

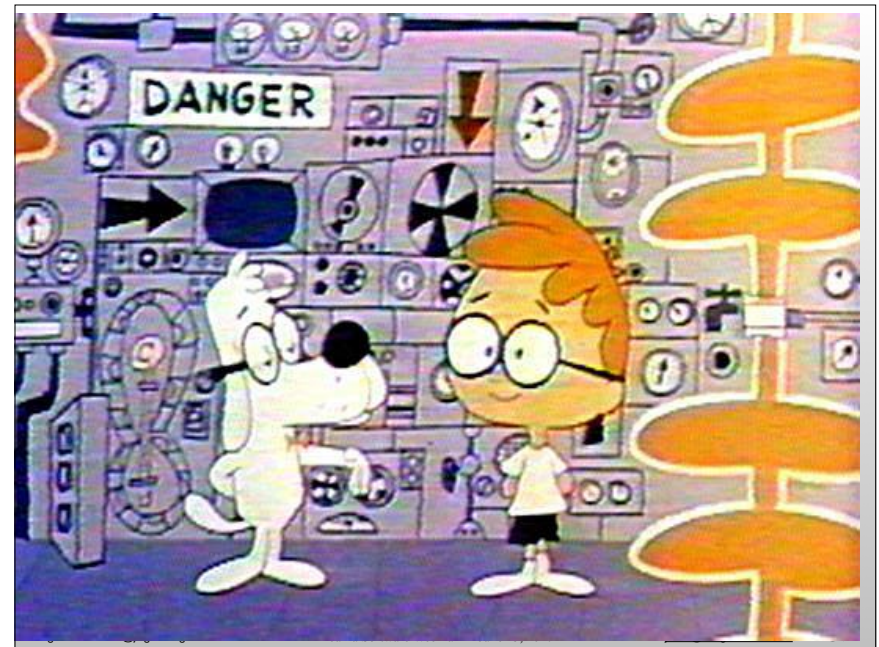






James W Grenning  
[wingman-sw.com](http://wingman-sw.com)  
 @jwgrenning  
<http://facebook.com/wingman-sw>



Copyright © 2008-2017 James W. Grenning  
 All Rights Reserved @jwgrenning



# I Tried to Avoid Computers





Copyright © 2008-2017 James W. Grenning  
 All Rights Reserved @jwgrenning

Agile for Embedded — Overview and Pitfalls  
 Jama Software Webinar — June 15, 2017

[www.wingman-sw.com](http://www.wingman-sw.com)  
[jama@wingman-sw.com](http://jama@wingman-sw.com)

3



Copyright © 2008-2017 James W. Grenning  
 All Rights Reserved @jwgrenning

Agile for Embedded — Overview and Pitfalls  
 Jama Software Webinar — June 15, 2017

4

People will pay me to do this?!!



### 8251A PROGRAMMABLE COMMUNICATION INTERFACE

- Synchronous and Asynchronous Operation
- Synchronous 5-8 Bit Characters; Internal or External Character Synchronization; Automatic Sync Insertion
- Asynchronous 5-8 Bit Characters; Clock Rate—1, 16 or 64 Times Baud Rate; Break Character Generation; 1, 1½, or 2 Stop Bits; False Start Bit Detection; Automatic Break Detect and Handling
- Synchronous Baud Rate—DC to 64K Baud
- Asynchronous Baud Rate—DC to 19.2K Baud
- Full-Duplex, Double-Buffered Transmitter and Receiver
- Error Detection—Parity, Overrun and Framing
- Compatible with an Extended Range of Intel Microprocessors
- 28-Pin DIP Package
- All Inputs and Outputs are TTL Compatible
- Available in EXPRESS and Military Versions

2

The Intel® 8251A is the industry standard Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel's microprocessor families such as MCS-48, 80, 85, and iAPX-86, 88. The 8251A is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM "bi-sync"). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TXEMPTY. The chip is fabricated using Intel's high performance HMOS technology.

