest Modern C++

```
TEST_GROUP(LightScheduler)
{
    FakeTimeService fakeTime;
    LightControllerSpy lightSpy;
    LightScheduler lightScheduler{fakeTime, lightSpy};

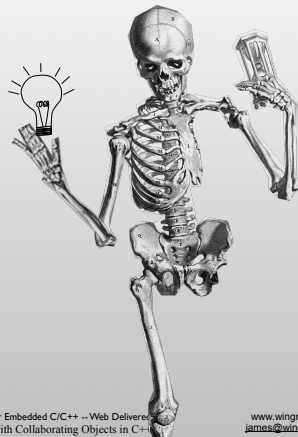
    void setup()
    {
    }

    void teardown()
    {
    }
};
```

A Complex system that works is invariably found to have evolved from a simple system that worked.



The Walking Skeleton Let's Try the Design and Test Ideas Early



Test-Driven LightScheduler.c at this point

```
#include "LightScheduler.h"
#include "LightController.h"
#include "TimeService.h"

struct ScheduledEvent
{
    int minute;
};

ScheduledEvent event;

LightScheduler::LightScheduler(TimeService& ts, LightController& lc)
: timeService(ts), lightController(lc)
{
    event.minute = -1;
}

LightScheduler::~LightScheduler()
{
}
```

Test-Driven LightScheduler.c at this point

```
void LightScheduler::wakeup()
{
    Time t = timeService.getTime();
    if (event.minute == t.getMinute())
        lightController.on(13);
}

void LightScheduler::addTurnOn(int lightId, int day, int minute)
{
    event.minute = minute;
    usedSlots++;
}
```

Clean Your Tests

Refer to these slides during exercise Part-3 Cleanup Tests

Don't
Repeat
Yourself

- Is this about code duplication?
- It's about idea representation duplication!

C LightScheduler TEST Before Refactoring

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    fakeTime.setDay(Time::MONDAY);
    fakeTime.setMinute(1199);
    lightScheduler.wakeup();
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastState())
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastId())
}

//this test requires the wiring
TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    fakeTime.setDay(Time::MONDAY);
    fakeTime.setMinute(1200);
    lightScheduler.wakeup();
    LONGS_EQUAL(LightControllerSpy::ON, lightSpy.getLastState())
    LONGS_EQUAL(13, lightSpy.getLastId())
}
```

Each test has to be inspected carefully to understand it.

Commonality is there but not obvious at a glance.

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    ScheduleLightOnThenSetTimeAndCheckLightState(3, Time::EVERYDAY, 1200,
    Time::SUNDAY, 1199,
    LightControllerSpy::NONE, LightControllerSpy::NONE);
}

TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    ScheduleLightOnThenSetTimeAndCheckLightState(3, Time::EVERYDAY, 1200,
    Time::SUNDAY, 1199,
    3, LightControllerSpy::ON);
}
```

Tests should follow the Four-Phase Test Pattern

Gerard Meszaros - xUnit Testing Patterns [XUNIT]

Setup - Establish the preconditions.
Exercise - Do something to the system.
Verify - Check the outcome.
Cleanup - Restore the system to its initial state.

a.k.a.

- Arrange, Act, Assert
- Given, When, Then

Descriptive And Meaningful Phrases

C LightScheduler TEST Refactored -- Copy/Paste/Think --

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    SetDayMinuteAndWakeUp(Time::MONDAY, 1199);
    CHECK_LIGHT_ID_STATE(LightControllerSpy::NONE, LightControllerSpy::NONE);
}

TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    SetDayMinuteAndWakeUp(Time::MONDAY, 1200);
    CHECK_LIGHT_ID_STATE(13, LightControllerSpy::ON);
}
```

Each test reads like a story.

C LightScheduler TEST Refactored -- Copy/Paste/Think --

```
TEST(LightScheduler, no_lights_controlled_at_the_wrong_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    TransitionClockTo(Time::MONDAY, 1199);
    THEN_NO_LIGHTS_CHANGED;
}

TEST(LightScheduler, light_turns_on_at_the_scheduled_minute)
{
    lightScheduler.addTurnOn(13, Time::EVERYDAY, 1200);
    TransitionClockTo(Time::MONDAY, 1200);
    THEN_LIGHT_IS_ON(13);
}
```

Each test reads like a story.

All the wiring details have been extracted to one place, making changes easier.

Extracting CHECKS in CppUTest Using Macros

```
#define LIGHTS_ARE_UNCHANGED\  
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastState())\  
    LONGS_EQUAL(LightControllerSpy::NONE, lightSpy.getLastId())  
  
#define THEN_LIGHT_IS_ON(id)\  
    LONGS_EQUAL(LightControllerSpy::ON, lightSpy.getLastState())\  
    LONGS_EQUAL(id, lightSpy.getLastId())
```

A failure in a macro will expand where used, and cause the test failure to report the line of the failed check.

A failure in a helper function will point to the line of the failed check in the helper function. — Not so helpful —

Extracting Setup Helpers in the TEST_GROUP

```
TEST_GROUP(LightScheduler)  
{  
    void setup()  
    {  
        //...  
    }  
    void teardown()  
    {  
        //...  
    }  
    void TransitionClockTo(int day, int minute)  
    {  
        fakeTime.setDay(Time::MONDAY);  
        fakeTime.setMinute(minute);  
        lightScheduler.wakeup();  
    }  
};
```

TransitionClockTo() does not generate any failures, so use a helper function rather than a macro.

Keep Tests Clean and Expressive

- Organize tests around common setup data and functions.
- if (Duplication between TEST cases)
 - Create helper routines in the TEST_GROUP.
- if (Duplication between TEST_GROUPS)
 - Create a helper classes that can be used for multiple TEST_GROUPS.
- Make the differences between tests obvious.

Example (real) Bad Test - Ramble On

```
TEST(CircularBuffer, test_it)
{
    CHECK(CircularBuffer_IsEmpty(buffer));
    CHECK_FALSE(CircularBuffer_IsFull(buffer));
    CircularBuffer_Put(buffer, 42);
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(42, CircularBuffer_Get(buffer));
    CircularBuffer_Put(buffer, 42);
    CircularBuffer_Put(buffer, 43);
    CircularBuffer_Put(buffer, 44);
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(42, CircularBuffer_Get(buffer));
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(43, CircularBuffer_Get(buffer));
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(44, CircularBuffer_Get(buffer));
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
    for (int i = 0; i < CircularBuffer_Capacity(buffer); i++)
        CircularBuffer_Put(buffer, i);
    CHECK_TRUE(CircularBuffer_IsFull(buffer));
    for (int i = 0; i < CircularBuffer_Capacity(buffer); i++)
        LONGS_EQUAL(i, CircularBuffer_Get(buffer));
    CHECK_TRUE(CircularBuffer_IsEmpty(buffer));
    for (int i = 0; i < CircularBuffer_Capacity(buffer); i++)
        CircularBuffer_Put(buffer, i);
    CHECK_TRUE(CircularBuffer_IsFull(buffer));
    LONGS_EQUAL(0, CircularBuffer_Get(buffer));
    CircularBuffer_Put(buffer, 999);
    for (int i = 1; i < CircularBuffer_Capacity(buffer); i++)
        LONGS_EQUAL(i, CircularBuffer_Get(buffer));
    CHECK_FALSE(CircularBuffer_IsEmpty(buffer));
    LONGS_EQUAL(999, CircularBuffer_Get(buffer));
}
```

Refactor Tests and Code as You Go

Test-case Uses

- Get the code working the 1st time
- Keep it working during changes
 - regression test safety net
- Documentation
 - Shows what the code does, unambiguously
- Examples
 - Ditto
- Helps see interface problems early
 - The test is the first user of the test case
- Helps see design problems
 - More on this later

Starting Point for TDD in C++

Write a script to generate these files,
or use CppUTest scripts.

SomeClass.h

