Test-Driven Development for Embedded C - Workshop

Talk to me on Twitter
@jwgrenning

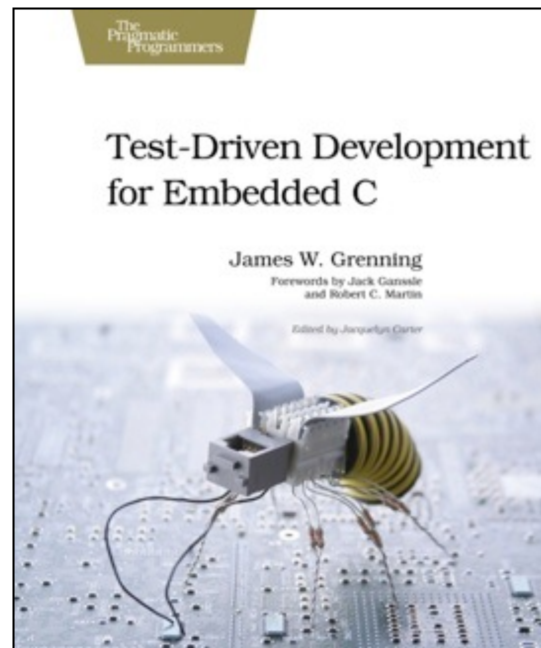
Find my book at
http://www.pragprog.com/titles/jgade

Find me on linkedin.com
http://www.linkedin.com/in/jwgrenning
Please remind me how we met.

http://www.wingman-sw.com
http://www.wingman-sw.com/blog
http:// www.jamesgrenning.com

By
James Grenning

Nov 15, 2013



Write Three Post-it Notes

What do you like about development?

What is your least favorite development activity?

What percentage of your development time is spent testing and removing bugs from your code?

I like...

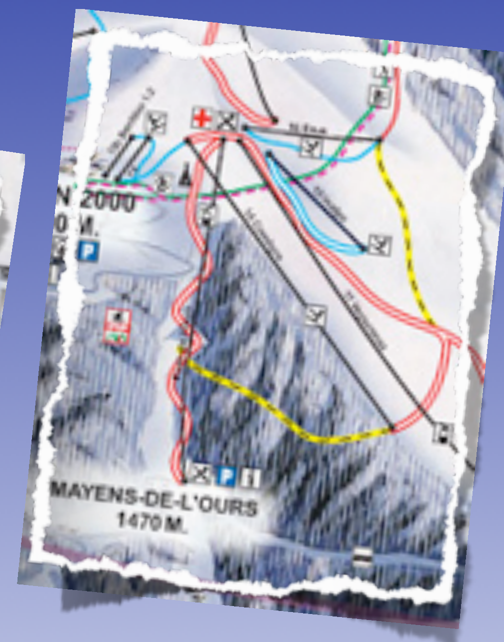
I don't like...

coding %
+ testing %
+ debugging %
= 100%

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

There is a Lot to TDD

We’ll Look at a Few Parts of TDD

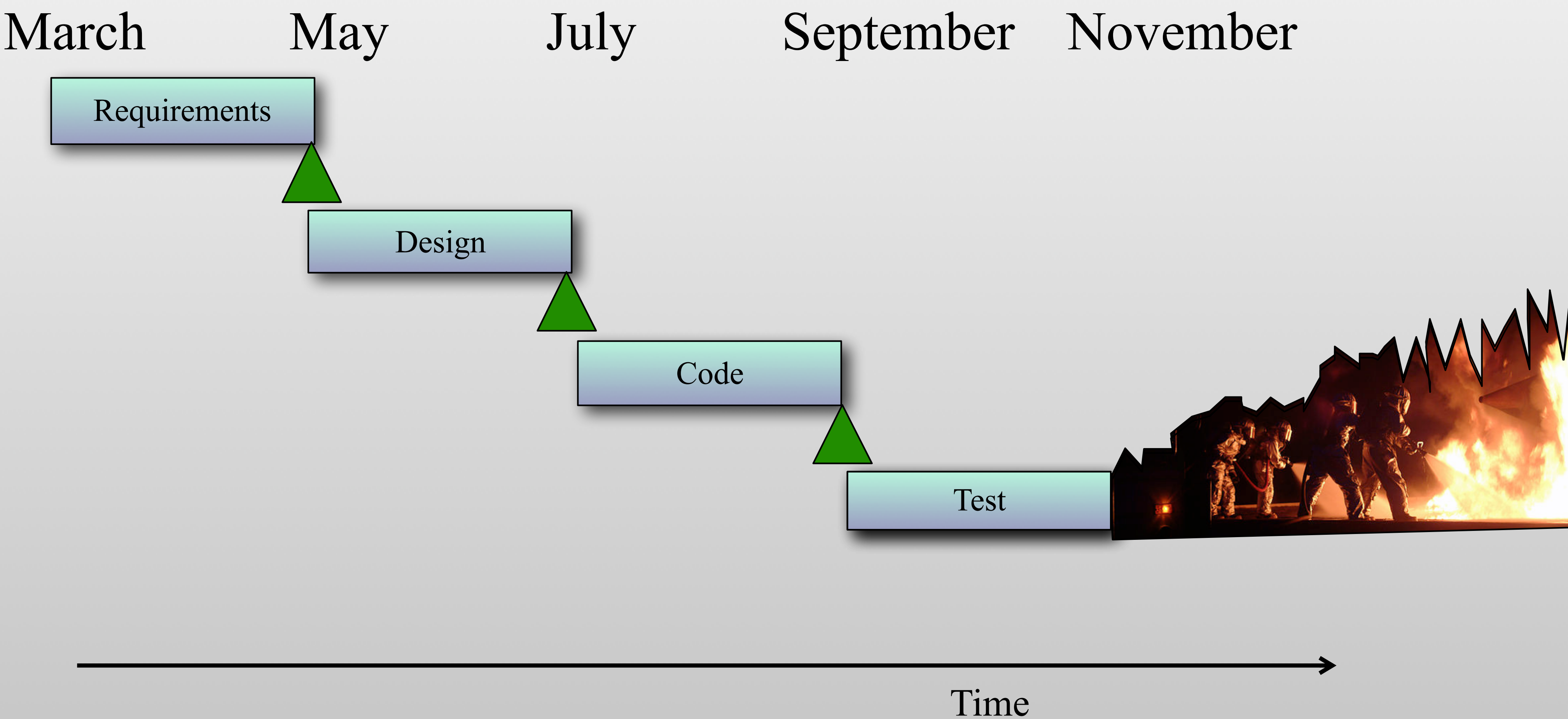


There’s more.

Questions I'll Answer

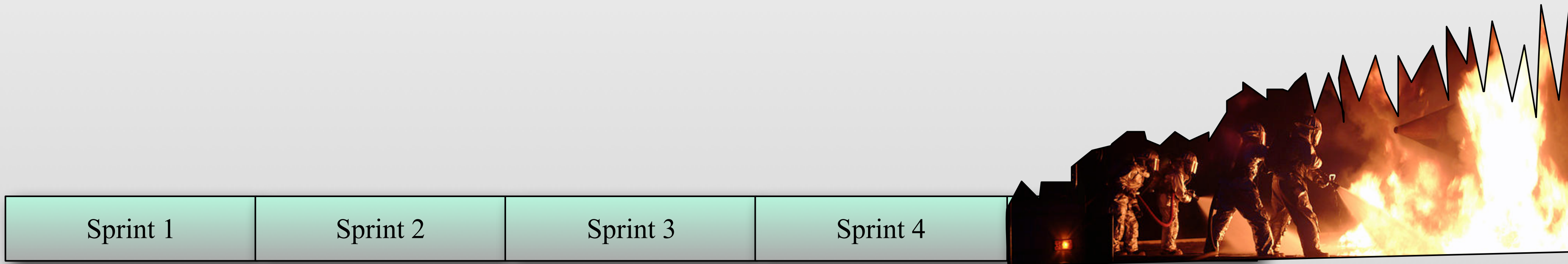
- Why should I care about TDD?
- What is TDD?
- How do I test code with dependencies?
- How is TDD adapted to embedded development?

Saving Testing Until the End

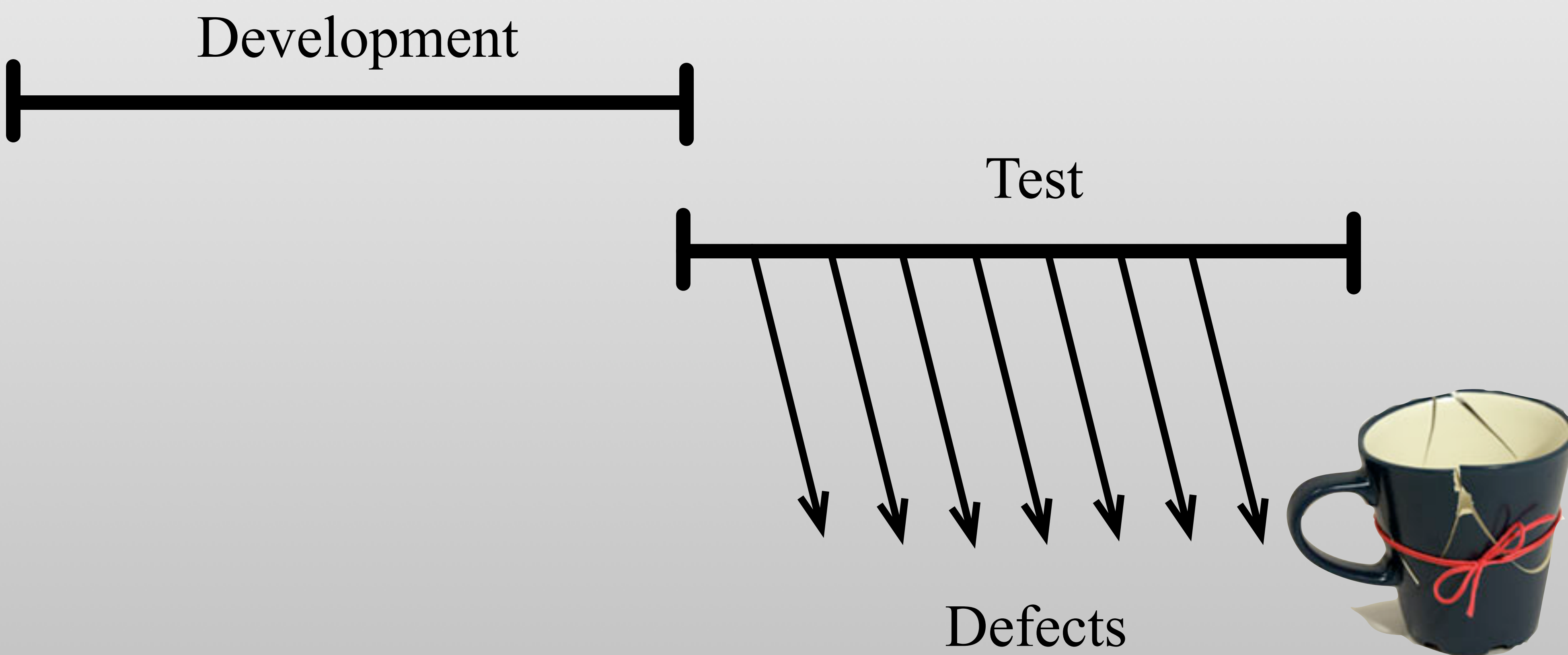


www.wingman-sw.com
james@wingman-sw.com

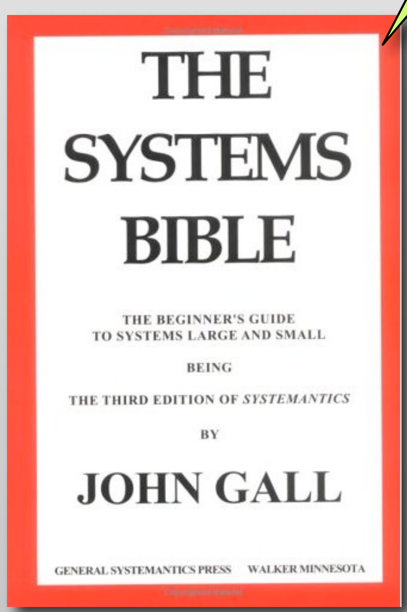
Agile without TDD
with Hardening Sprints



This Work Flow is Designed to Produce Defects

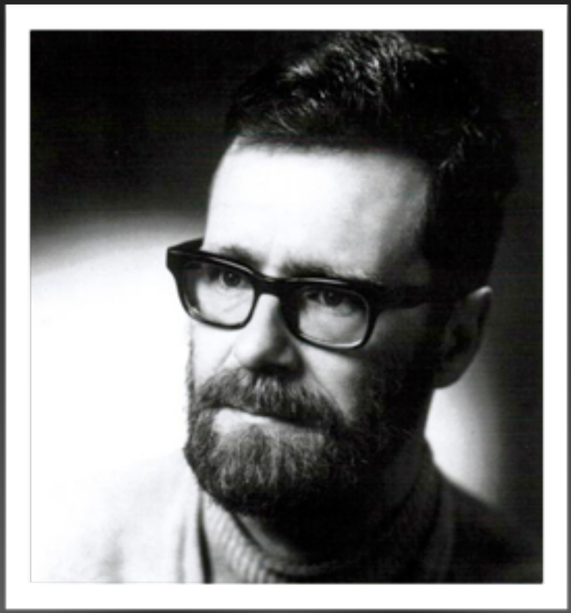


Your program will have bugs. And they will surprise you when you find them.