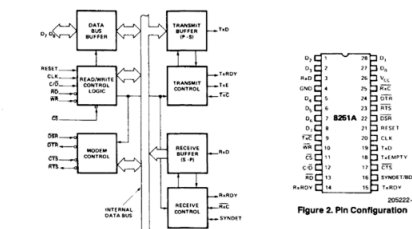


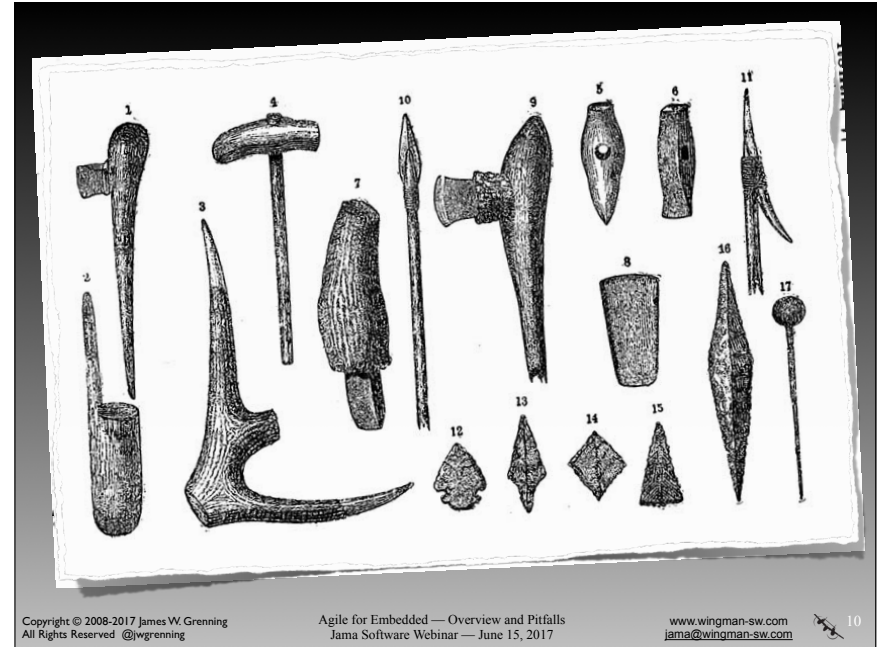
Figure 2. Pin Configuration







Copyright ©  
All Rights Re

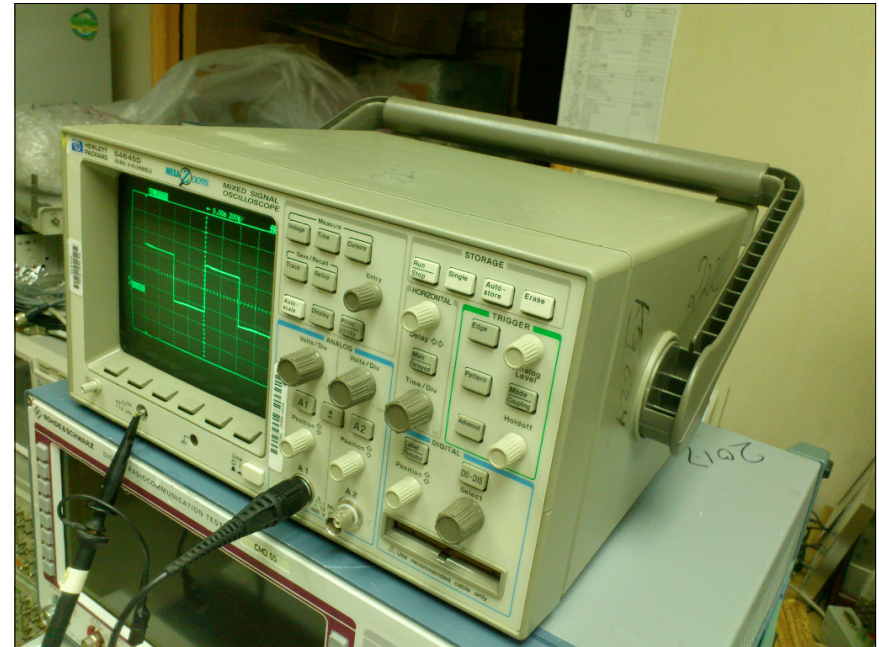
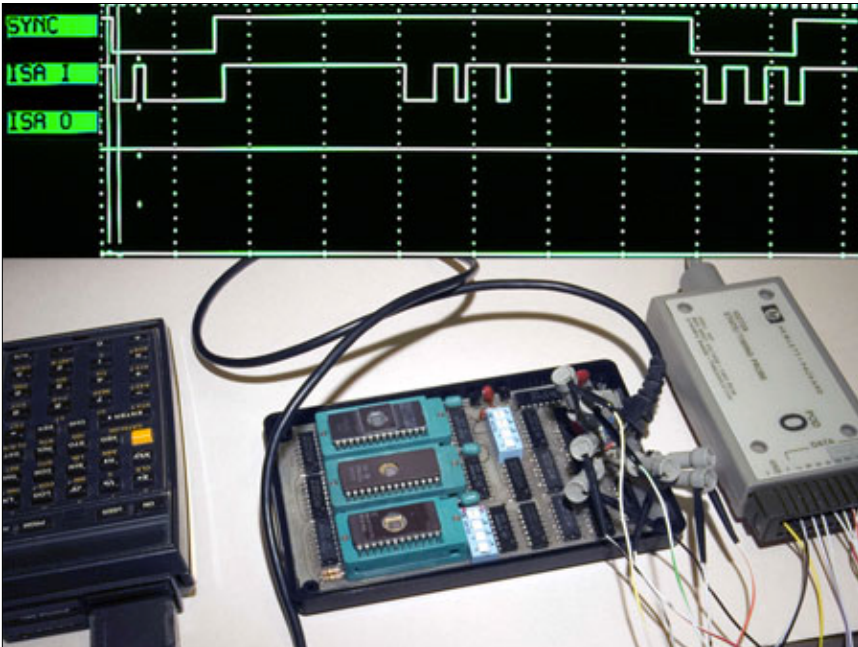


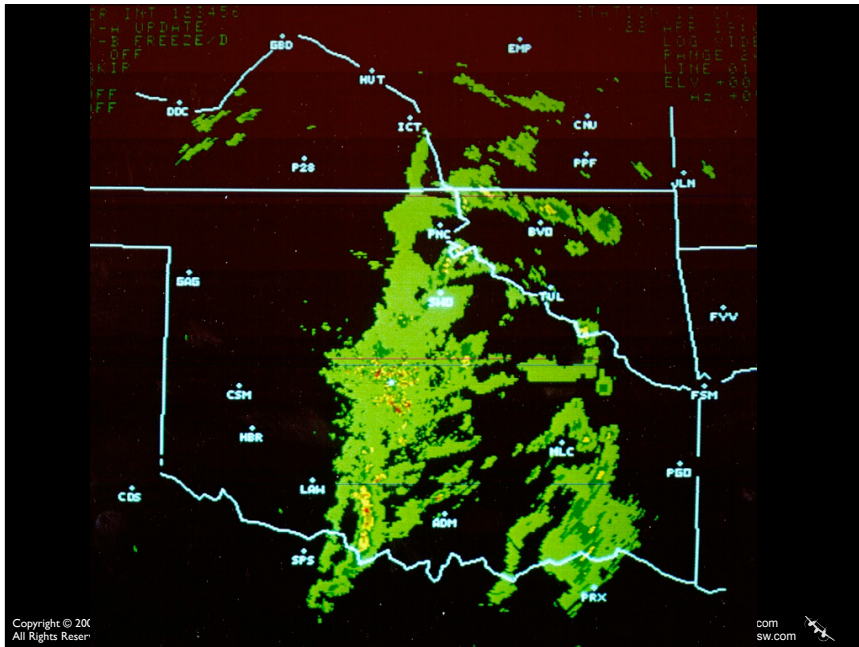
Copyright © 2008-2017 James V. Grenning  
All Rights Reserved @wgrnning

Agile for Embedded — Overview and Pitfalls  
Jama Software Webinar — June 15, 2017

www.wingman-sw.com  
jama@wingman-sw.com

10





# My Roots in Agile

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved @jwrenning

Agile for Embedded — Overview and Pitfalls  
Jama Software Webinar — June 15, 2017

www.wingman-sw.com  
jama@wingman-sw.com 14

?

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved @jwrenning

Agile for Embedded — Overview and Pitfalls  
Jama Software Webinar — June 15, 2017

www.wingman-sw.com  
jama@wingman-sw.com 15

## How Did I Find Myself at Agile Manifesto Meeting?

James, do you want to go to the Lightweight Methods Summit in Snowbird?

Skiing!