```
#ifndef SOME_CLASS_INCLUDED
#define SOME_CLASS_INCLUDED

class SomeClass
{
public:
    explicit SomeClass();
    virtual ~SomeClass();

private:
    SomeClass(const SomeClass&);
    SomeClass& operator=(const SomeClass&);
};

#endif
```

This form helps
keep the
compiler from
being too helpful.
Why?

SomeClass.h - Modern C++

```
#ifndef SOME_CLASS_INCLUDED
#define SOME_CLASS_INCLUDED

class SomeClass
{
public:
    explicit SomeClass();
    virtual ~SomeClass();

private:
    SomeClass(const SomeClass&) = delete;
    SomeClass(SomeClass&) = delete;
    SomeClass& operator=(const SomeClass&) = delete;
};

#endif
```

This form helps
keep the
compiler from
being too helpful.
Why?

SomeClass.cpp

```
#include "SomeClass.h"

SomeClass::SomeClass()
{
}

SomeClass::~SomeClass()
{
}
```

Production code
constructor and
destructor are in
the cpp file.

CppUTest - SomeClassTest.cpp

```
#include "SomeClass.h"
#include "CppUTest/TestHarness.h"

TEST_GROUP(SomeClass)
{
    SomeClass* someClass;

    void setup()
    {
        someClass = new SomeClass();
    }

    void teardown()
    {
        delete someClass;
    }
};

TEST(SomeClass, Create)
{
    FAIL("Start here");
}
```

setup() is called
before each TEST.

teardown() is
called after each
TEST.

TEST_GROUPS and
TEST cases are
automatically
registered.

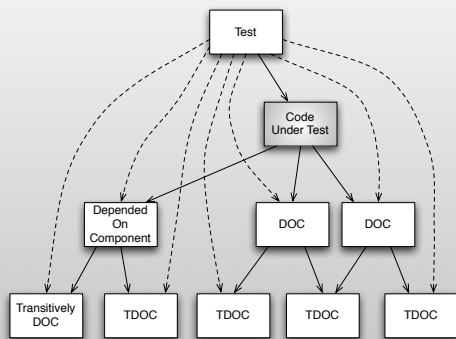


Test Doubles

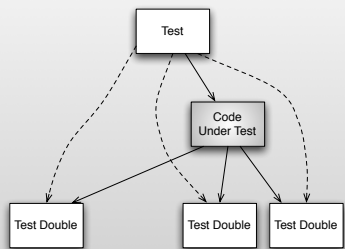
Test Double

- A test double stands in for some part of the production code during testing
- Test doubles are part of the test fixture that allows a module to be unit tested

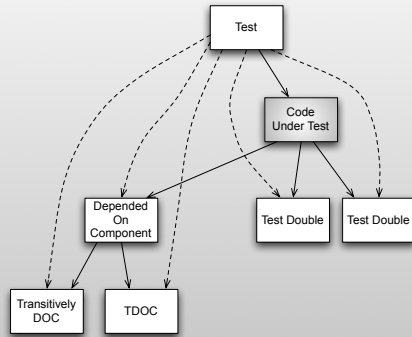
Test Dependency Octopus



Managing the Octopus



Use Production Code when you Can Test Doubles when you Must

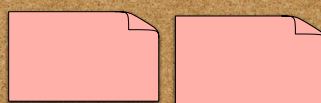
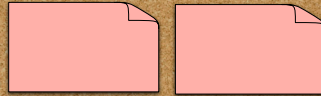
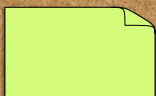


Kinds of Test Doubles [XUNIT]

- Other names
 - Fakes
 - Spy
 - Null object
 - Exploding fakes
 - Mock object
 - there are others

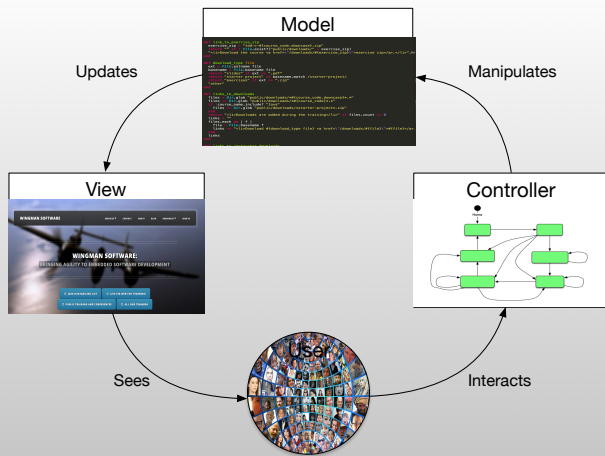
When
should you
use a test-
double?

When should
you not use a
test-double?



When Must You use Test-Doubles?

- When the code under test cannot be conveniently tested with the real collaborators, use a test double.
- Examples:
 - When manual verification is needed (Printed output, user interaction).
 - User Interface (use Model-View-Controller).
 - Database.
 - When the results change (Time, random events).
 - When failures need to be simulated (Network down).
 - Operating system calls, file system calls.
 - Special hardware interaction.

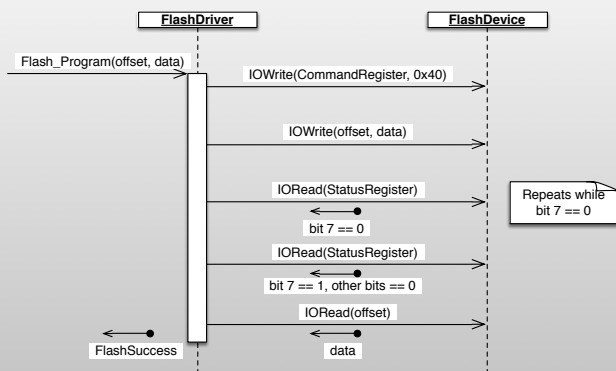


Mocking the Silicon

Mock Object

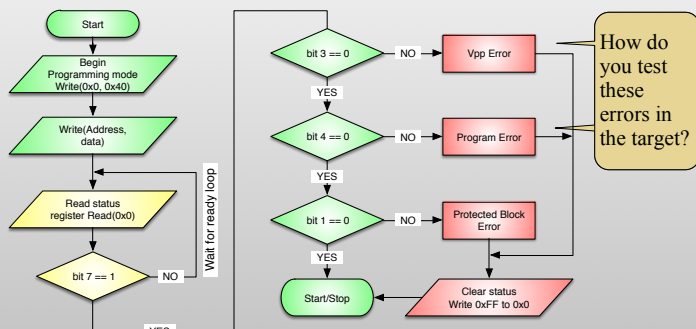
- Problem - Complex collaborator interactions cannot be captured with a simple spy.
- Solution - Mock Object
 - A Mock Object is a Test Double that verifies that the code being tested interacts with its collaborator properly.
 - The test tells the mock
 - The expected calls
 - In the expected order
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

Message Flow for Flash Program

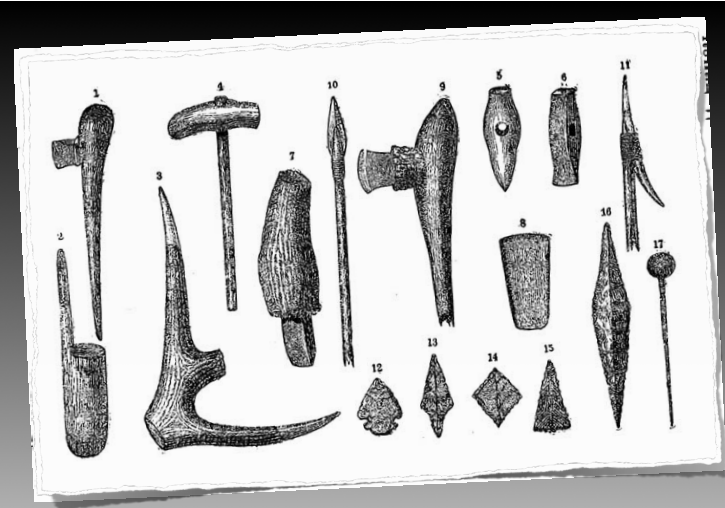
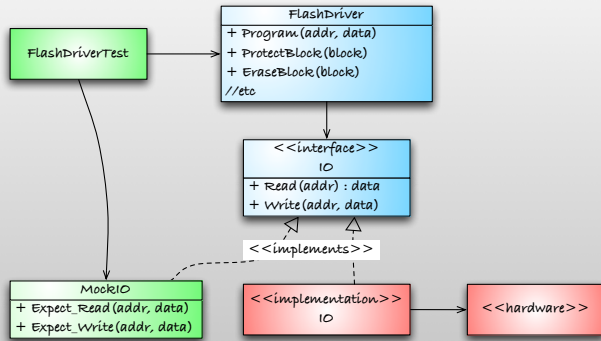


Flash Program Flow Chart

How Many Tests are Needed?



Flash Driver Test Fixture



Minimize

DoH!



Debug
On
Hardware

Copyright Fox Broadcasting. Used under fair use.

CppUMock

Mocking Support Built into CppUTest

CppUMock

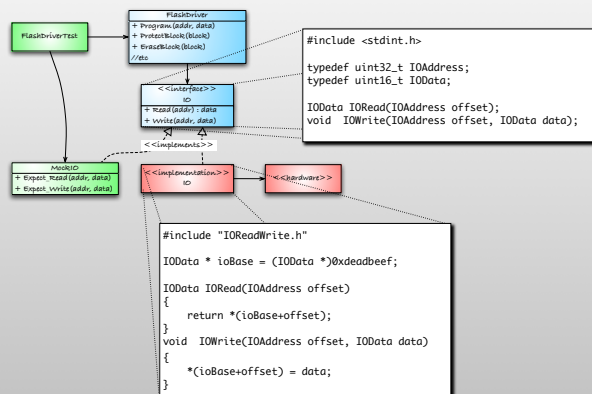
- Let's you set expectations for a series of calls on an object or function
- Let's CppUMock know when the code under test calls the mocked API
- Let's mock the IOReadWrite API