Edsger Dijkstra

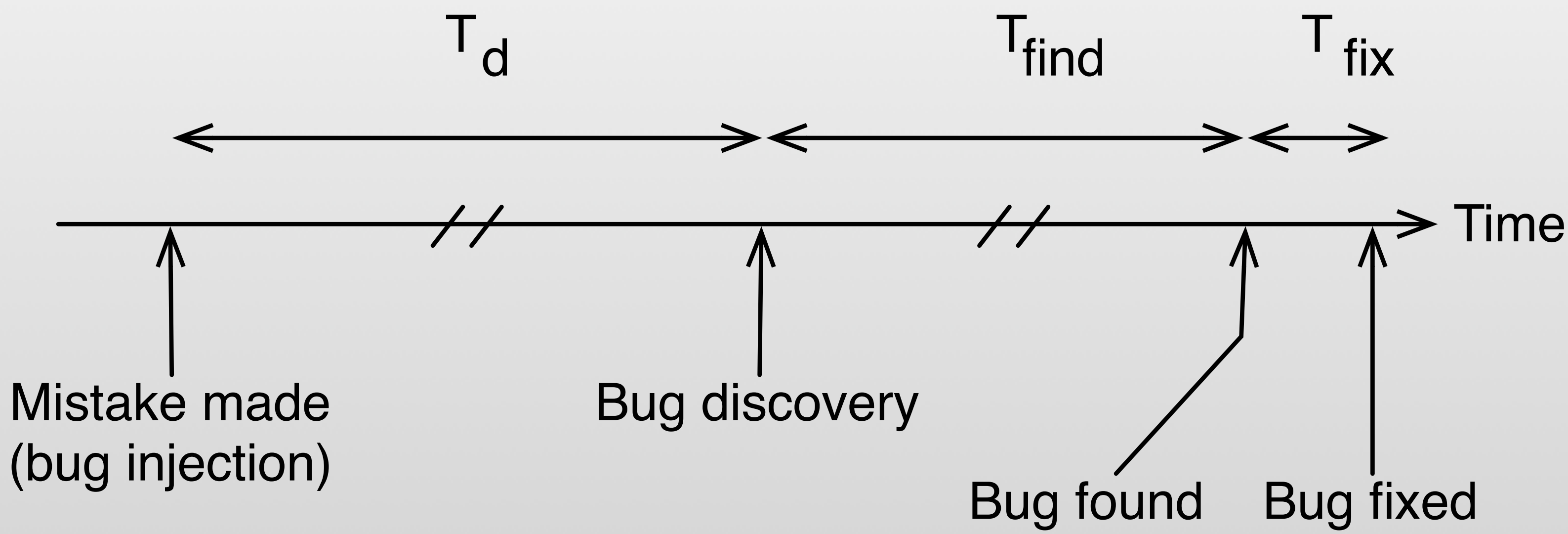
Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result, the programming process will become cheaper. If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.



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

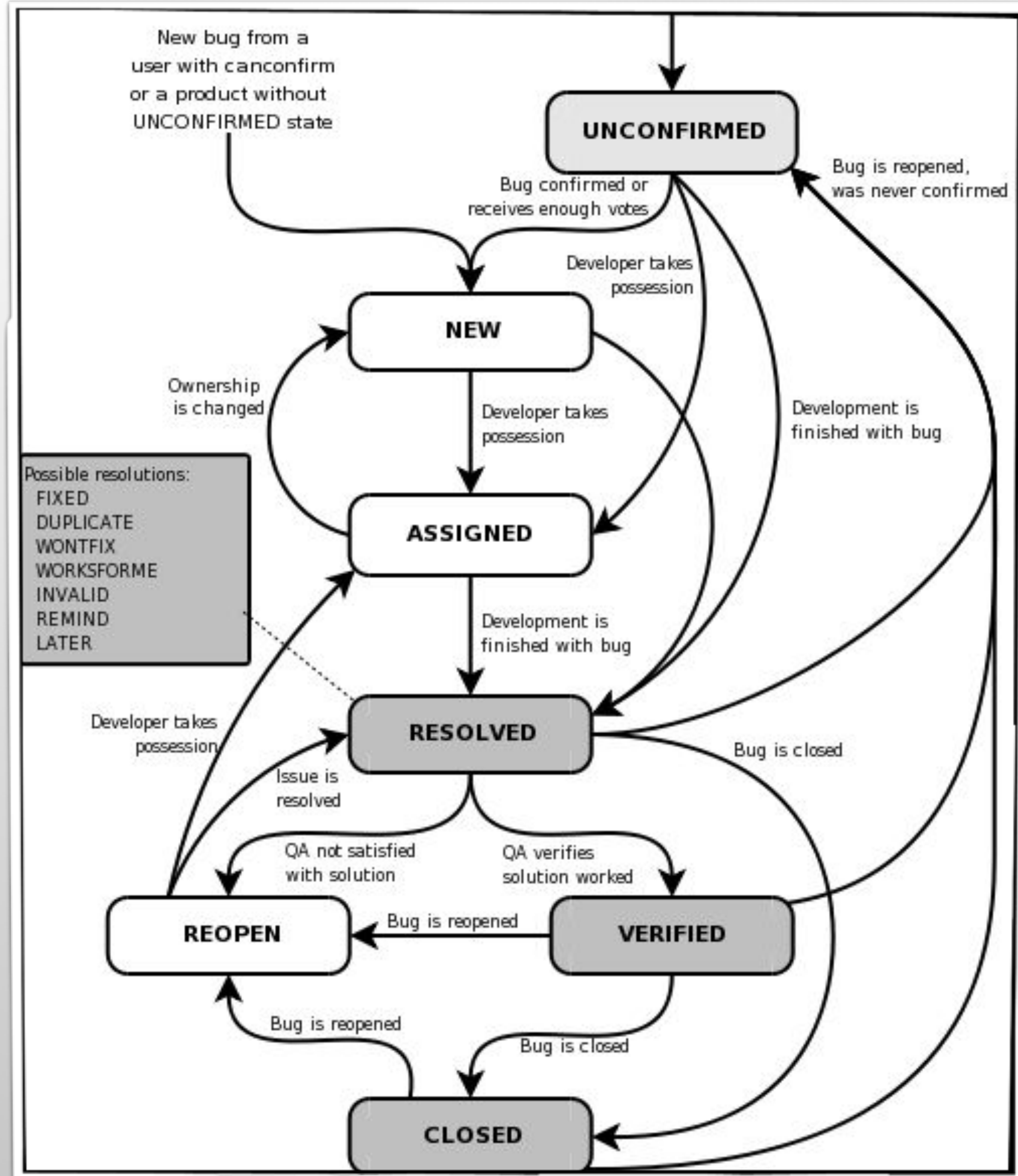
Can we Realize Dijkstra's Dream and Prevent Defects with Test Driven Development?

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

A Bug's Life



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

What is TDD?

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

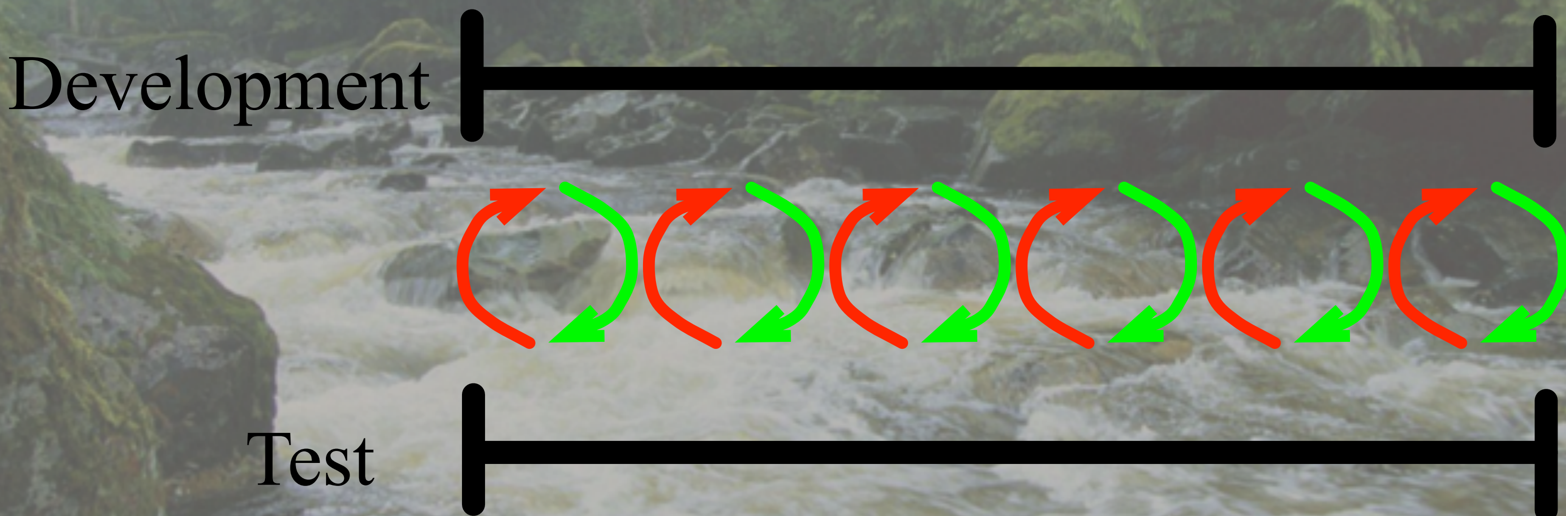
T
D
D

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

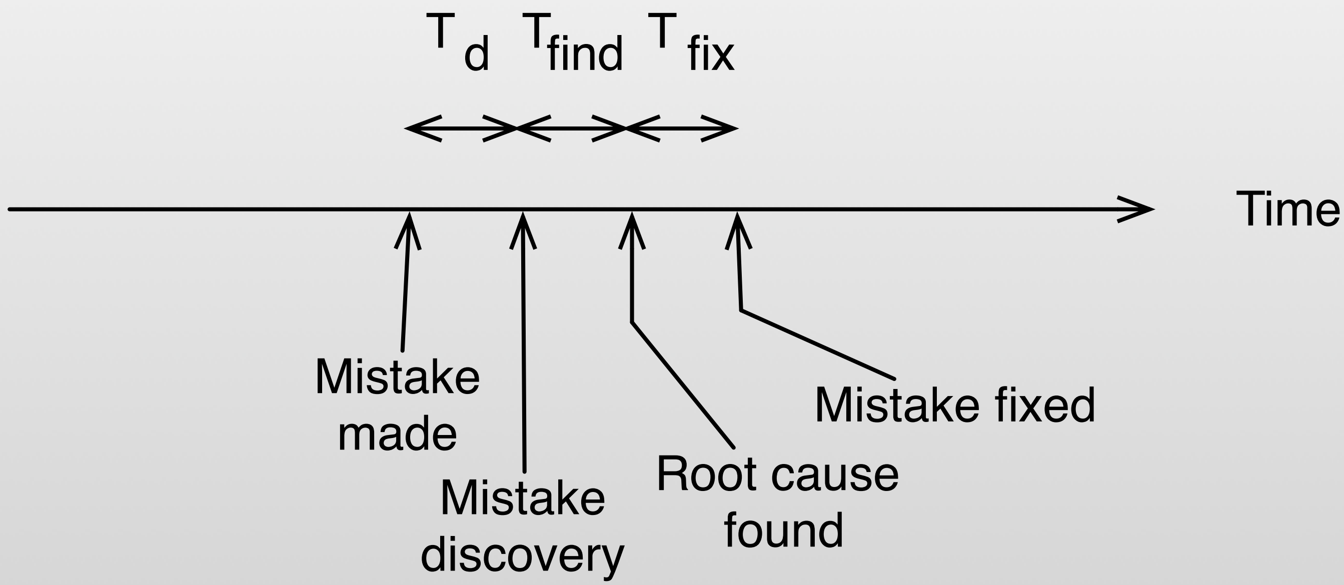
www.wingman-sw.com
james@wingman-sw.com

If test moved upstream, could we get working features to flow?