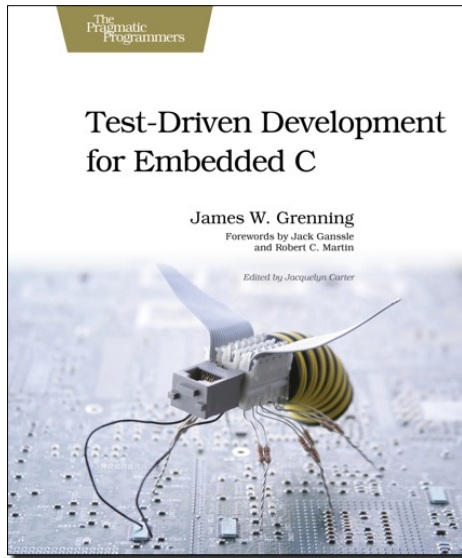
Sure Bob, I'll go....

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved @jwrenning

Agile for Embedded — Overview and Pitfalls  
Jama Software Webinar — June 15, 2017

www.wingman-sw.com  
jama@wingman-sw.com 16

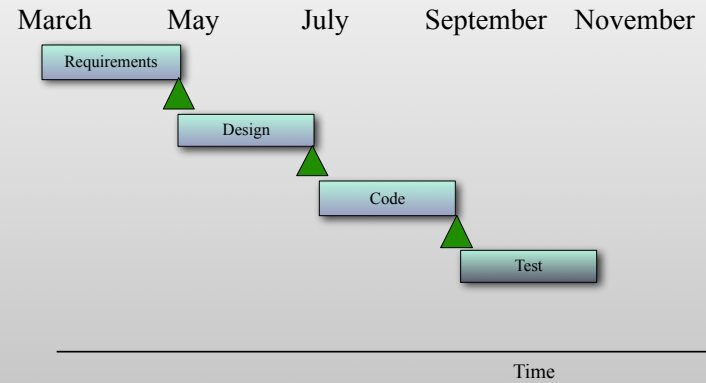




JAMA-2



## Waterfall Model



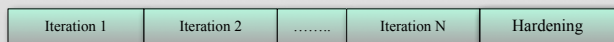
Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

Jama Software, Webinar -- August 24, 2017 -- JAMA-2  
Test Driven Development

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

18

## Iterative Model



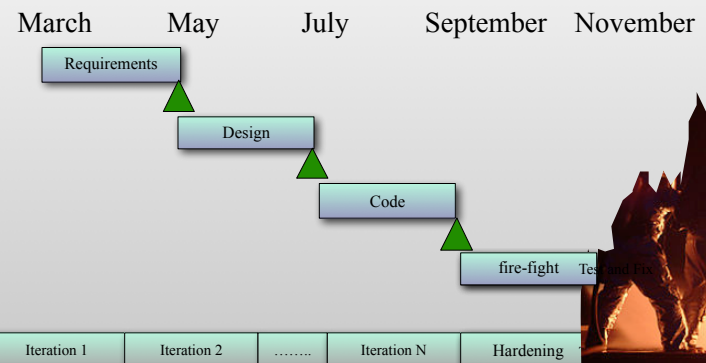
Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

Jama Software, Webinar -- August 24, 2017 -- JAMA-2  
Test Driven Development

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

19

## Saving Testing Until the the End

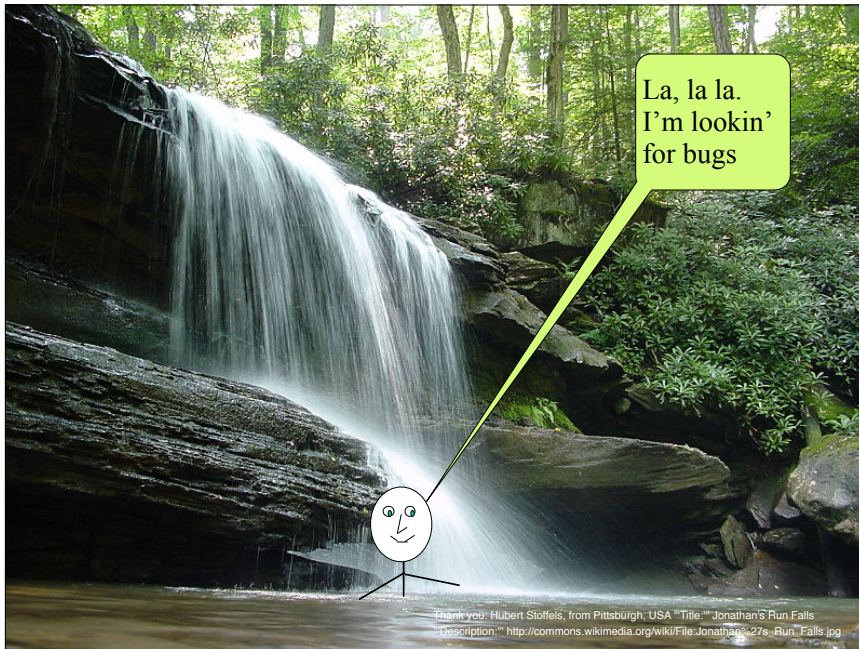


Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

Jama Software, Webinar -- August 24, 2017 -- JAMA-2  
Test Driven Development

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

20



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

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

Jama Software, Webinar -- August 24, 2017 -- JAMA-2

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

23

### The Last Bug

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

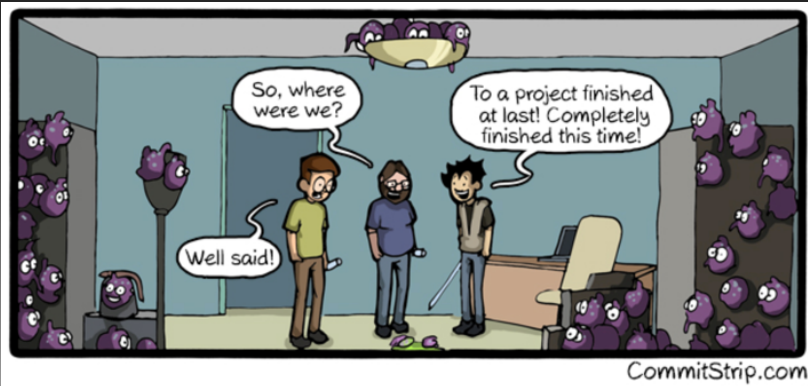
Jama Software, Webinar -- August 24, 2017 -- JAMA-2  
Test Driven Development