```
#include <stdint.h>

typedef uint32_t IOAddress;
typedef uint16_t IOData;

IOData IORead(IOAddress offset);
void IOWrite(IOAddress offset, IOData data);
```

Flash Driver Test Fixture



```

TEST(Flash, Program_succeeds_ready_immediately)
{
    mock("IO")
    .expectOneCall("IOWrite")
    .withParameter("addr", 0x0)
    .withParameter("data", 0x400);
    mock("IO")
    .expectOneCall("IOWrite")
    .withParameter("addr", 1000)
    .withParameter("data", 9999);
    mock("IO")
    .expectOneCall("IORead")
    .withParameter("addr", 0x0)
    .andReturnValue(1<<7);
    LONGS_EQUAL(Flash_OK, Flash_Program(1000, 9999));
}

```

```

FlashDriverTest
+ FlashDriverTest
+ Program(addr, data)
+ WriteBlock(block)
+ EraseBlock(block)
//etc

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

<<implementation>>
IO
+ Read(addr, data)
+ Write(addr, data)

<<hardware>>
IO
+ Read(addr, data)
+ Write(addr, data)

```

```

void IOWrite(IOAddress addr, IOData data)
{
    mock("IO")
    .actualCall("IOWrite")
    .withParameter("addr", (int)addr)
    .withParameter("data", (int)data);
}

IOData IORead(IOAddress addr)
{
    return mock("IO")
    .actualCall("IORead")
    .withParameter("addr", (int)addr)
    .returnValue().getIntValue();
}

```

Expectations are defined and recorded in the test case by CppUMock.

Expectations are matched by CppUMock when the actual calls to the mock version of the functions are reported via actualCall().

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Doubles

www.wingman-sw.com
james@wingman-sw.com

253

Set Expectations in Test Cases

```

TEST(MockIO, IOWrite)
{
    mock("IO")
    .expectOneCall("IOWrite")
    .withParameter("addr", 0x1000)
    .withParameter("data", 0xa000);
    IOWrite(0x1000, 0xa000);
}

TEST(MockIO, IORead)
{
    mock("IO")
    .expectOneCall("IORead")
    .withParameter("addr", 1000)
    .andReturnValue(55);
    LONGS_EQUAL(55, IORead(1000));
}

```

the mock's name

the function name being mocked

the parameter names and values passed from the code under test

IORead's return value when matched

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Doubles

www.wingman-sw.com
james@wingman-sw.com

254

The Test-Double Function Lets the mock know what happens

```

//From MockIO.CppUMock.cpp
#include "CppUTestExt/MockSupport.h"

extern "C"
{
    #include "IOReadWrite.h"
}

void IOWrite(IOAddress addr, IOData data)
{
    mock("IO")
    .actualCall("IOWrite")
    .withParameter("addr", (int)addr)
    .withParameter("data", (int)data);
}

IOData IORead(IOAddress addr)
{
    return mock("IO")
    .actualCall("IORead")
    .withParameter("addr", (int)addr)
    .returnValue().getIntValue();
}

```

the mock's name

the function name being mocked

the parameter names and values passed from the code under test

IORead's return value when matched

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Test Doubles

www.wingman-sw.com
james@wingman-sw.com

255

teardown() Tells the mock to Check Expectations

- Unmet expectations cause test failures

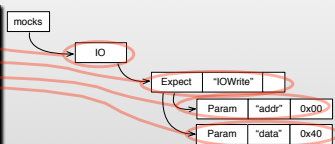
```
TEST_GROUP(MockIO)
{
    void setup()
    {
    }

    void teardown()
    {
        mock("IO").checkExpectations();
        mock("IO").clear();
        mock().clear();
    }
};
```

the mock's name

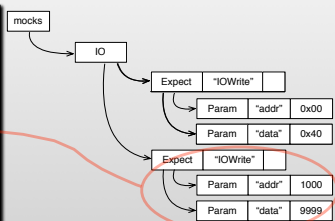
CppUMock Under the Hood