Development and Test are a Continuum preventing defects

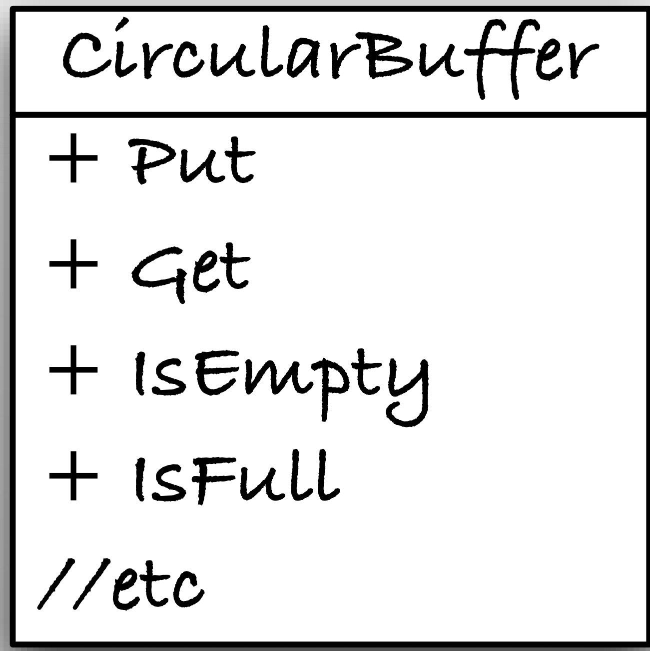


The Physics of Test Driven Development



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

Let's Test-Drive a CircularBuffer that Holds Integers

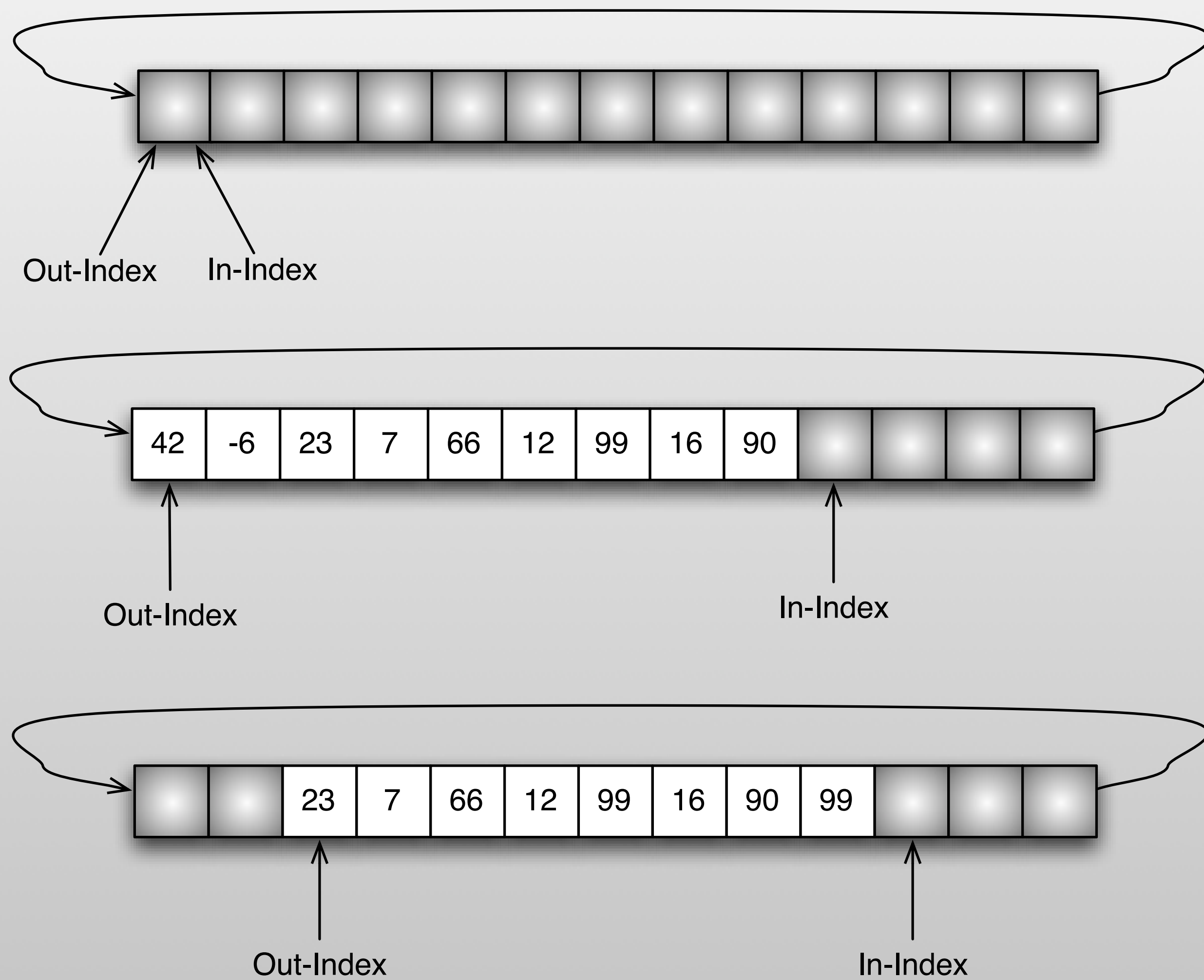


Create a Test-List
(avoid analysis paralysis)

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

Have an Idea of How it Will Work



Setup a Test Fixture

```
TEST_GROUP(CircularBuffer)
{
    CircularBuffer* buffer;

    void setup()
    {
      buffer = CircularBuffer_Create();
    }

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

TEST(CircularBuffer, TestName)
{
}
```


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

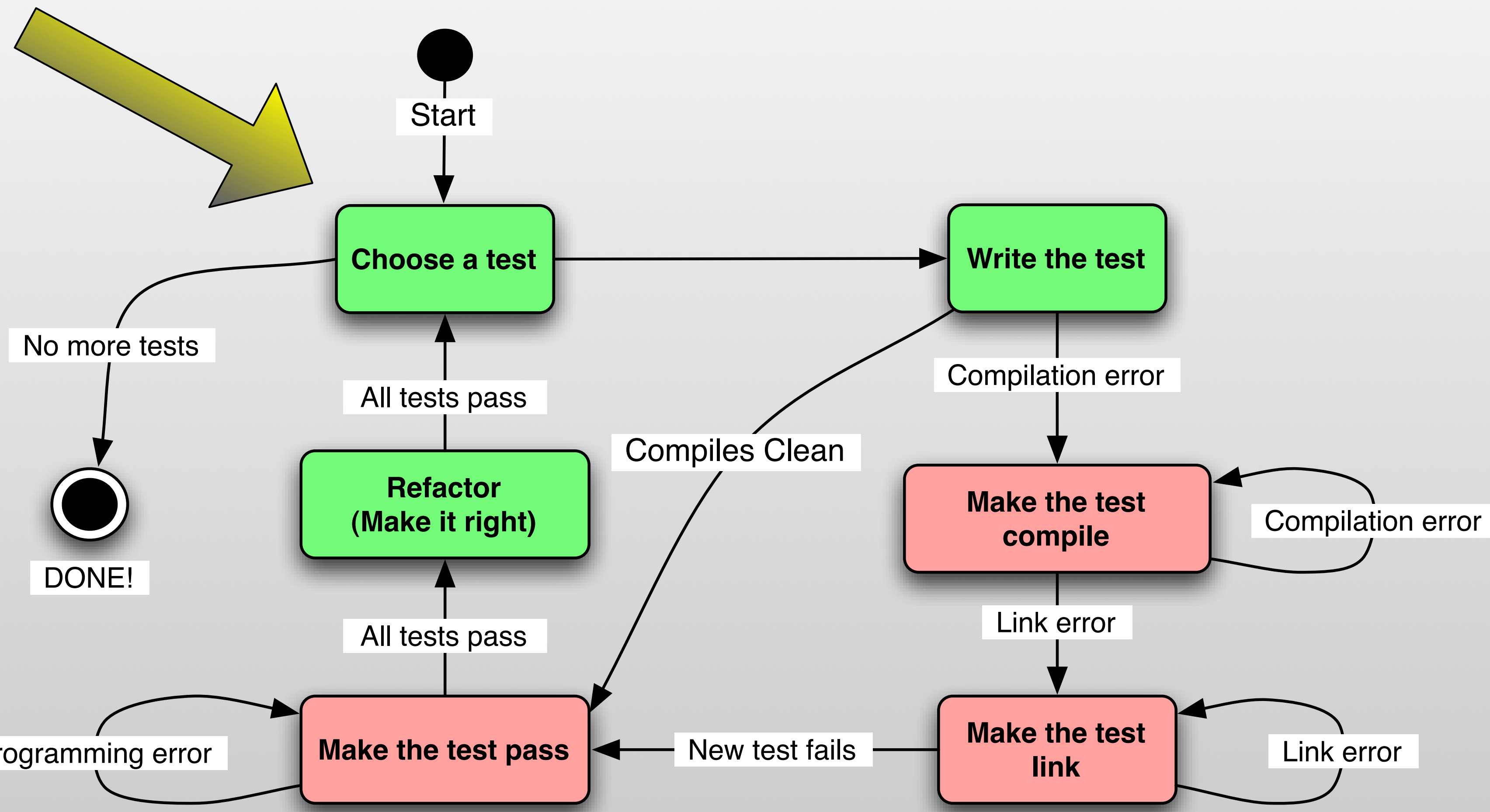
Make a Bare Skeleton of the Production Code Header File

```
#ifndef D_CircularBuffer_H
#define D_CircularBuffer_H

typedef struct CircularBuffer CircularBuffer;

CircularBuffer * CircularBuffer_Create();
void CircularBuffer_Destroy(CircularBuffer *);

#endif  // D_CircularBuffer_H
```