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

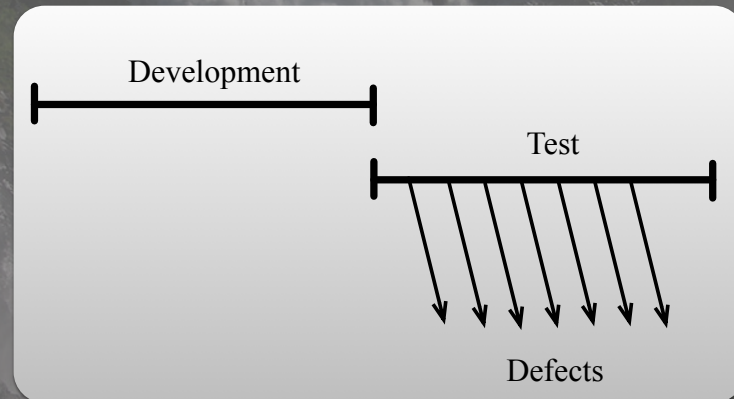
24



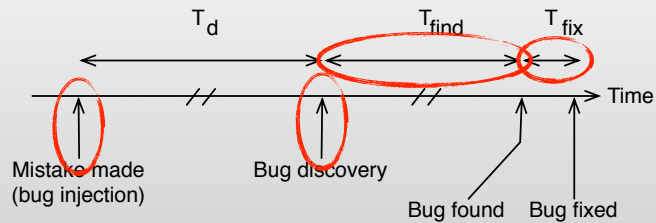
## Finished at Last



## This Work Flow is Designed to Allow Defects



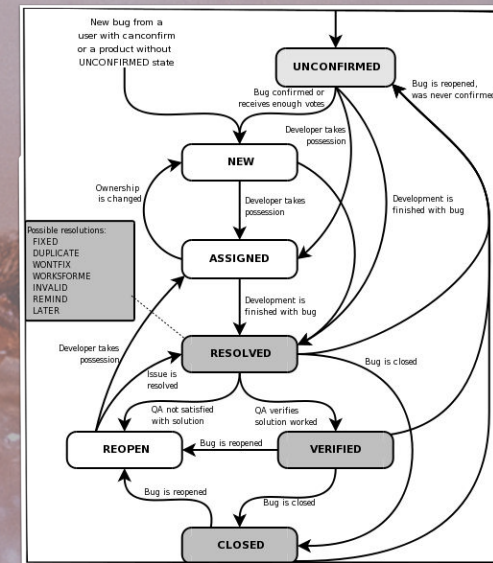
## The Physics of Debug Later Programming (DLP)



- 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>

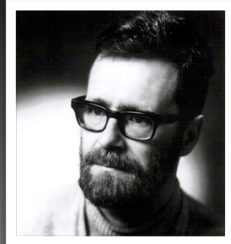
## A Bug's Life



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

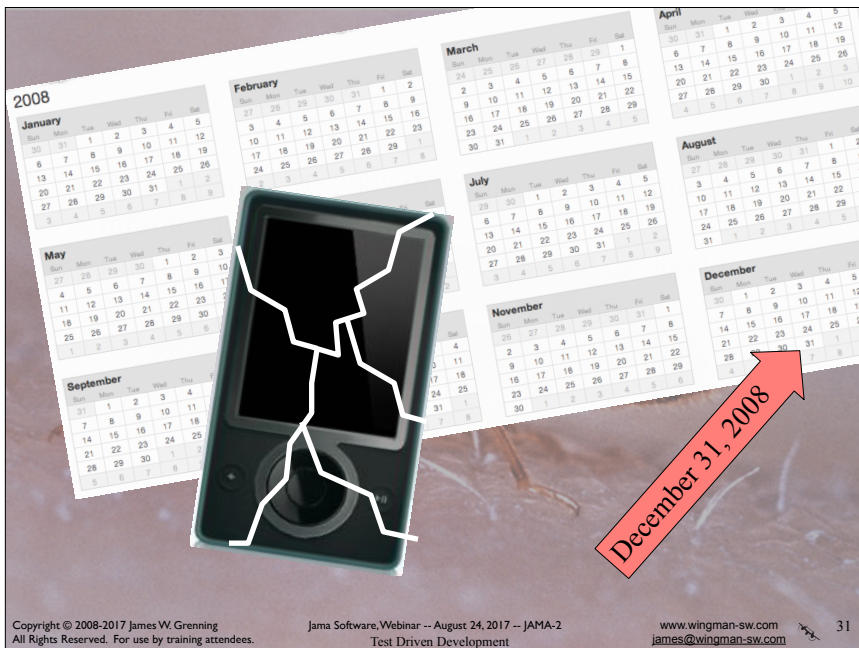
## Edsger Dijkstra

*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.*



## The First Step to Recovery

I am a programmer and I write bugs



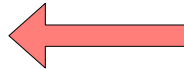
```
BOOL ConvertDays(UINT32 days, SYSTEMTIME* lpTime)
```



## One That Got Away

```
static void SetYearAndDayOfYear(RtcTime* time)
{
    int days = time->daysSince1980;
    int year = STARTING_YEAR;
    while (days > 365)
    {
        if (IsLeapYear(year))
        {
            if (days > 366)
            {
                days -= 366;
                year += 1;
            }
        }
        else
        {
            days -= 365;
            year += 1;
        }
    }

    time->dayOfYear = days;
    time->year = year;
}
```



## 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, LARGE AND SMALL  
REVISI  
THE THIRD EDITION OF SYSTEMS  
BY  
**JOHN GALL**

## Bugs Can Hide Anywhere. What has to be tested?



CommitStrip.com

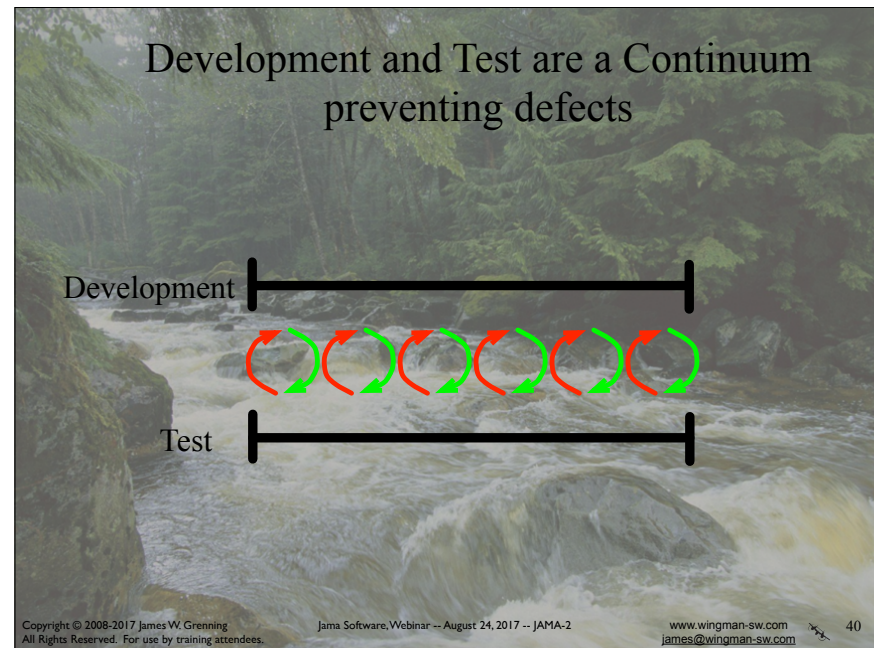
Can we Realize Dijkstra's Vision?