```
TEST(Flash, Program_succeeds_ready_immediately)
{
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 0x0)
        .withParameter("data", 0x40);
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 1000)
        .withParameter("data", 9999);
    mock("IO")
        .expectOneCall("IORead")
        .withParameter("addr", 0x0)
        .andReturnValue(1<<7);
    LONGS_EQUAL(Flash_OK, Flash_Program(1000, 9999));
}
```



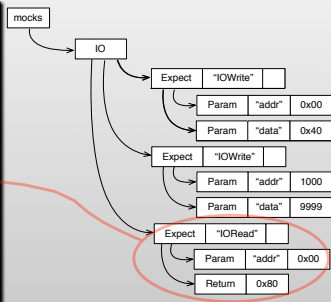
CppUMock Under the Hood

```
TEST(Flash, Program_succeeds_ready_immediately)
{
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 0x0)
        .withParameter("data", 0x40);
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 1000)
        .withParameter("data", 9999);
    mock("IO")
        .expectOneCall("IORead")
        .withParameter("addr", 0x0)
        .andReturnValue(1<<7);
    LONGS_EQUAL(Flash_OK, Flash_Program(1000, 9999));
}
```



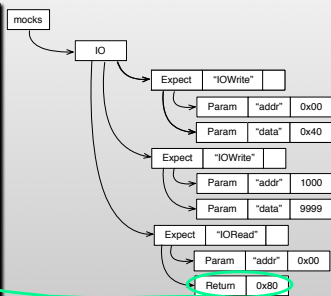
CppUMock Under the Hood

```
TEST(Flash, Program_succeeds_ready_immediately)
{
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 0x0)
        .withParameter("data", 0x40);
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 1000)
        .withParameter("data", 9999);
    mock("IO")
        .expectOneCall("IORead")
        .withParameter("addr", 0x0)
        .andReturnValue(1<<7);
    LONGS_EQUAL(Flash_OK, Flash_Program(1000, 9999));
}
```



CppUMock Under the Hood

```
TEST(Flash, Program_succeeds_ready_immediately)
{
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 0x0)
        .withParameter("data", 0x40);
    mock("IO")
        .expectOneCall("IOWrite")
        .withParameter("addr", 1000)
        .withParameter("data", 9999);
    mock("IO")
        .expectOneCall("IORead")
        .withParameter("addr", 0x0)
        .andReturnValue(1<<7);
    LONGS_EQUAL(Flash_OK, Flash_Program(1000, 9999));
}
```

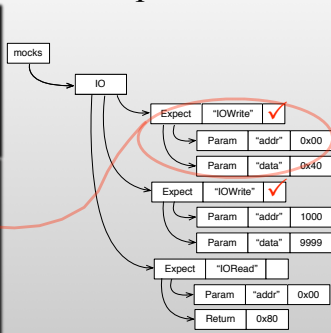


As Expectation are Met They are Recorded as Complete

```
Flash_Result Flash_Program(IOAddress addr,
IOData data)
{
    IOWrite(0, 0x40);
    IOWrite(addr, data);
    IORead(0);
    return Flash_OK;
}

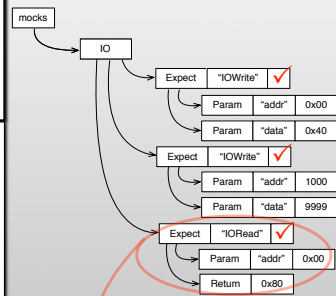
void IOWrite(IOAddress addr, IOData data)
{
    mock("IO")
        .actualCall("IOWrite")
        .withParameter("addr", (int)addr)
        .withParameter("data", (int)data);
}

IOData IORead(IOAddress addr)
{
    return mock("IO")
        .actualCall("IORead")
        .withParameter("addr", (int)addr)
        .returnValue().getIntValue();
}
```



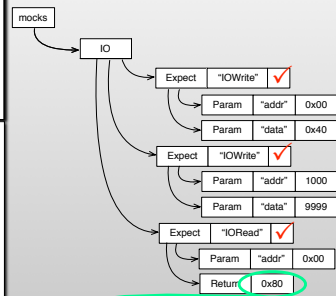
As Expectation are Met They are Recorded as Complete

```
Flash_Result Flash_Program(IOAddress addr,  
IOData data)  
{  
    IOWrite(0, 0x40);  
    IOWrite(addr, data);  
    IOWrite(0, 0x40);  
    return Flash_OK;  
}  
  
void IOWrite(IOAddress addr, IOData data)  
{  
    mock("IO")  
        .actualCall("IOWrite")  
        .withParameter("addr", (int)addr)  
        .withParameter("data", (int)data);  
}  
  
IOData IOWrite(IOAddress addr)  
{  
    return mock("IO")  
        .actualCall("IOWrite")  
        .withParameter("addr", (int)addr)  
        .returnValue().getIntValue();  
}
```

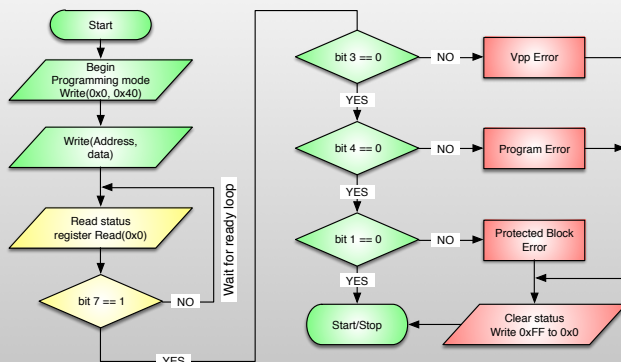


As Expectation are Met They are Recorded as Complete

```
Flash_Result Flash_Program(IOAddress addr,  
IOData data)  
{  
    IOWrite(0, 0x40);  
    IOWrite(addr, data);  
    IOWrite(0, 0x40);  
    return Flash_OK;  
}  
  
void IOWrite(IOAddress addr, IOData data)  
{  
    mock("IO")  
        .actualCall("IOWrite")  
        .withParameter("addr", (int)addr)  
        .withParameter("data", (int)data);  
}  
  
IOData IOWrite(IOAddress addr)  
{  
    return mock("IO")  
        .actualCall("IOWrite")  
        .withParameter("addr", (int)addr)  
        .returnValue().getIntValue();  
}
```



Flash Program Flow Chart



Uncle Bob's Three Laws

- I. DO NOT WRITE ANY PRODUCTION CODE UNLESS IT IS TO MAKE A FAILING UNIT TEST PASS.
- II. DO NOT WRITE ANY MORE OF A UNIT TEST THAN IS SUFFICIENT TO FAIL; AND COMPILATION FAILURES ARE FAILURES.
- III. DO NOT WRITE ANY MORE PRODUCTION CODE THAN IS SUFFICIENT TO PASS THE ONE FAILING UNIT TEST.

The Most Flexible Test-Double in C

```
//from FakeTimeService.h  
extern int (*FakeTimeService_GetMinute_custom)(void);
```

```
//from FakeTimeService.c  
int (*FakeTimeService_GetMinute_custom)(void) = 0;  
int TimeService_GetMinute(void)  
{  
    if (FakeTimeService_GetMinute_custom)  
        return FakeTimeService_GetMinute_custom();  
    return theMinute;  
}
```

I need incrementing GetMinute()