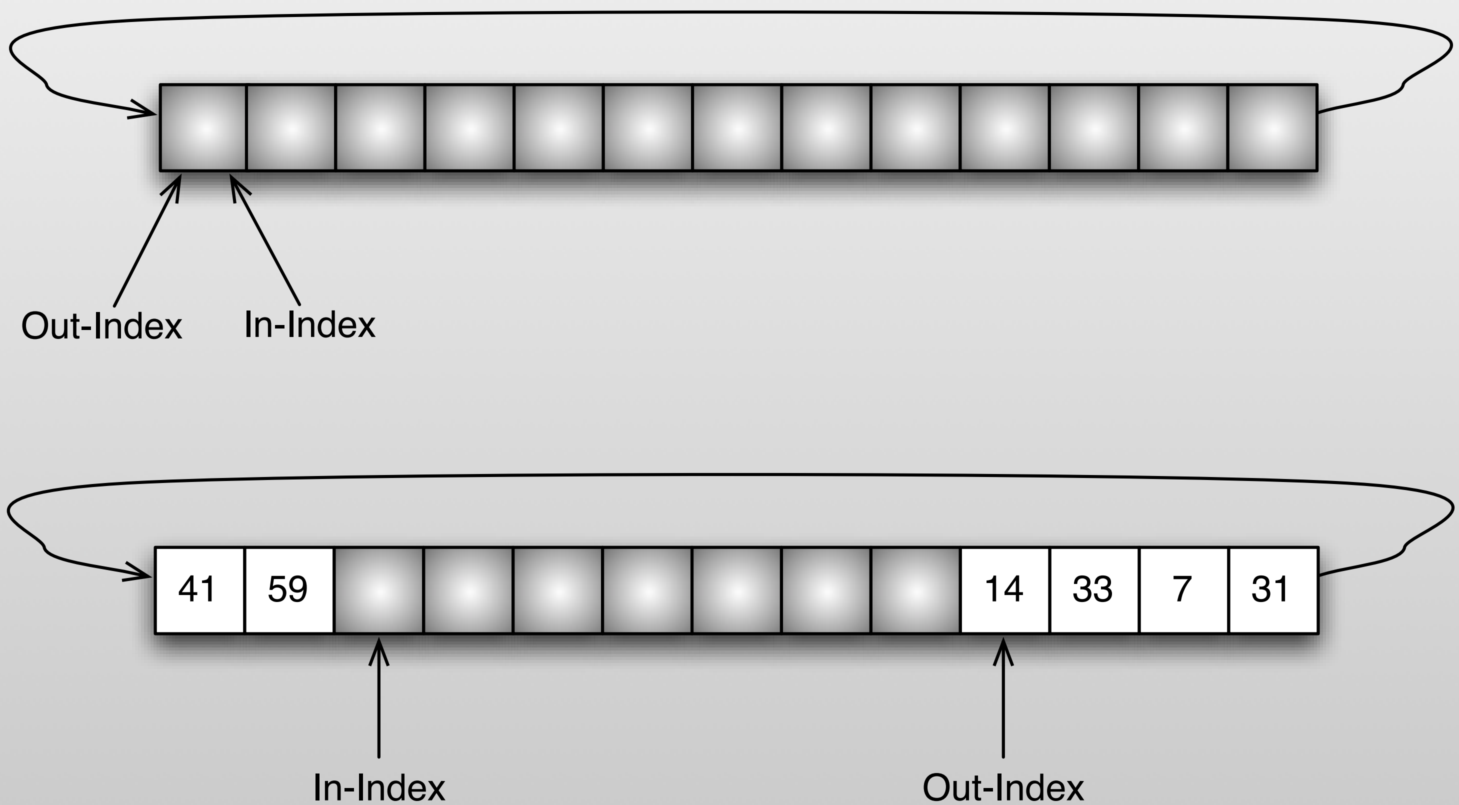
Make a Bare Skeleton of the Production Code Source File

```c
#include "CircularBuffer.h"
#include <stdlib.h>

struct CircularBuffer
{
    int dummy;
} ;

CircularBuffer* CircularBuffer_Create()
{
    CircularBuffer* self = calloc(1, sizeof(CircularBuffer));
    return self;
}

void CircularBuffer_Destroy(CircularBuffer* self)
{
    free(self);
}
```

Choose a Test



Which Test Should We Start With?



Exercise

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

Write Three Post-it Notes

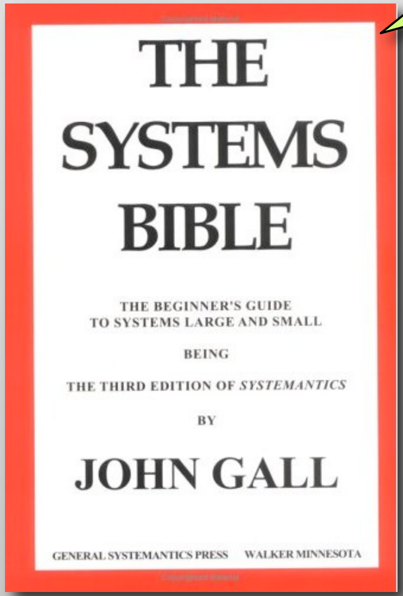
What did you like about TDD?

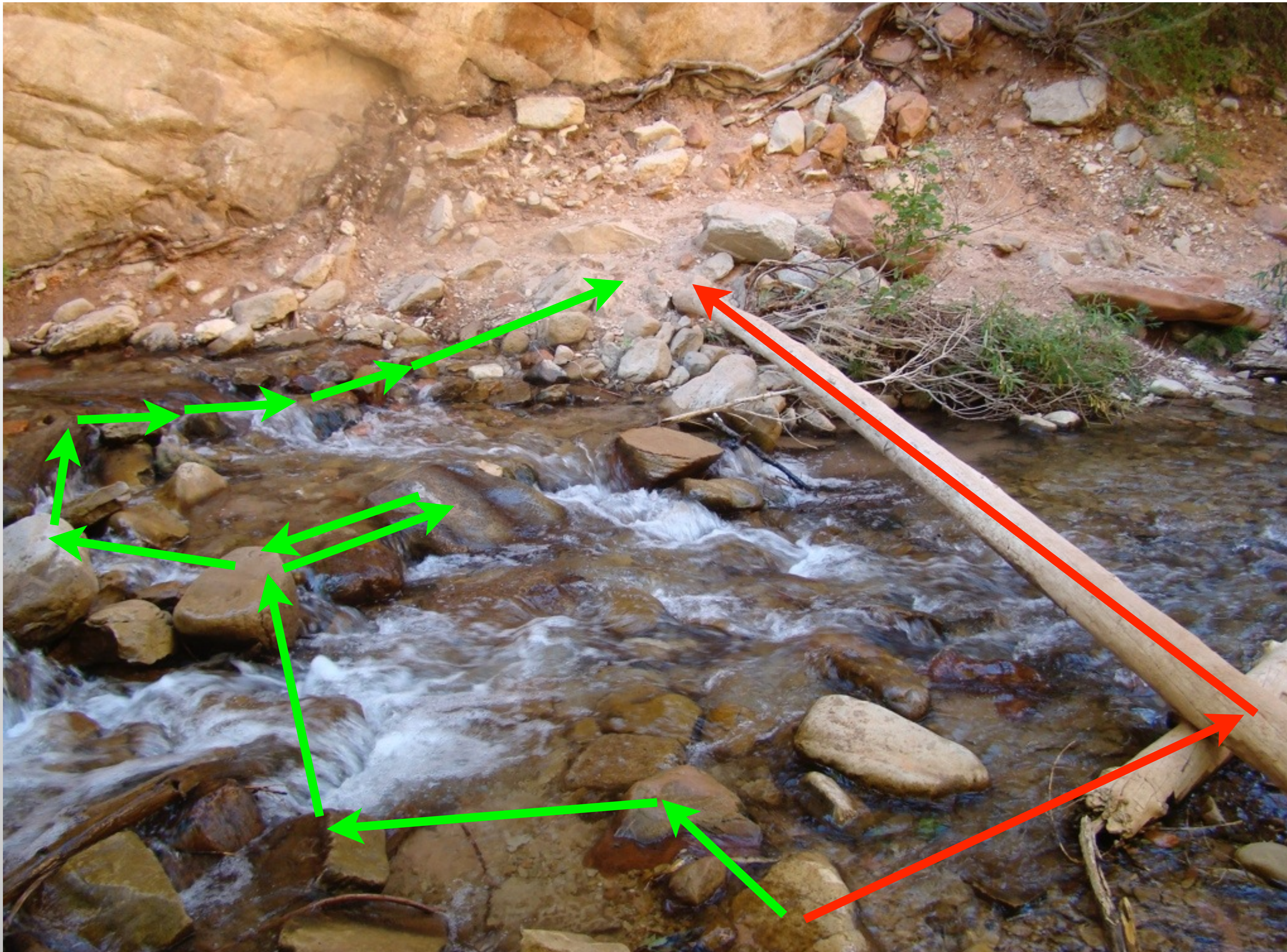
What concerns you about TDD?

Liked

Concern

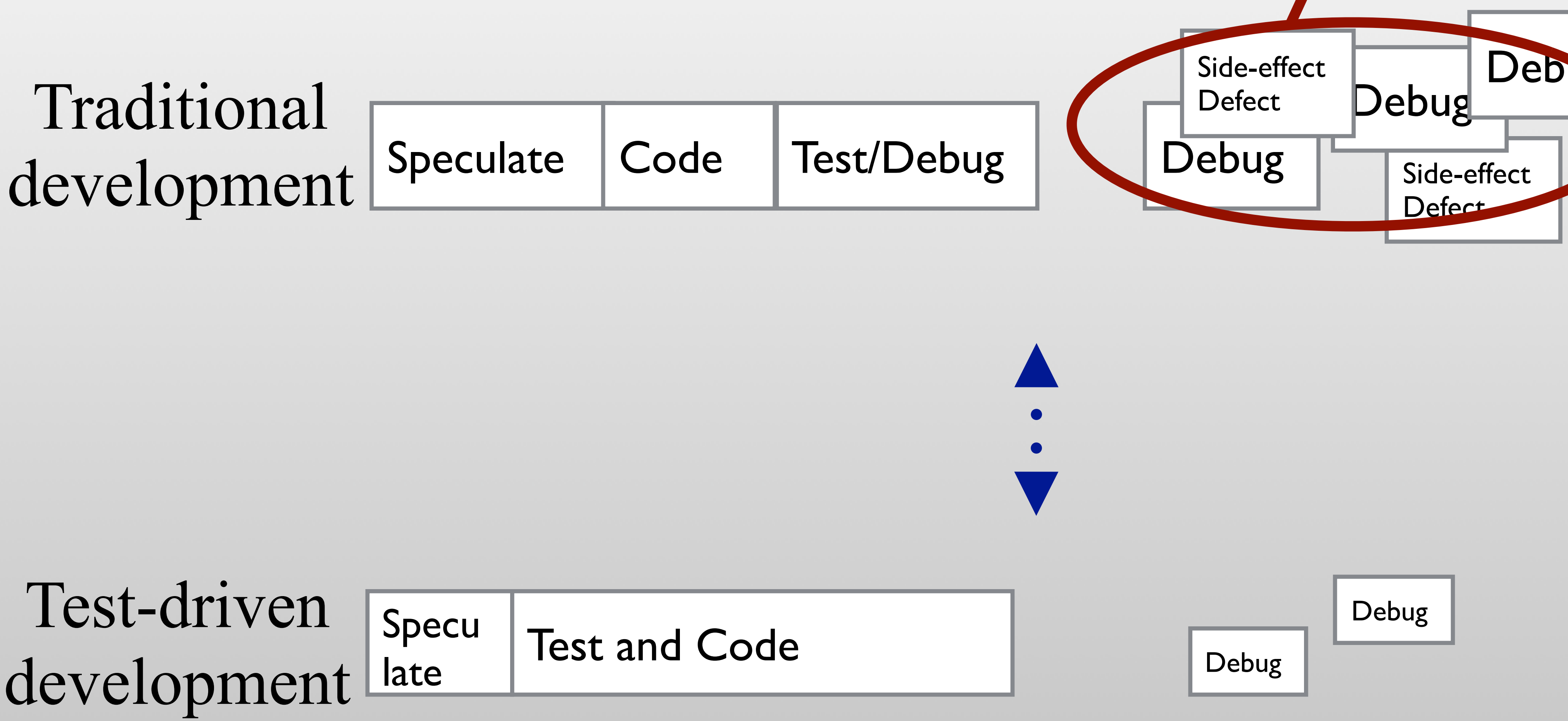
A Complex system that works is invariably found to have evolved from a simple system that worked.





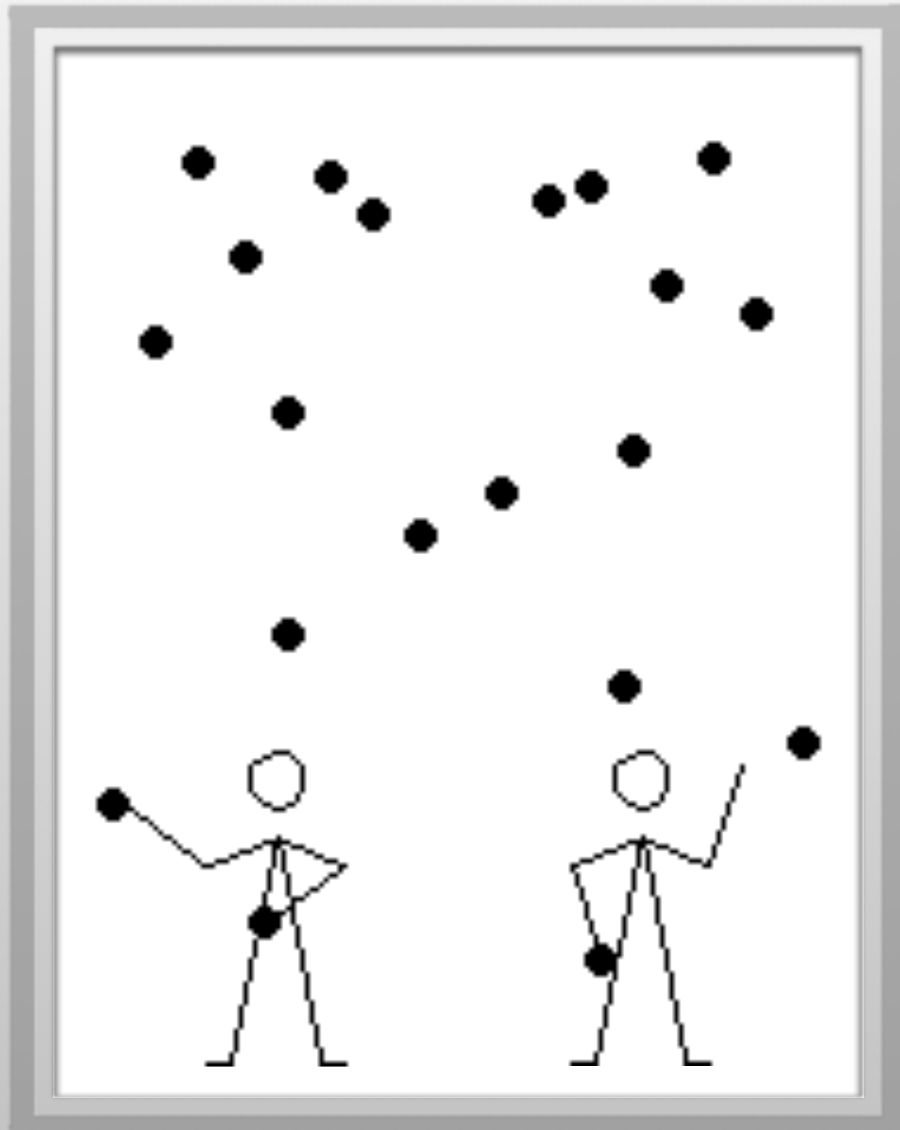
Sustainability

The cost of doing business debug-later-programming



Managing Cognitive Load

Debug Later Programming is Like Juggling



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

How Many Knives Do You Like to Juggle?