Can we Prevent Defects  
with Test-Driven  
Development?

Being good at chasing bugs  
is not Technical Excellence

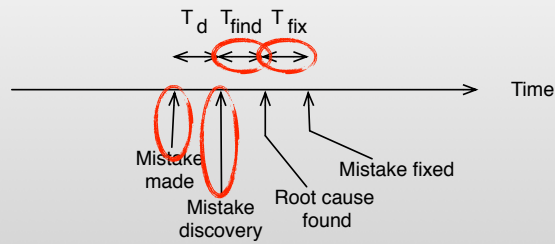


T  
D  
D



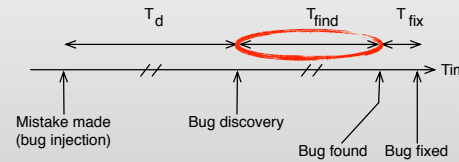


# The Physics of Test Driven Development

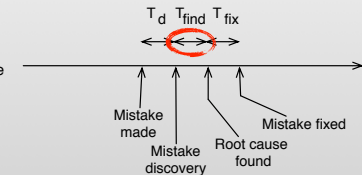


- When  $T_d$  approaches zero,  $T_{find}$  approaches zero
- In many cases, bugs are not around long enough to be considered bugs.
- See: <https://wingman-sw.com/articles/the-physics-of-test-driven-development>