```
static int minute;  
static int My_GetMinute(void)  
{  
    return minute++;  
}  
  
TEST(TimeService, CustomGetMinute)  
{  
    minute = 0;  
    FakeTimeService_GetMinute_custom = My_GetMinute;  
    LONGS_EQUAL(0, TimeService_GetMinute());  
    LONGS_EQUAL(1, TimeService_GetMinute());  
    LONGS_EQUAL(2, TimeService_GetMinute());  
}
```

I need Mock GetMinute ()

```
static int Mock_GetMinute(void)
{
    return mock("TimeService")
        .actualCall("GetMinute")
        .returnValue().getIntValue();
}

TEST(TimeService, MockGetMinute)
{
    minute = 0;
    FakeTimeService_GetMinute_custom = Mock_GetMinute;
    LONGS_EQUAL(0, TimeService_GetMinute());
    LONGS_EQUAL(1, TimeService_GetMinute());
    LONGS_EQUAL(2, TimeService_GetMinute());
}
```

CppUMock Demo

FlashDriver-CppUMock

Exercise and Discussion

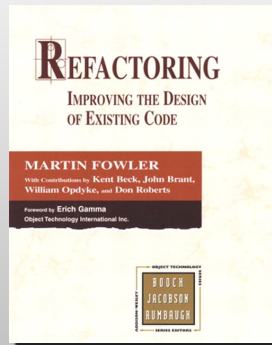
FlashDriver-CppUMock

Exercise and Discussion

- Exercise Objectives
 - Using a Mock object
 - Test-drive next to the Silicon
 - Going from data sheet to test cases
 - Deliberate TDD practice
- See Tdd-FlashDriver-CppuMock



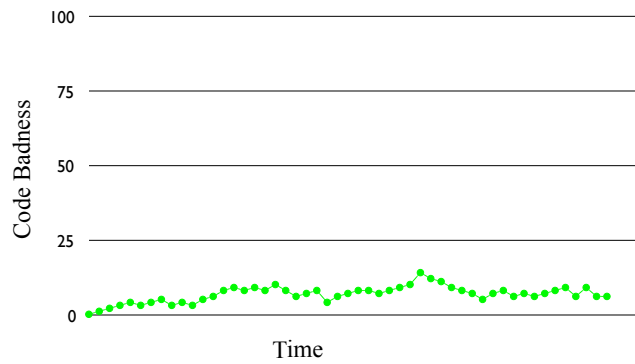
Refactoring



T
D
D

- Write a test
- Watch it not build
- Make it build, but fail
- Make it pass
- Refactor (clean up any mess)
- Repeat until done

Refactor to Sustain Low Code 'Badness'



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

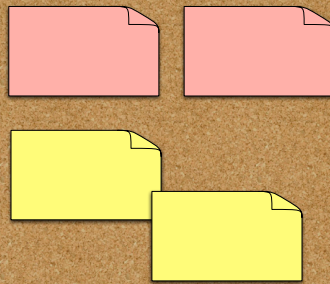
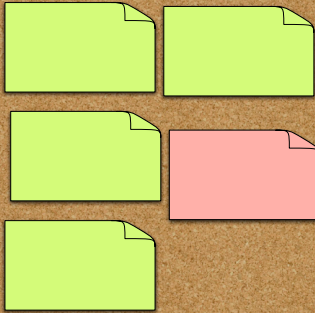
TDD for Embedded C/C++ -- Web Delivered
Refactoring

www.wingman-sw.com
james@wingman-sw.com

274

When is
Design Done?

Why is
Software
valuable?



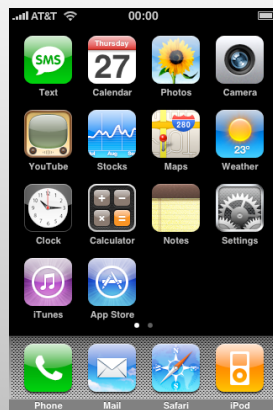
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Refactoring

www.wingman-sw.com
james@wingman-sw.com

275

The Two values of Software



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Refactoring

www.wingman-sw.com
james@wingman-sw.com

276

What is the consequence of rushing your work?

Helpful

Harmful

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Refactoring

www.wingman-sw.com
james@wingman-sw.com

277

The Financial Implications of Technical Debt

- MIT Sloan School of Management, Agile businesses have a huge “Return on Agility”
- Technical debt can bring a “smooth running agile business to a screeching halt”

<http://blog.castsoftware.com/the-financial-implications-of-technical-debt/>

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Refactoring

www.wingman-sw.com
james@wingman-sw.com

278

Rushing Slows You Down

Cause and effect
Opposite effect
Delayed effect
Delayed opposite effect

Copyright (c) 2009 James Grenning

Inspired by Scaling Lean and Agile Development [SLAD]

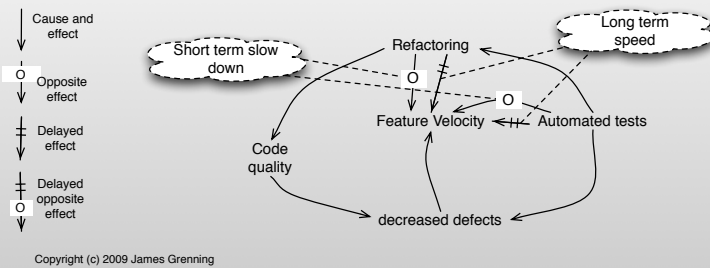
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Refactoring

www.wingman-sw.com
james@wingman-sw.com

279

Slow Down to Go Faster



Inspired by Scaling Lean and Agile Development [SLAD]

What Slows You Down?



"We don't have time for refactoring, there's still too much left to do."

Reply Retweet Favorite More

Are you too busy to improve?



Refactoring

See Martin Fowler's definitive work in:
Refactoring. Improving the Design of Existing Code. [REF]

Defined

“Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.” [REF]

- Refactoring Debt

Refactoring Goal

- “Clean code that works”
– Ron Jeffries

How?

- Remove duplication
- Fix bad names
– J.B. Rainsberger

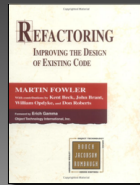
Donald Knuth Says

Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.



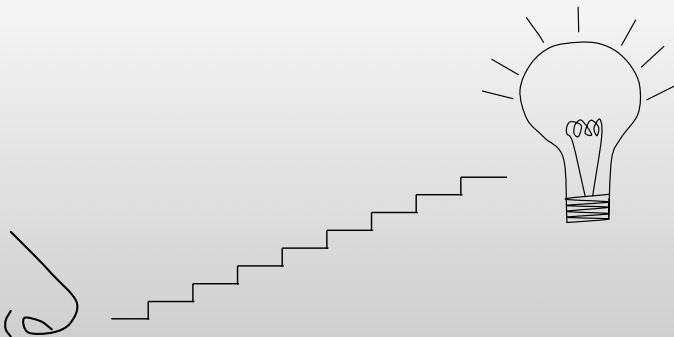
Martin Fowler Says

Any fool can write code that the compiler understands, but it takes real skill to write code other programmers can understand.



From "Refactoring - Improving the Design of Existing Code"

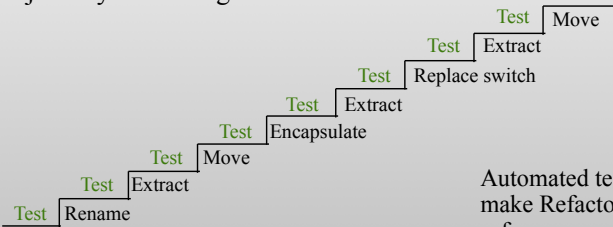
Three Critical Skills



Recognize what is wrong and fix it!

Refactorings

- Follow a course that keeps all test running as you move the design closer to the envisioned solution. Keep in mind, you are likely to see things on the journey that change the destination.



Automated tests
make Refactoring
safe.

Refactoring vs. Humpty Dumpty Redesign



Code with
design problem

Take it apart, put it back together.
The system is broken during redesign effort.

Refactor to the better design.
Test are always passing.
The system is never broken.

Envisioned
design

Broken
Debug
Rework
Unpredictable
Unhappy