- TDD is solving one problem at a time

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

- Get good at being careful and you can go fast.

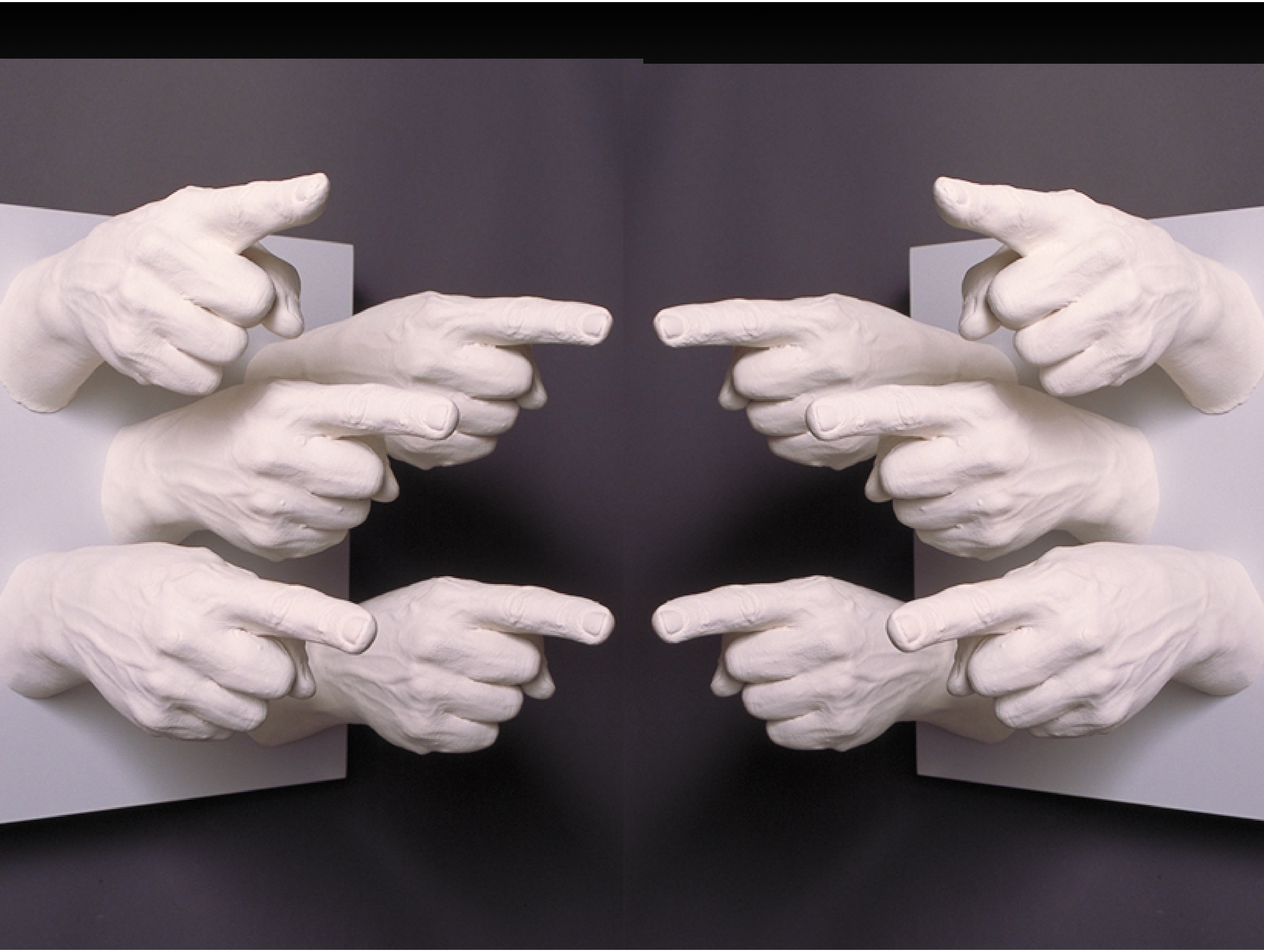
Adaptation for Embedded Software

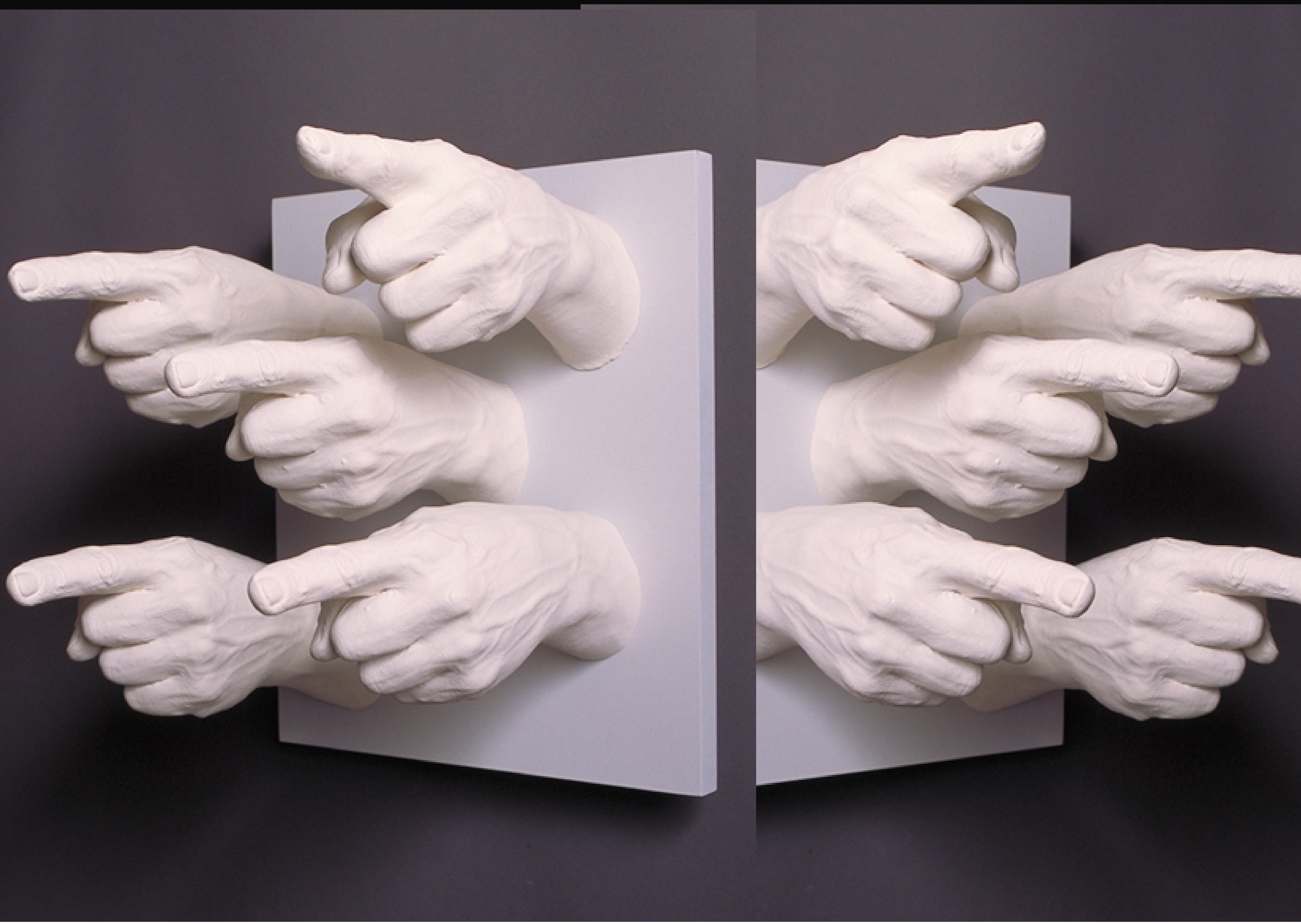
What is special about embedded

- Concurrent HW development
- HW bottleneck
- Cross compilation
- Limited memory and IO
- Target debug
- Architecture

Hardware is Scarce!

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





Minimize DoH!

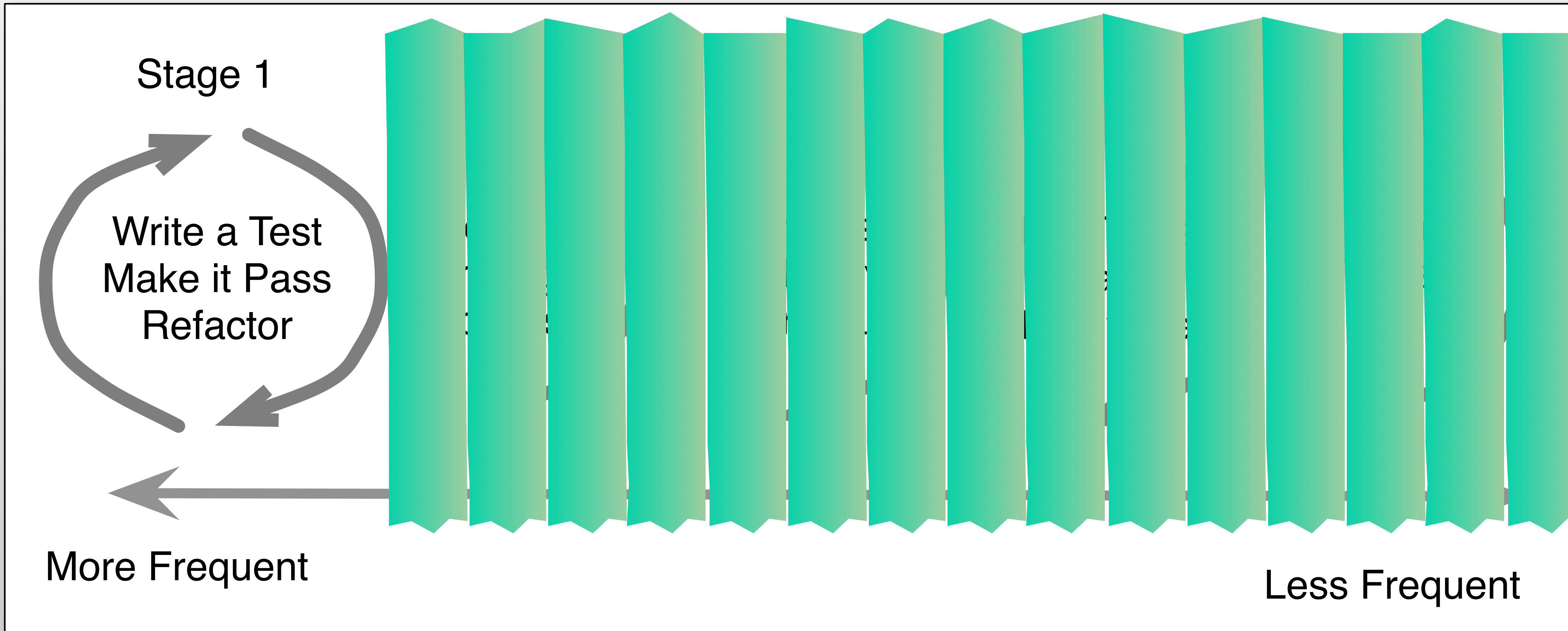
Copyright Fox Broadcasting. Used under fair use.