# The Physics of Debug Later Programming

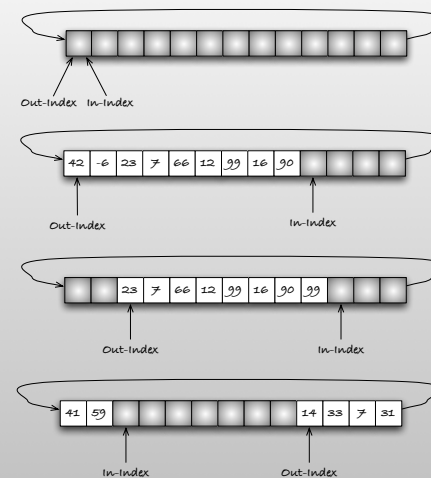


# The Physics of Test-Driven Development

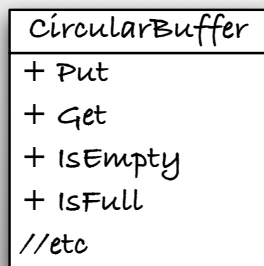


# TDD Example

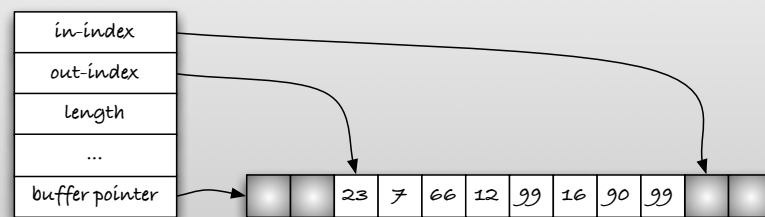
# Create a Circular Buffer Module



## The Anticipated Interface



## Anticipated Internal Structure



## 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

```
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



When you code for an hour

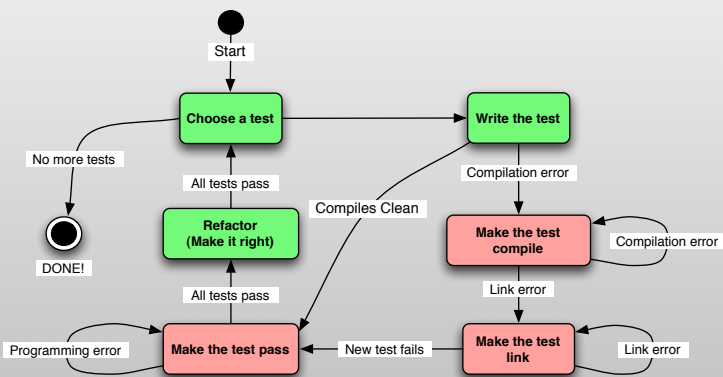
How long do you fight the compiler?

How long do you search for problems?

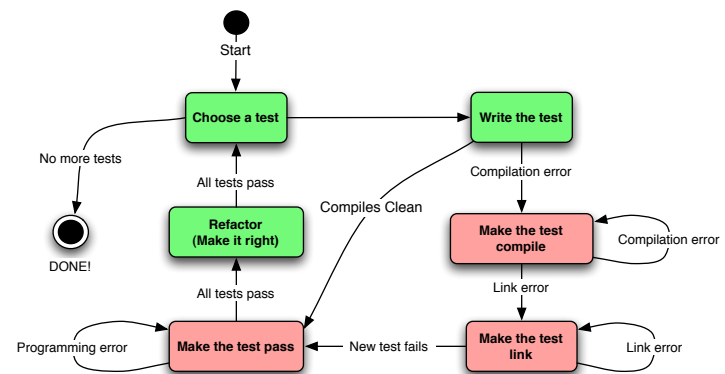
What if there was only one outstanding problem at a time?

## Continue Until You Can't Think of Any More Tests

When your tests are done and your code is done.



## TDD State Machine in C/C++ (solving one problem at a time)



Add new tests to the test list as needed

## Small Verifiable Steps

- Each change is verified
- Maybe by seeing the test output
- Maybe by seeing how the tools react

## Adaptation for Embedded Software

## What is Special About Embedded Software?

Limited memory

Hard to debug

HW has bugs

Concurrent HW/SW development

Primitive UI

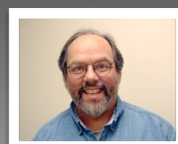
Special tools

No HW until late in cycle

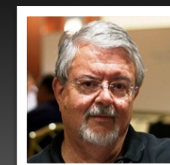
Real time debugging



Kent Beck



Ward Cunningham



Ron Jeffries



Martin Fowler

?





## Hardware has its own Defects

A collage of images illustrating hardware defects. It includes a close-up of a brown bug on a pink surface, a circuit board with numerous components, and a person standing on a sidewalk whose body is composed of various electronic hardware components like circuit boards and wires.

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

Jama Software, Webinar -- August 24, 2017 -- JAMA-2  
Test Driven Development - Embedded

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

66

A group of several hands, appearing to be from different people, are shown pointing towards the center of the frame. The hands are arranged in a circular pattern, creating a sense of collective attention or focus.

A group of several hands, appearing to be from different people, are shown pointing towards the center of the frame. The hands are arranged in a circular pattern, creating a sense of collective attention or focus.

Minimize DoH!



Debug  
On  