Improved
design that works

Don't Burn Bridges

- Build new code, get it to compile.
- Cut over to it in an easy to undo operation.
- Check in frequently.
- Keep code running.
- Don't debug, undo to a prior working state. Then try again.



Smells Found in Object Oriented Code [REF]

- Long method
- Large class
- Feature envy
- Duplicate code
- Inappropriate intimacy
- Refused bequest
- Lazy class
- Comments
- Contrived Complexity
- Long parameter list
- Primitive obsession



What is Well Commented Code?

Ottinger's Comment Rules

- Comment Rule: Comments are for things that cannot be expressed in code.
- Comment Redundancy Rule: Comments that restate the code must be deleted.
- Comment Single Truth Rule: If the comment says what the code could say, then the code must change to make the comment redundant.

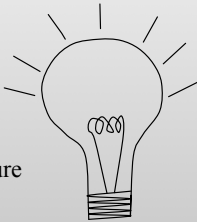
Smells Found in C Code

- Duplicate code
- Bad Pasta
- Long function
- Cryptic names
- Abstraction Distraction
- Switch case disgrace
- Bewildering Boolean
- Nefarious Nesting
- Long parameter list
- Willy-nilly initialization
- Global free-for-all
- Primitive obsession
- Comments
- Dead code



Envisioning

- Once a design problem is identified, you must envision a better solution
- Look to apply the design principles
 - Single Responsibility
 - Substitutability
 - DRY - Don't Repeat Yourself
 - from The Pragmatic Programmer
 - Ports and Adaptors — Hexagonal Architecture
 - Law of Demeter (LoD).
 - Principle of Least Knowledge.
 - Separation of Concerns (SoC).



TDD - the Code Rot Radar

- As code becomes opaque, it resists tests.
- Lack of testability provides an early warning of impending design problems.
- Keeping the code clean, before it gets too messy, keeps the cost of refactoring low.

Don't Live with Broken Windows

Andy Hunt, Dave Thomas [PRAG]

- Broken windows:
 - poor design
 - bad names
 - smelly code
 - repetitive manual processes



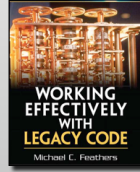
Legacy Code Refactoring

See Michael Feather's definitive work:

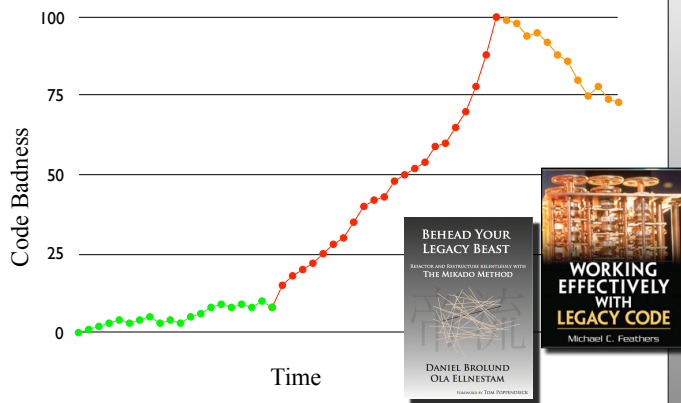
Working Effectively with Legacy Code [WELC]

LegacyCode == CodeWithoutTests

You can write legacy code today!
Although that is not recommended.



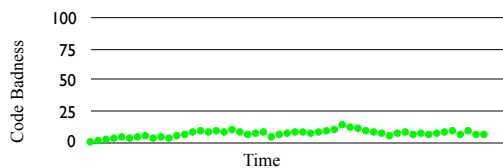
Code 'Badness' Over time



Review -- the Refactoring Mindset

- Refactoring is part of everyday engineering.
- Considered the duty a professional.
- Tests are needed to safely refactor.

Refactor to Sustain Low Code 'Badness'



Legacy Code Change Algorithm in Practice

1. Identify change points.
2. Make changes
3. Pray



Legacy Mindset

- Refactoring legacy code is very risky.
- Without tests we have to be very careful -- blindfolded is not careful!
- Legacy code behavior is the spec.
- Focus on making minor changes that enable testing.
- With tests in place, risk is lowered.



Legacy Code Change Algorithm

Revised (based on [WELC])

1. Identify change points.
2. Find test points.
3. Break dependencies.
4. Write tests.
5. Make changes
6. Refactor
7. (less prayer needed)

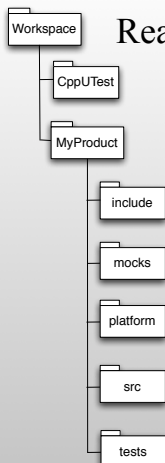
Careful, the
Fast Way is!



Boy Scout Rule

Leave the campsite cleaner than you found it.

- Adding a single line to a long function
 - makes the code worse.
 - Extract something first.
- Adding another clause to a complex if statement
 - makes the code worse.
 - Extract the whole complex conditional.
- Adding another parameter to a long parameter list
 - makes the code worse.
 - Create a new module (struct + functions).
- Adding another nesting level to a for/if/switch
 - makes the code worse.
 - Extract the inner nesting level.



Reasonable Workspace Structure

- **MyProduct** might be a whole system or a subsystem. You may have multiple products
- **include** contains header files that define public interfaces
- **mocks** contains test stubs
- **platform** contains platform specific production code
- **src** contains production code, and private header files
- **tests** contains test cases and maybe some stubs too
- There is usually a directory structure under **include**, **mocks**, **src** and **tests**, depending on your needs

Crash to Pass Algorithm

- Goal: call the function under test from the test harness
- Caution: Don't get discouraged.
- Once the first test runs and the initialization is complete, subsequent tests are created much more rapidly.
- See
 - <https://wingman-sw.com/articles/tdd-legacy-c>