Debug

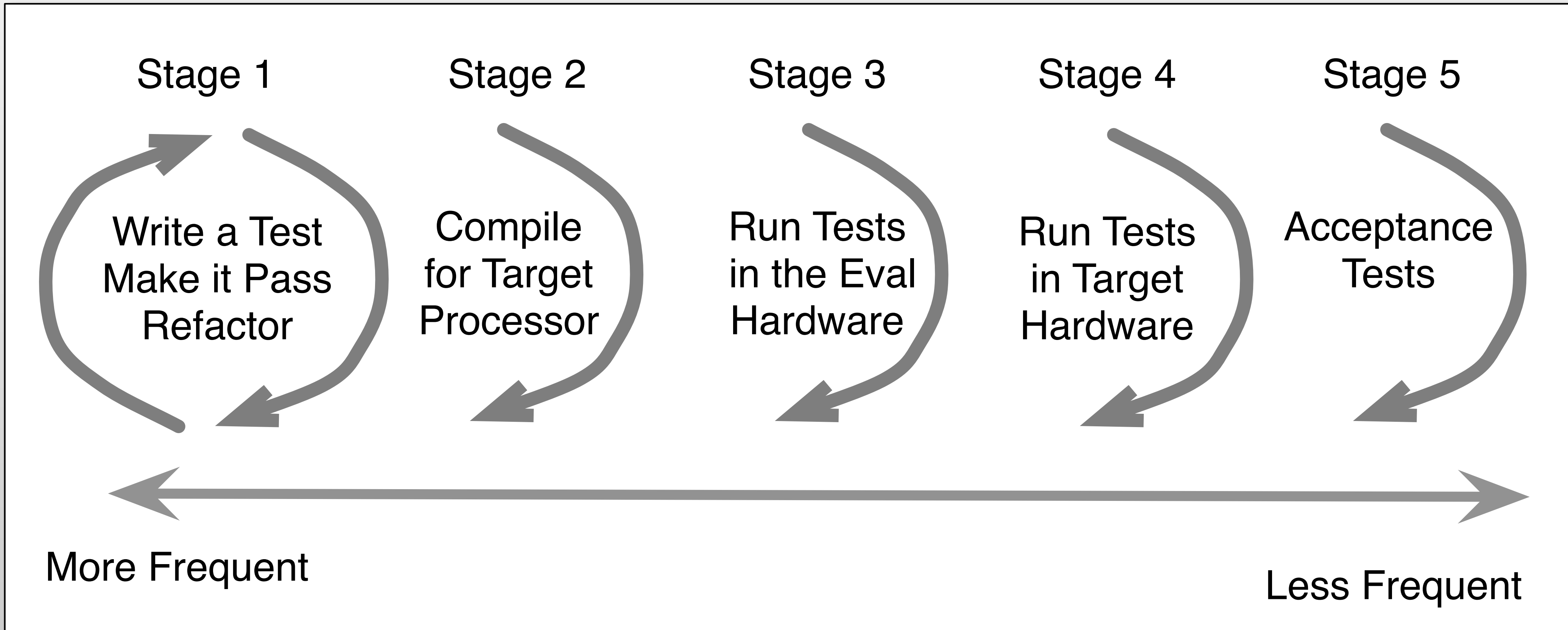
On

Hardware

TDD Adaptation for Embedded Development

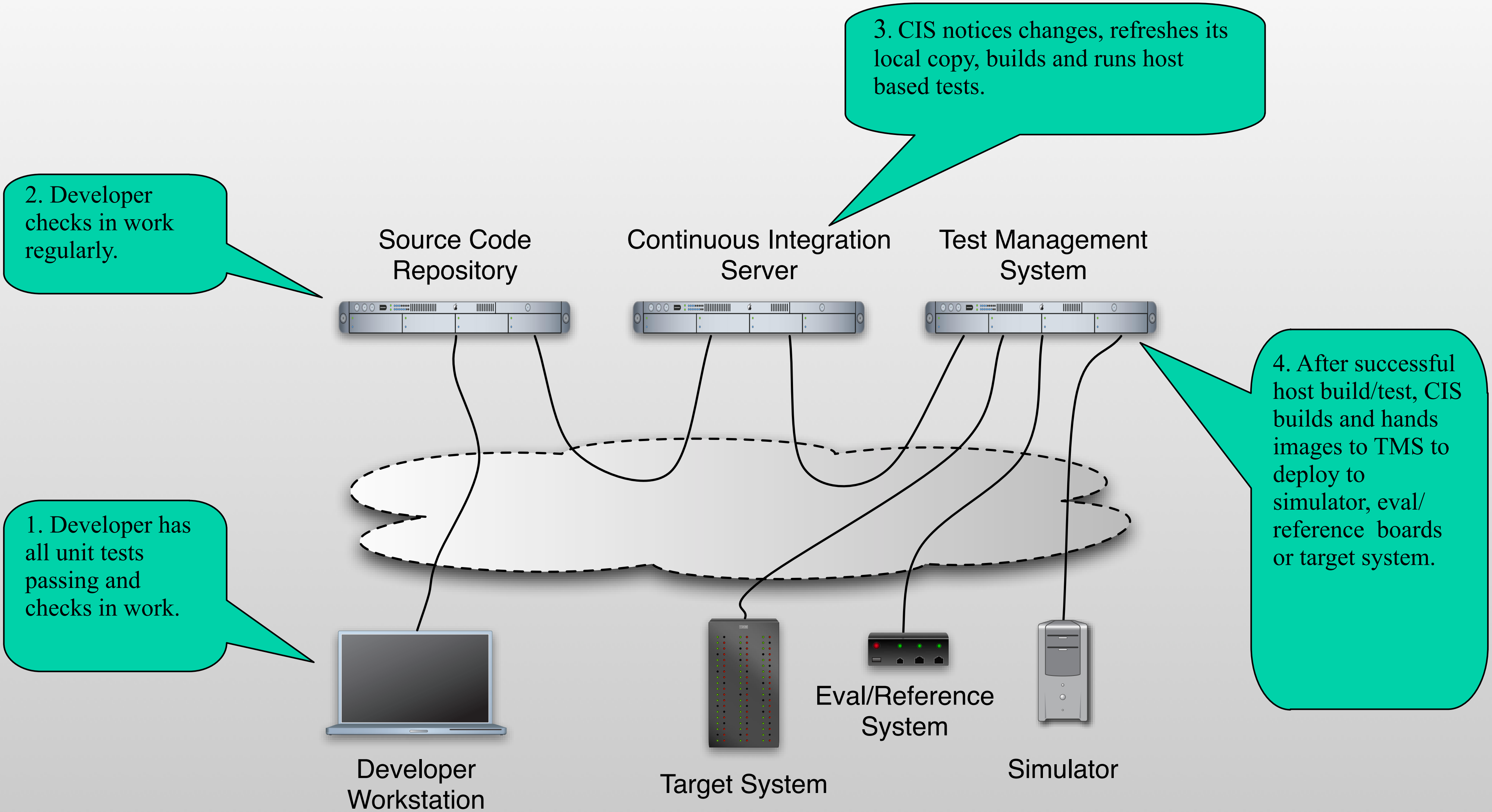


TDD Adaptation for Embedded Development



See : http://renaissancesoftware.net/files/articles/ProgressBeforeHardware.pdf

Continuous Integration - Embedded



The SCR, CIS and TMS are not necessarily separate hardware systems.

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

Problems Found at Appropriate Stage

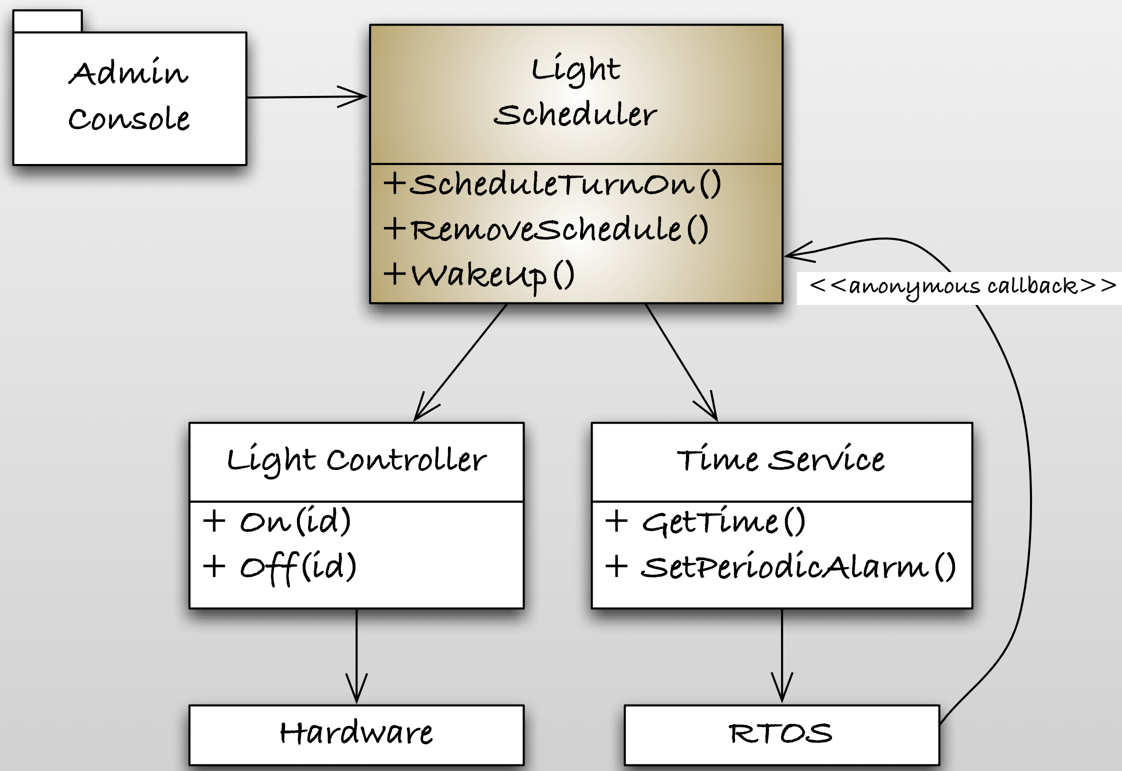
Stage	Problems Likely to Find in Stage
1	Logic, design, modularity, interface, boundary conditions
2	Compiler compatibility (language features) Library compatibility (header files, declarations)
3	Processor executions problems (compiler and library bugs) Portability problems (word size, alignment, endian)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

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

How do I test Code with Dependencies?