Hardware

Copyright Fox Broadcasting. Used under fair use.

Target Hardware  
Bottleneck

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

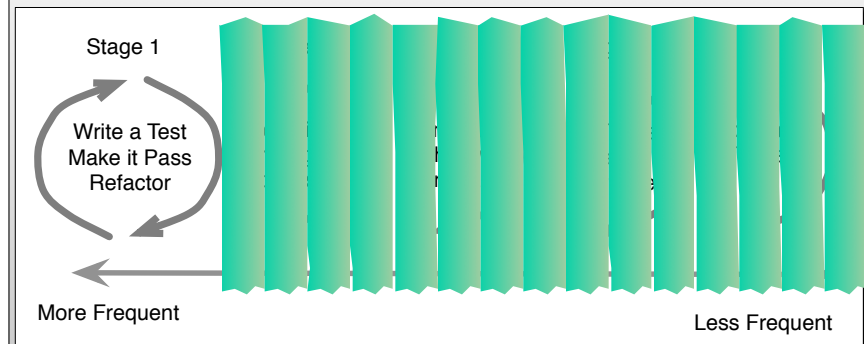


## How Long is Your Edit, Build, Run Cycle?

- Do you lose focus while you wait?

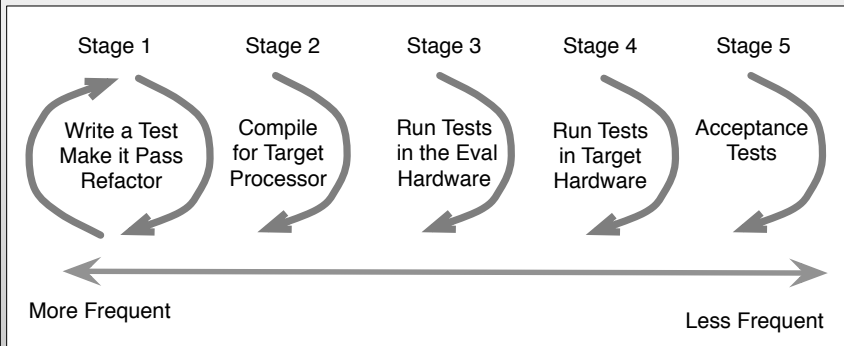


## TDD Adaptation for Embedded Development





## TDD Adaptation for Embedded Development



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

There is a lot more  
to talk about!!

Code with  
dependencies

Faking,  
mocking  
spying

Legacy code

Your concerns  
and questions

TDD next to  
the Silicon

Writing  
good tests

SOLID Design

Refactoring

You can learn more at  
[wingman-sw.com](http://wingman-sw.com)

**WINGMAN SOFTWARE:**

BRINGING AGILITY TO EMBEDDED SOFTWARE DEVELOPMENT

JOIN OUR MAILING LIST

LIVE VIA WEB TDD TRAINING!

PUBLIC TRAINING

ALL OUR TRAINING

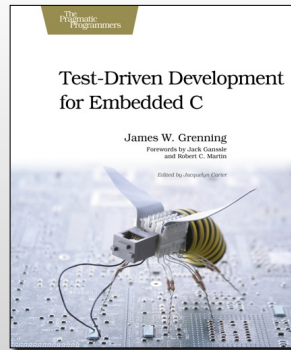
You can learn more

- At one of my public training events (fall 2017)
  - <https://wingman-sw.com/training/public>
- Web delivered training
  - <https://wingman-sw.com/training/remote-delivered-tdd>
- Join my mailing list
  - <https://wingman-sw.com/join-mailing-list>
- Find out about on-site training
  - <https://wingman-sw.com/contact-us>

Talk to me on Twitter  
@jwgrenning

Connect with me  
[www.linkedin.com/in/jwgrenning](http://www.linkedin.com/in/jwgrenning)  
Remind me how we met.

[www.wingman-sw.com](http://www.wingman-sw.com)  
[blog.wingman-sw.com](http://blog.wingman-sw.com)  
[www.jamesgrenning.com](http://www.jamesgrenning.com)  
[facebook.com/wingman.sw](https://facebook.com/wingman.sw)



[wingman-sw.com/tddc/](http://wingman-sw.com/tddc/)

# Unit Tests are Necessary but Not Sufficient

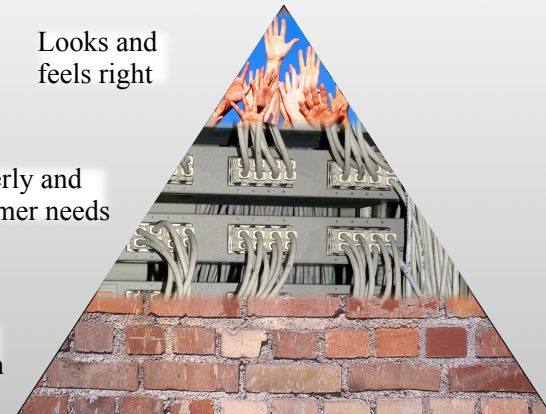


<http://blog.mountaingoatsoftware.com/the-forgotten-layer-of-the-test-automation-pyramid>

Looks and feels right

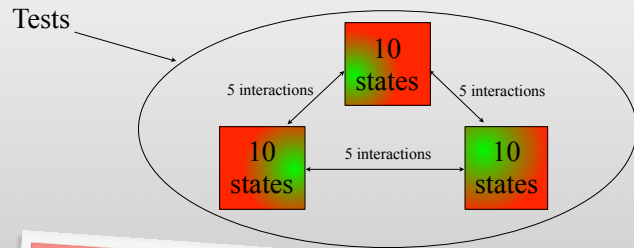
Wired properly and meets customer needs

Solid foundation





## Higher Level Tests Cannot be Thorough

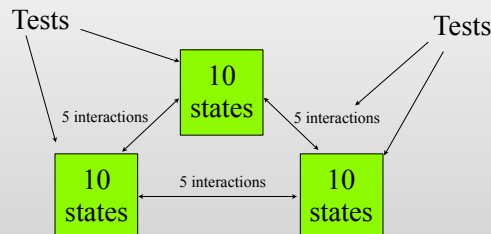


1000 (or more) tests are needed to test this simple system

## Higher Level Tests are Needed to Verify Functionality and Interconnections



## Unit Tests Can Be Thorough

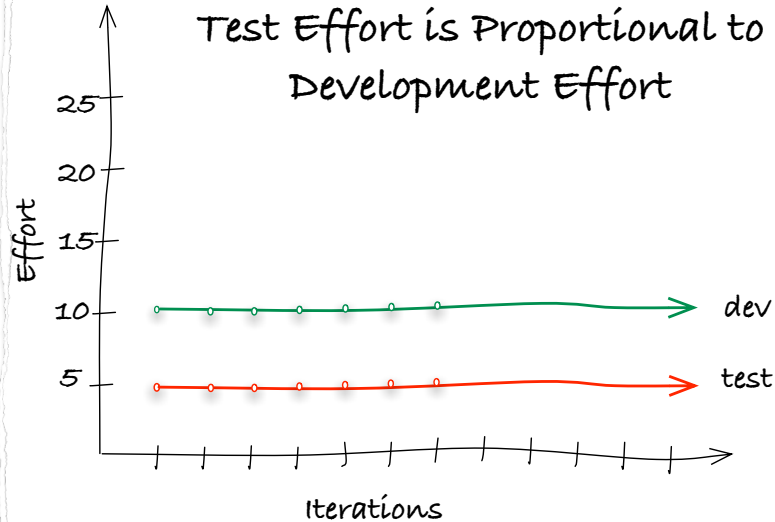


As few as 30 unit tests and 15 integration test when tested as units