```
void addNewLegacyCTest()
{
    while (makeItCompile() && makeItLink())
        ;
    while (runCrashes())
    {
        findRuntimeDependency();
        fixRuntimeDependency();
    }
    makeTestMoreMeaningful();
}
```

Get Legacy C Help

wingman-sw.com/articles/tdd-legacy-c

- You'll find
 - A detailed procedure to get your legacy code into CppUTest
 - A list of problems with solution options
- Part of the challenge is recognizing the problem at hand so you can choose a viable solution

Fix One Problem at a Time!

Rely on Cause and Effect

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Legacy Code Refactoring

www.wingman-sw.com
james@wingman-sw.com

313

No Batch Mode

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Legacy Code Refactoring

www.wingman-sw.com
james@wingman-sw.com

314



Dependencies and C Linker Errors

```
Linking HomeAutomation_tests
objs/LightScheduler.o: In function `LightScheduler_AddTurnOn':
/sandbox/LightScheduler.c:86: undefined reference to `LightController_On'
objs/LightScheduler.o: In function `LightScheduler_AddTurnOff':
/sandbox/LightScheduler.c:91: undefined reference to `LightController_Off'
lib/libHomeAutomation.a(Scheduler.o): In function `Scheduler_Create':
/sandbox/Scheduler.c:37: undefined reference to `TimeService_SchedulePeriodicAlarm'
lib/libHomeAutomation.a(Scheduler.o): In function `Scheduler_Destroy':
/sandbox/Scheduler.c:53: undefined reference to `TimeService_CancelPeriodicAlarm'
lib/libHomeAutomation.a(Scheduler.o): In function `processReadyEvents':
/sandbox/Scheduler.c:70: undefined reference to `TimeService_DaysMatch'
lib/libHomeAutomation.a(Scheduler.o): In function `Scheduler_Wakeup':
/sandbox/Scheduler.c:79: undefined reference to `TimeService_GetDay'
/sandbox/Scheduler.c:80: undefined reference to `TimeService_GetMinute'
... etc ...
collect2: error: ld returned 1 exit status
```

Generated Exploding Fakes

```
#ifndef EXPLODING_FAKE_INCLUDED
#define EXPLODING_FAKE_INCLUDED
#include "CppUTest/TestHarness.c.h"

#define EXPLODING_FAKE_FOR(f) \
    void f() { FAIL_TEXT_C("go write a proper stub for " #f); }
#endif
```

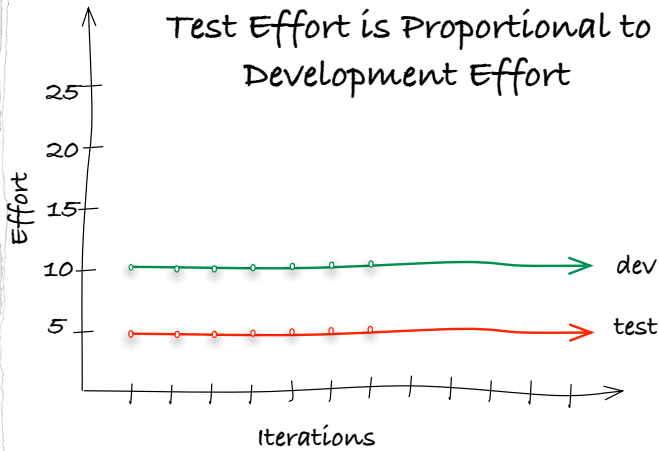
```
//Generated from linker errors
#include "explodingfake.h"

EXPLODING_FAKE_FOR(LightController_Off)
EXPLODING_FAKE_FOR(LightController_On)
EXPLODING_FAKE_FOR(TimeService_CancelPeriodicAlarm)
EXPLODING_FAKE_FOR(TimeService_DaysMatch)
EXPLODING_FAKE_FOR(TimeService_GetDay)
EXPLODING_FAKE_FOR(TimeService_GetMinute)
```



Manual Test is Not Sustainable

Illustrates the Assumption that
Test Effort is Proportional to
Development Effort



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Manual Test is Unsustainable

www.wingman-sw.com
james@wingman-sw.com

If a system is working, leave it
alone. Don't change anything.

THE SYSTEMS BIBLE

THE BEGINNER'S GUIDE
TO SYSTEMS LARGE AND SMALL
BEING
THE THIRD EDITION OF SYSTEMANTICS

BY
JOHN GALL

GENERAL SYSTEMANTICS PRESS WALKER, MINNESOTA

Systems don't
appreciate being fiddled
and diddled with.

SYSTEMANTICS

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Manual Test is Unsustainable

www.wingman-sw.com
james@wingman-sw.com

System Antics

System - something setup to attain a goal

Antics - amusing, frivolous, or eccentric
behavior

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
Manual Test is Unsustainable

www.wingman-sw.com
james@wingman-sw.com

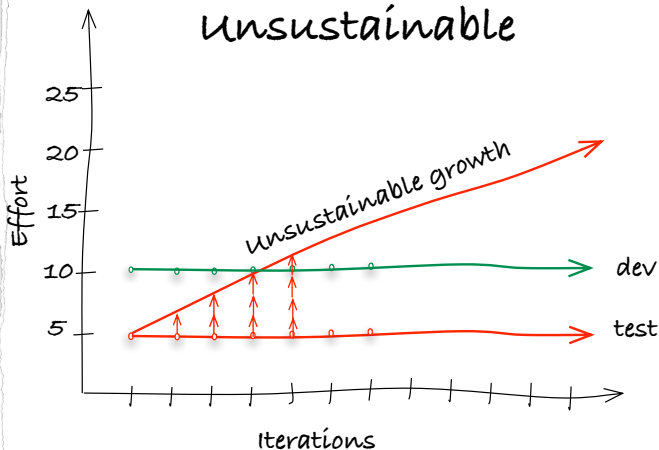
System

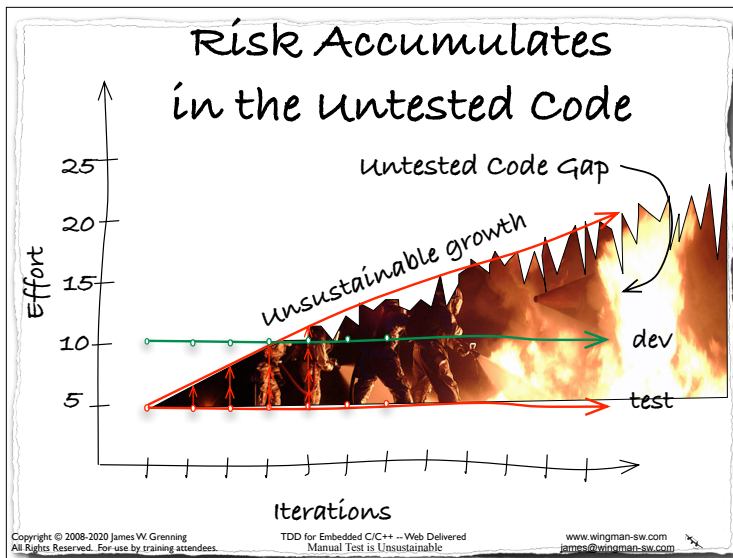
Antics



Because systems act
up -- we have to be
very careful.

Manual Test is
unsustainable





WINGMAN
SOFTWARE

Closing Thoughts Questions Empty the Parking Lot

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered

www.wingman-sw.com
james@wingman-sw.com

326

Repeat After Me

I am a programmer and I write bugs tests!

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

327

Write Two Post-it Notes

What do you like about development?

I like good code

I like coding

I like problem solving

I like doing quality work

What is your least favorite development activity?

I don't like documentation

I don't like testing. It's boring

I don't like wasting time

Chasing someone else's bugs

I don't like hacking

I don't like messy code

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

328

UI

Service

Unit

Tests are needed at all levels of the pyramid

<http://blog.mountaingoatsoftware.com/the-forgotten-layer-of-the-test-automation-pyramid>

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

329

Higher Level Tests Cannot be Thorough

Tests