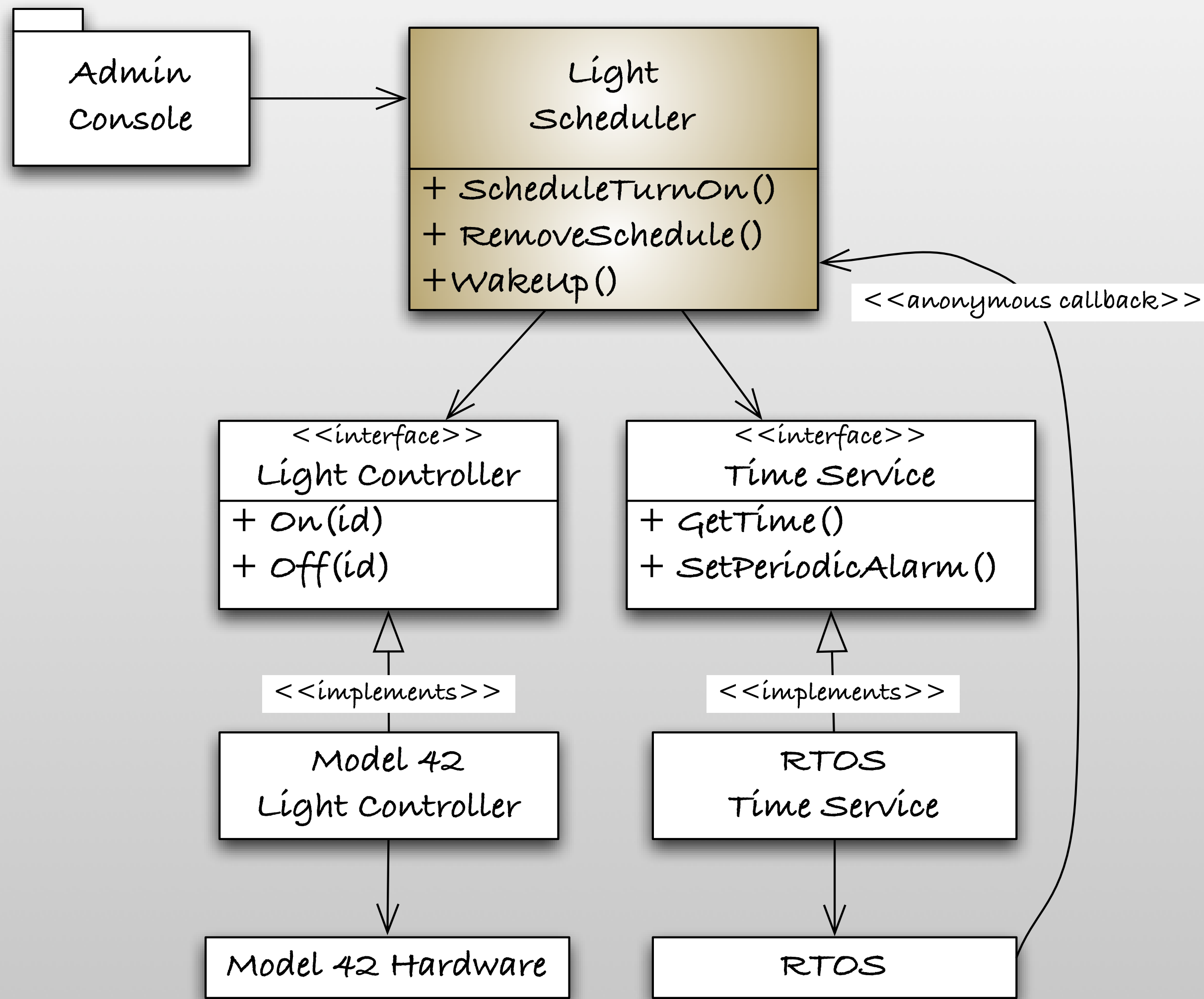
A Module with Collaborators

- 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.



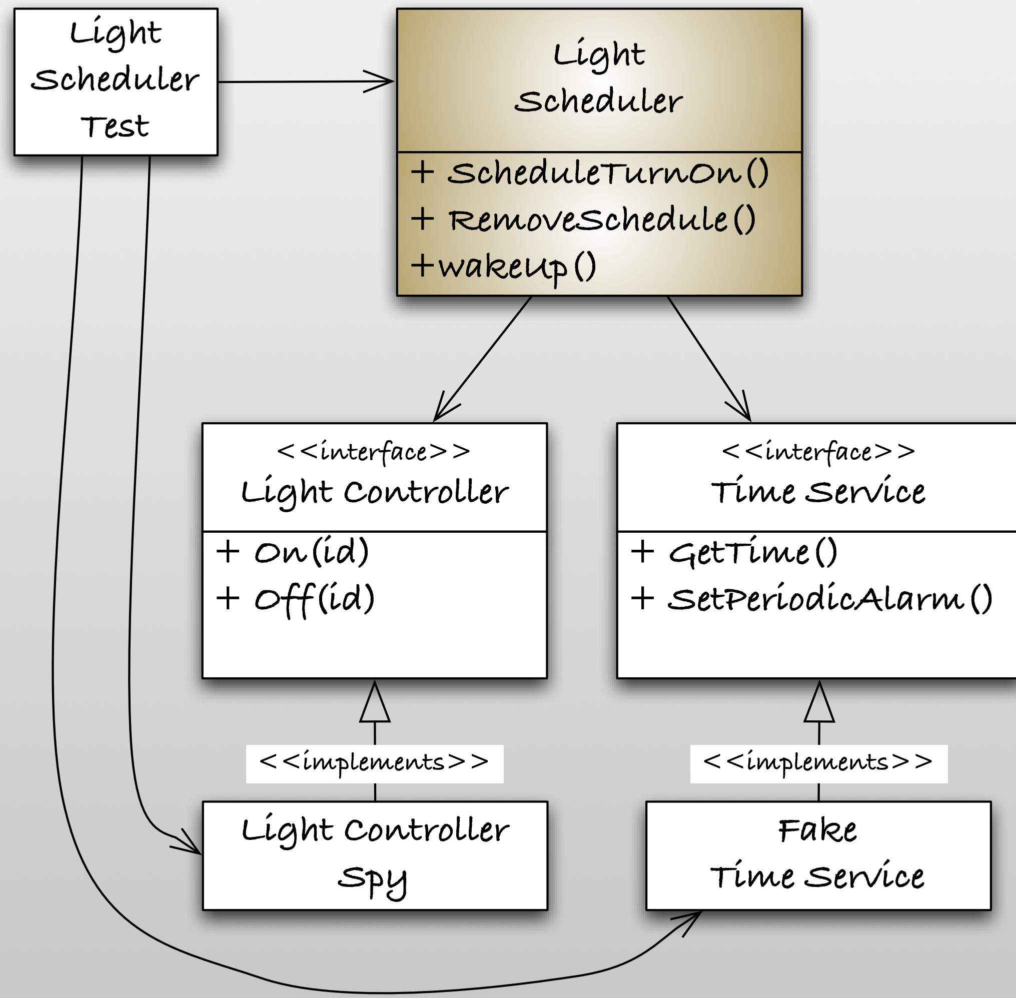
Program to Interfaces

- Separate interface and implementation as separate entities.
- This design has good separation of responsibilities



Testing a Module with Collaborators

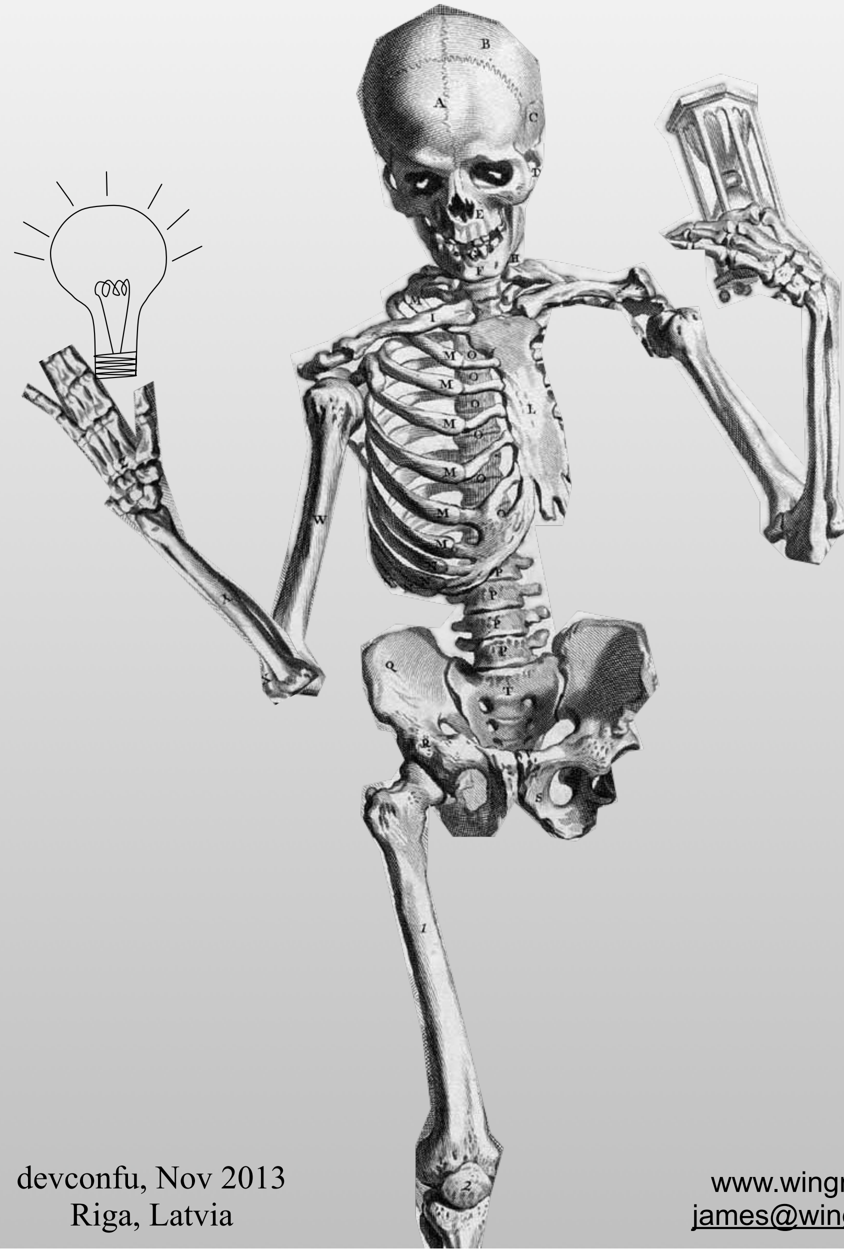
- Use the real collaborators if you can.
- Use fakes when you must.



Testing the Scheduler

```
TEST(LightScheduler, ScheduleOnTodayNotTimeYet)
{
    LightScheduler_ScheduleTurnOn(3, EVERYDAY, 1000);
    FakeTimeSource_SetMinute(999);

    LightScheduler_Wakeup();

    LONGS_EQUAL(LIGHT_NA, LightControllerSpy_GetState(3));
}

TEST(LightScheduler, ScheduleOnTodayItsTime)
{
    LightScheduler_ScheduleTurnOn(3, EVERYDAY, 1000);
    FakeTimeSource_SetMinute(1000);

    LightScheduler_Wakeup();

    LONGS_EQUAL(LIGHT_ON, LightControllerSpy_GetState(3));
}
```