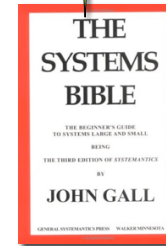
## Manual Test is Not Sustainable

Illustrates the Assumption that  
Test Effort is Proportional to  
Development Effort



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

Systems don't  
appreciate being fiddled  
and diddled with.



The fixing of known  
bugs will surely inject  
unknown bugs

System

Antics

System - something setup to attain a goal

Antics - amusing, frivolous, or eccentric  
behavior

System

Antics





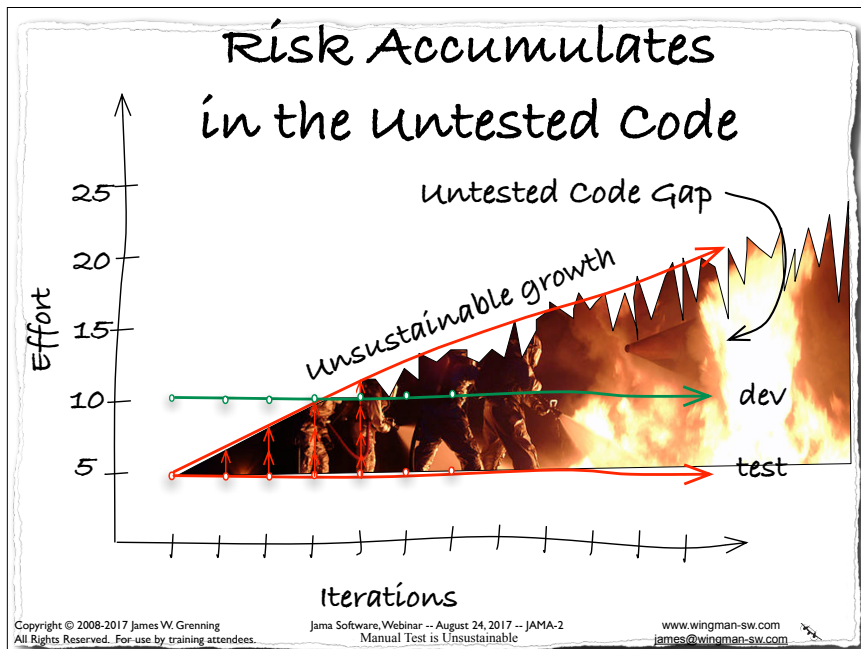
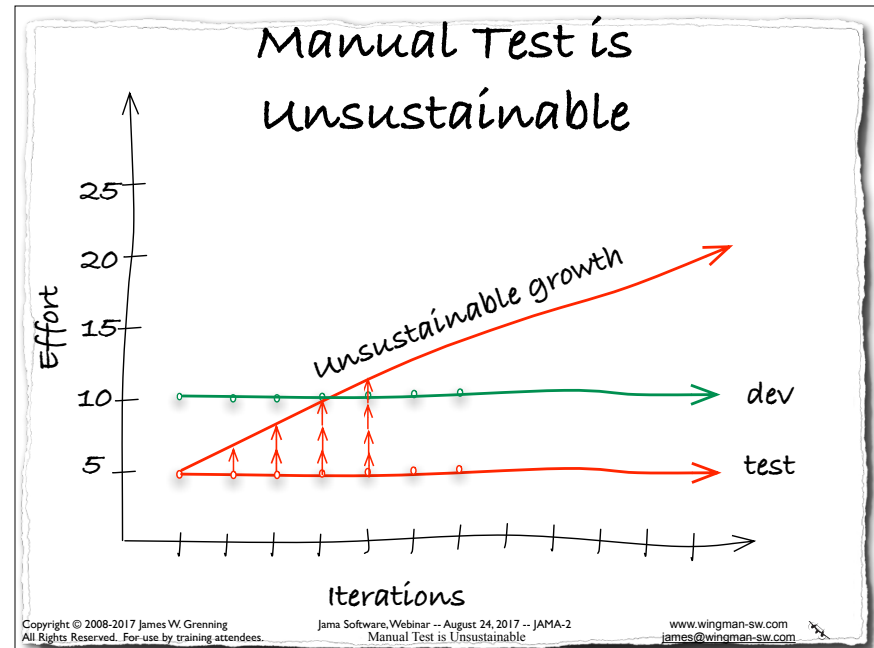
Because systems act up -- we have to be very careful.

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

Jama Software, Webinar -- August 24, 2017 -- JAMA-2  
Manual Test is Unsustainable

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

89



## Some of the Common Concerns

Debrief

Copyright © 2008-2017 James W. Grenning  
All Rights Reserved. For use by training attendees.

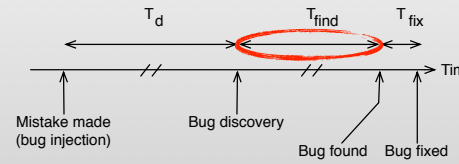
Jama Software, Webinar -- August 24, 2017 -- JAMA-2  
Test-Driven Development Debrief

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

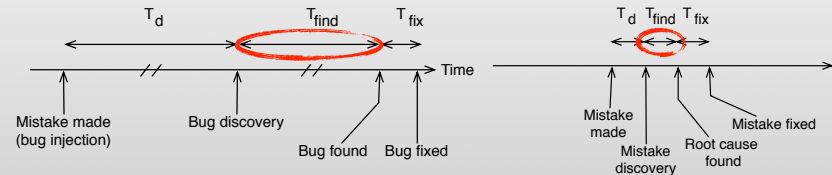
92

# TDD Will Take Too Long!

## The Physics of Debug Later Programming



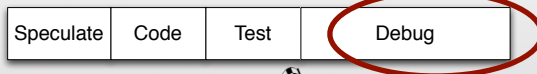
## The Physics of Test-Driven Development



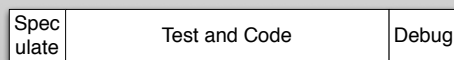
# Misconception and Sustainability

Time developers do not notice nor plan for

Traditional development



Test-driven development



# Too Many Lines of Code

## TDD Means Too Many Lines of Code? Think Again.

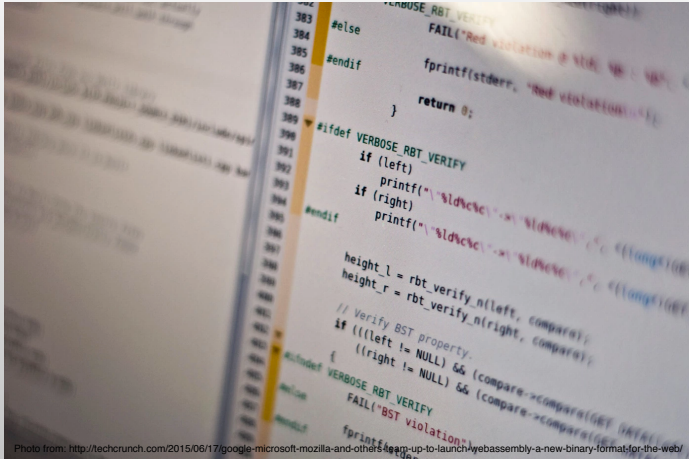
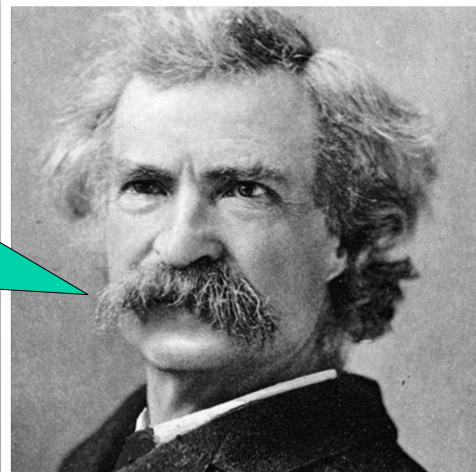


Photo from: <http://techcrunch.com/2015/06/17/google-microsoft-mozilla-and-other-teams-up-to-launch-yeassembly-a-new-binary-format-for-the-web/>

I already know how to code

## 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."

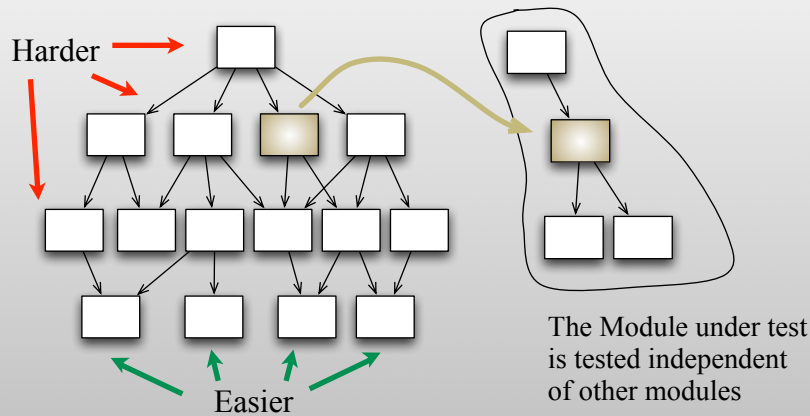


## TDD and the Code in the Middle

(most of the code!)

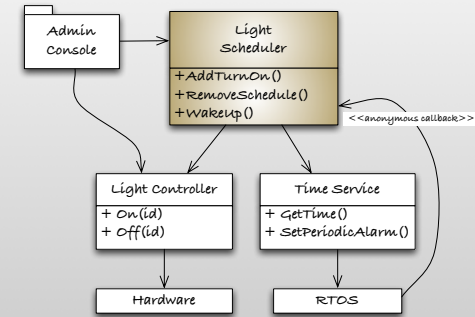


## Testing a Module in the Middle



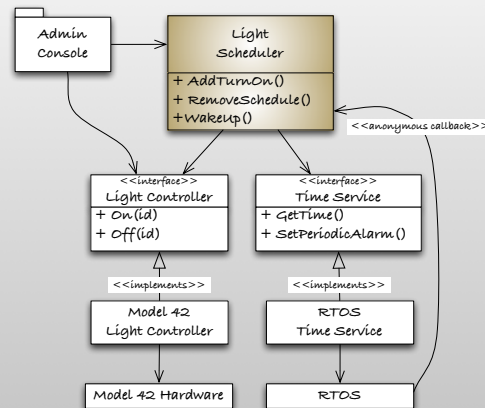
## Separation of Responsibilities

- 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.



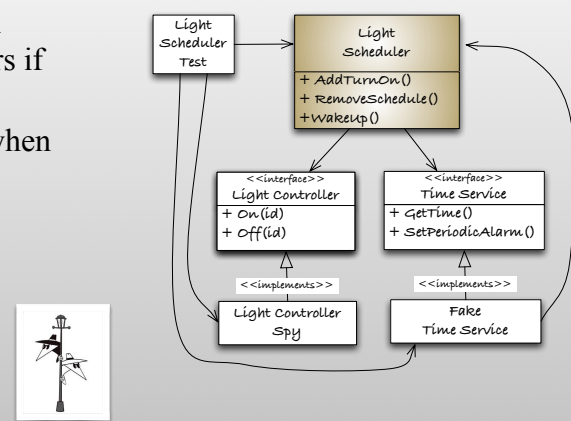
## Light Scheduler Design

- 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.



## 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.