```
graph TD; A[10 states] -- 5 interactions --> B[10 states]; B -- 5 interactions --> C[10 states]; C -- 5 interactions --> A;
```

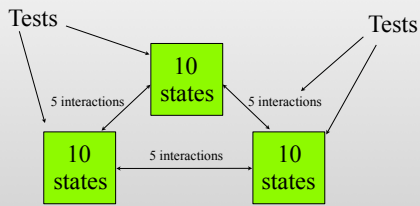
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

330

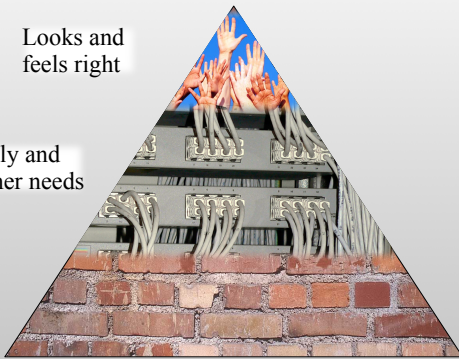
Unit Tests Can Be Thorough



Looks and
feels right

Wired properly and
meets customer needs

Solid
foundation



Excuses keeping
me from doing TDD

I don't have time!!

Learn it and TDD is faster than DLP

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

334



My Boss won't let me

**Become an expert in
your craft so you can
advise your boss**

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

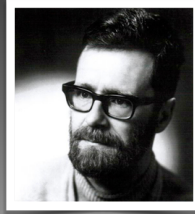
TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

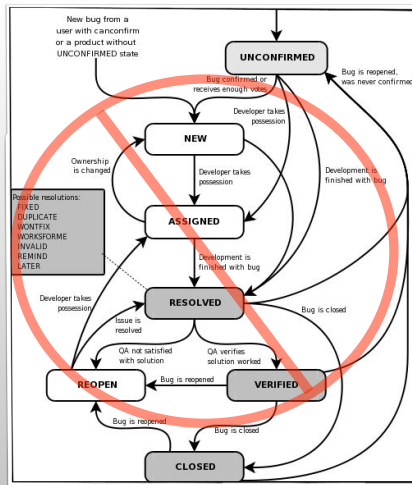
336

Edsger Dijkstra

... 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.



Prevent over Manage

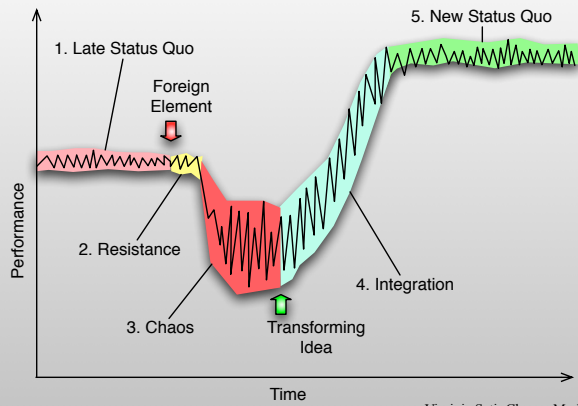


From <http://www.softwaretestinghelp.com/bug-life-cycle/>



Improvement Takes Time

Satir Change Model



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

Virginia Satir Change Model

www.wingman-sw.com
james@wingman-sw.com

340

Skills Take Time to Develop

Knowledge

Understanding

Skill

Mastery

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

341

Solving One Problem at a Time



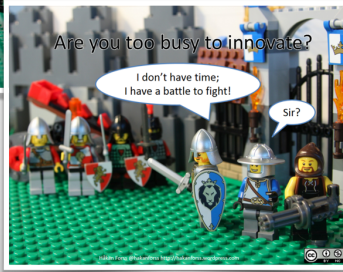
Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

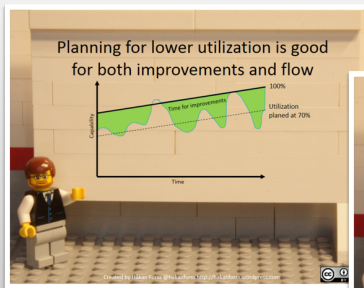
342

Are you too busy to improve?



<https://hakanforss.wordpress.com/2014/03/10/are-you-too-busy-to-improve/>

Make Time to Learn



<https://hakanforss.wordpress.com/2014/03/10/are-you-too-busy-to-improve/>

**"We are what we repeatedly do.
Excellence, then, is *not an act*,
but a habit."**

Aristotle



<https://hakanforss.wordpress.com/2014/03/10/are-you-too-busy-to-improve/>







http://commons.wikimedia.org/wiki/File:Abies_nordmanniana_PAN_2.jpg

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ --Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

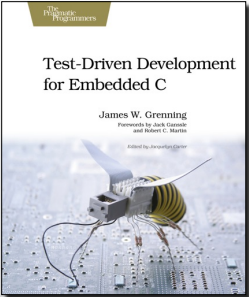
347

Talk to me on Twitter
@jwgrenning

Connect with me
www.linkedin.com/in/jwgrenning
Remind me how we met.

www.wingman-sw.com
blog.wingman-sw.com
www.jamesgrenning.com
facebook.com/wingman.sw





wingman-sw.com/tddc/



wingman-sw.com/tddc/

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ --Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

348

Have You Read These Important Works



Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

349

References

- [XP] Kent Beck, Extreme Programming Explained, 1999
- [REF] Martin Fowler. Refactoring. Improving the Design of Existing Code. 1999
- [WELC] Michael Feathers, Working Effectively with Legacy Code
- [TDD] Kent Beck, Test-Driven Development, 2003
- [XUNIT] Gerard Meszaros, xUnit Testing Patterns, 2008
- [PRAG] Andy Hunt, Dave Thomas, The Pragmatic Programmer
- [SLAD] Craig Larman and Bas Voode, Scaling Lean & Agile Development
- [POP] Mary Poppendieck and Tom Poppendieck, Implementing Lean Software Development: From Concept to Cash, 2006
- [AGILE] Robert C. Martin, Agile Software Development: Principles, Patterns, and Practices, 2002
- [CLEAN] Robert C. Martin, Clean Code, 2008
- [LESSONS] Cem Kaner, et. al. Lessons learned in Software Testing
- [TD] Lasse Koskela, Test Driven, 2007
- Test harnesses
 - [CPPTTEST] www.sourceforge.org, project CppUTest
 - [FITNESSE] www.fitnesse.org
- Groups
 - <http://groups.yahoo.com/group/testdrivendevelopment>
 - <https://groups.google.com/forum/?fromgroups#!forum/cpputest>
 - <http://groups.yahoo.com/group/AgileEmbedded>

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

350

Resources

Website	http://www.wingman-sw.com
Book	http://wingman-sw.com/tddcc/
Papers and Presentations	http://wingman-sw.com/papers
Blog	http://www.wingman-sw.com/blog/
CppUTest google group	https://groups.google.com/forum/?fromgroups#!forum/cpputest
Github code examples	https://github.com/jwrenning?tab=repositories
Agile Embedded Yahoo Group	https://groups.yahoo.com/neo/groups/AgileEmbedded/
Facebook page	www.facebook.com/wingman-sw

Copyright © 2008-2020 James W. Grenning
All Rights Reserved. For use by training attendees.

TDD for Embedded C/C++ -- Web Delivered
The End

www.wingman-sw.com
james@wingman-sw.com

351