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



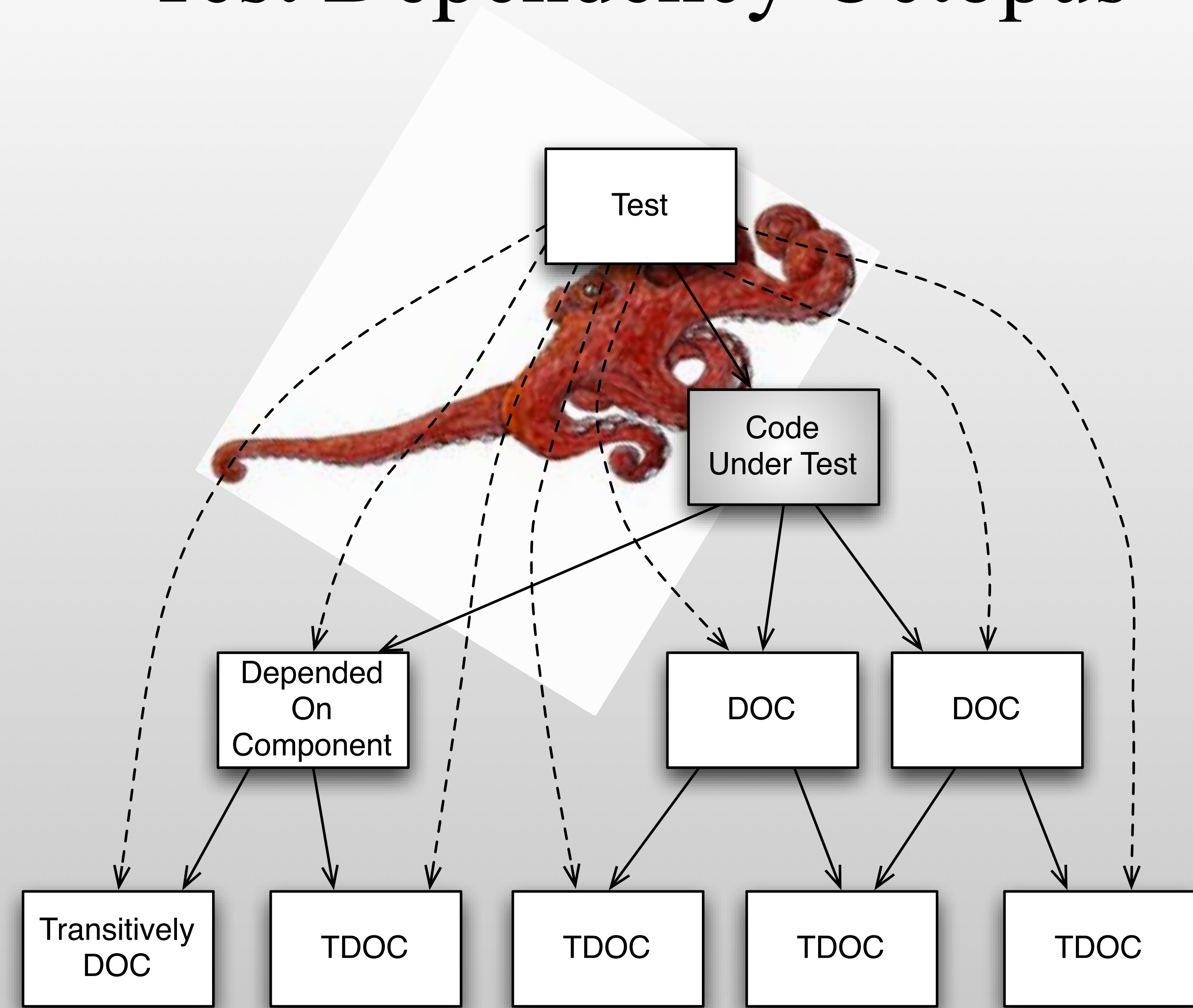
Test-Doubles

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

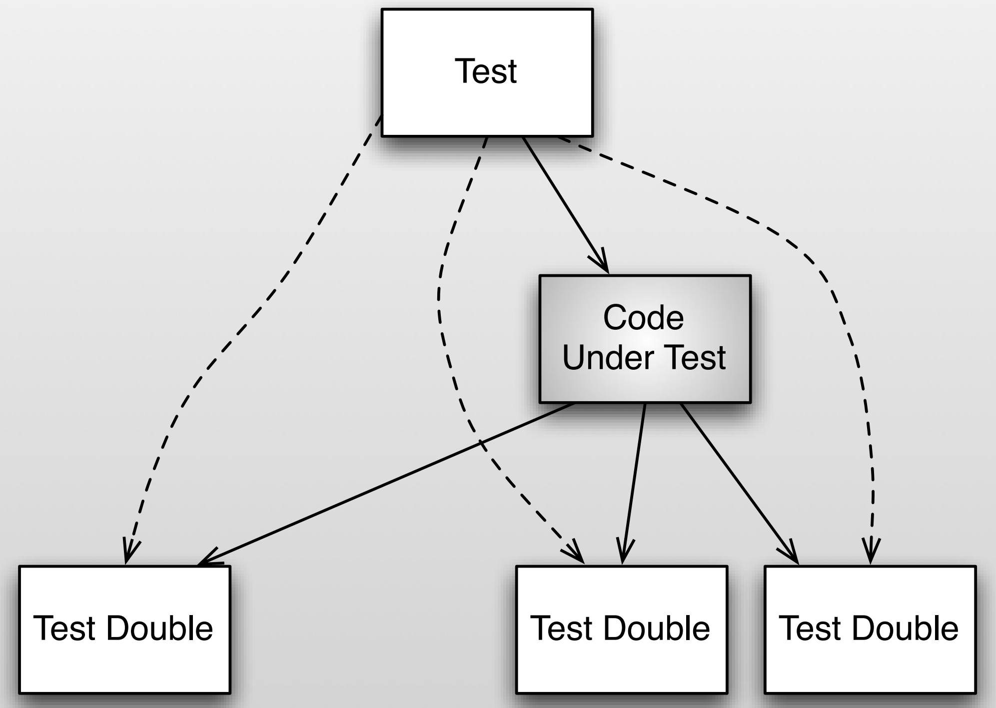
Test Double

- 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

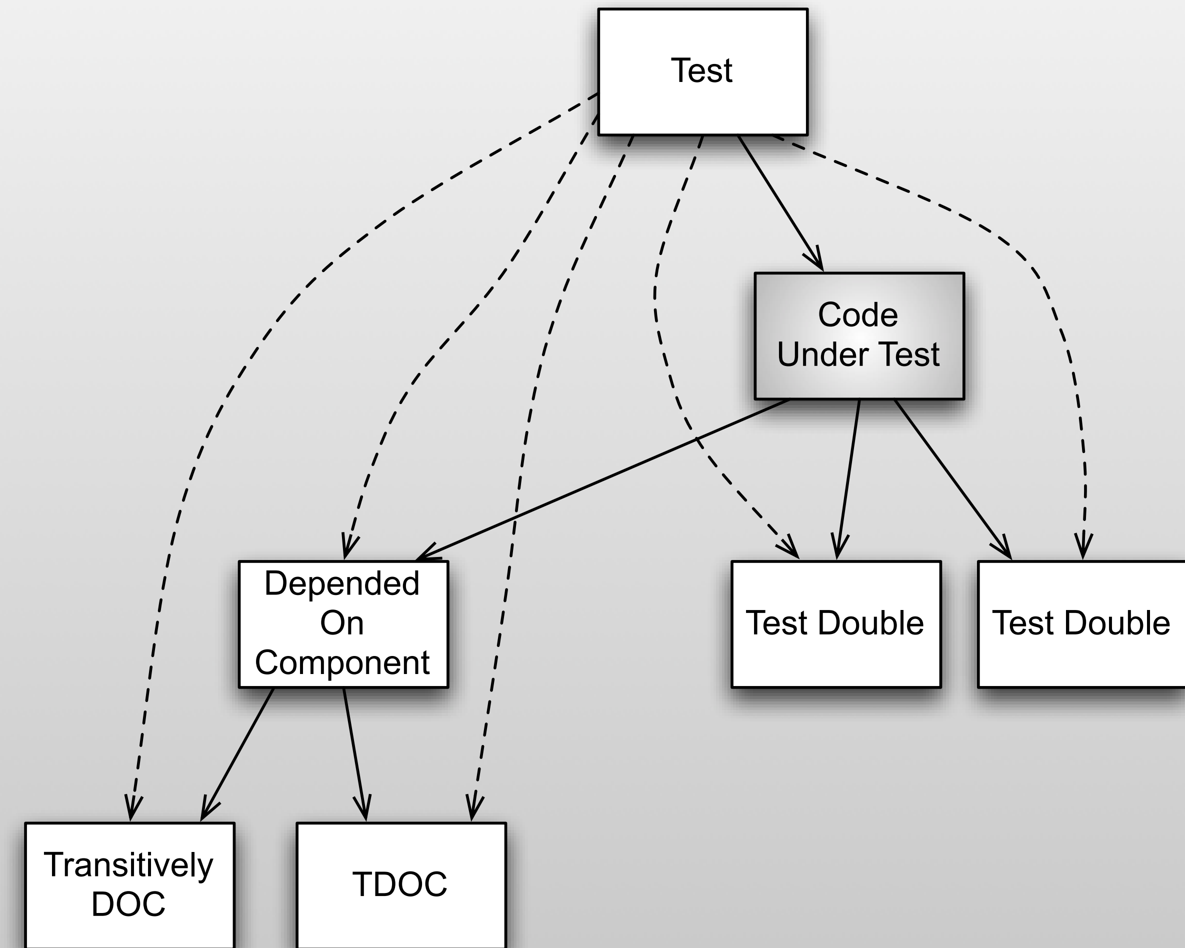
Test Dependency Octopus



Managing the Octopus



Use Production Code when you Can
Test Doubles when you Must



Kinds of Test Doubles
[XUNIT]

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

What are the Test Double Options in C?

- Link-time substitution.
  - Substitute whole object files or libraries
- Run time substitution.
  - Function pointers.
  - Function pointer tables.
- Preprocessor substitution
  - My least favorite, but sometimes necessary
- Use the simplest option that will work.

What are the Test Doubles Options in C++?

- Link-time substitution.
  - Substitute whole object files or libraries
- Run time substitution.
  - To substitute one class for another
    - Virtual functions.
    - Interface classes.
  - To substitute one function for another
    - C function pointers.
- Preprocessor substitution
  - my least favorite, but sometimes necessary
- Use the simplest option that will work.

Getting Around Tool Dependencies

Example Problems

- Compiler vendors *extend* C
- Embedded ‘`asm`’
- Header files lock you to the vendor

Compiler vendors *extend* C

```
//snip...
/* Address Mode Register */
extern cregister volatile unsigned int AMR;
//snip...

//snip...
interrupt void one_ms_tic(void)
{
  ...
}
//snip...
```

Hide Them with a Preprocessor Forced Include

```
//hide_stuff.h

#define interrupt
#define cregister
```

Force Include hidestuff.h, to make them go away

read the blog: http://www.renaissancesoftware.net/blog/archives/249

Developers embed '`asm`'

```
void foo(void)
{
    do
    {
      asm("NOP");
      asm("PLOP");
      asm("READ 0");
      status = IORead(0);
    } while ((status &amp; ReadyBit) == 0);
}
```

Hide **asm** Forced Include

```
//nullasm.h

#define asm
```

Force Include nullasm.h, to make asm go away

read the blog: http://www.renaissancesoftware.net/blog/archives/136

`AsmSpy` Test Double

```
#define asm AsmSpy

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

Force Include AsmSpy.h All `asm()` directives are converted to `AsmSpy()` calls, allowing capture.

read the blog: http://www.renaissancesoftware.net/blog/archives/136

AsmSpy Tests Show How to Monitor the asm

```
TEST(AsmSpy, CaptureAsmInstructions)
{
     AsmSpy("NOP");
     AsmSpy("FLOP");
     AsmSpy("POP");
     STRCMP_EQUAL("NOP;FLOP;POP;", AsmSpy_Debrief());
}
```

Force Include AsmSpy.h, to capture asm instructions

read the blog: http://www.renaissancesoftware.net/blog/archives/136

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

Header Files that Lock You to the Vendor's Compiler

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

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

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

Create a `#include` Test Double

```
#include <stdint.h>

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

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

#endif /* ACMETYPES_H_ */
```

Also, limit dependencies on vendor specific includes

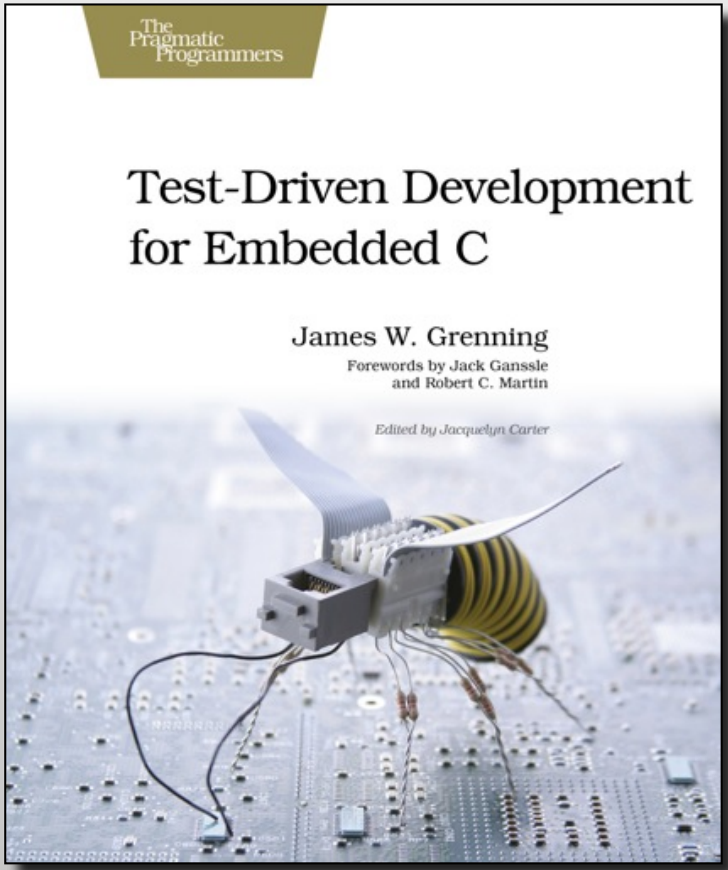
There is a Lot More to TDD

Talk to me on Twitter
@jwgrenning

Find my book at
http://www.pragprog.com/titles/jgade

Find me on linkedin.com
http://www.linkedin.com/in/jwgrenning
Please remind me how we met.

http://www.wingman-sw.com
http://www.wingman-sw.com/blog
http:// www.jamesgrenning.com



DevConFu

Developers Conference for You

Partners:

devtraining

Revelan.eu

web-development

Spotify

developer garden

devclub.ee

devclub.eu

devclub.ru

devclub.lv