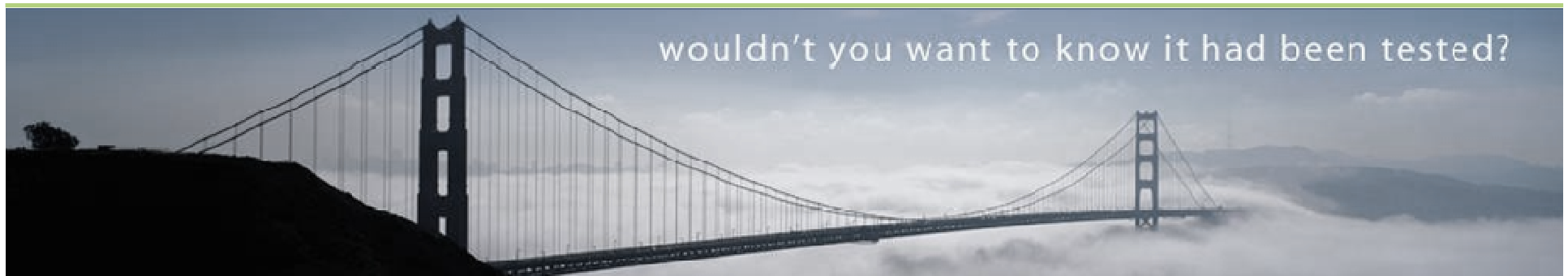


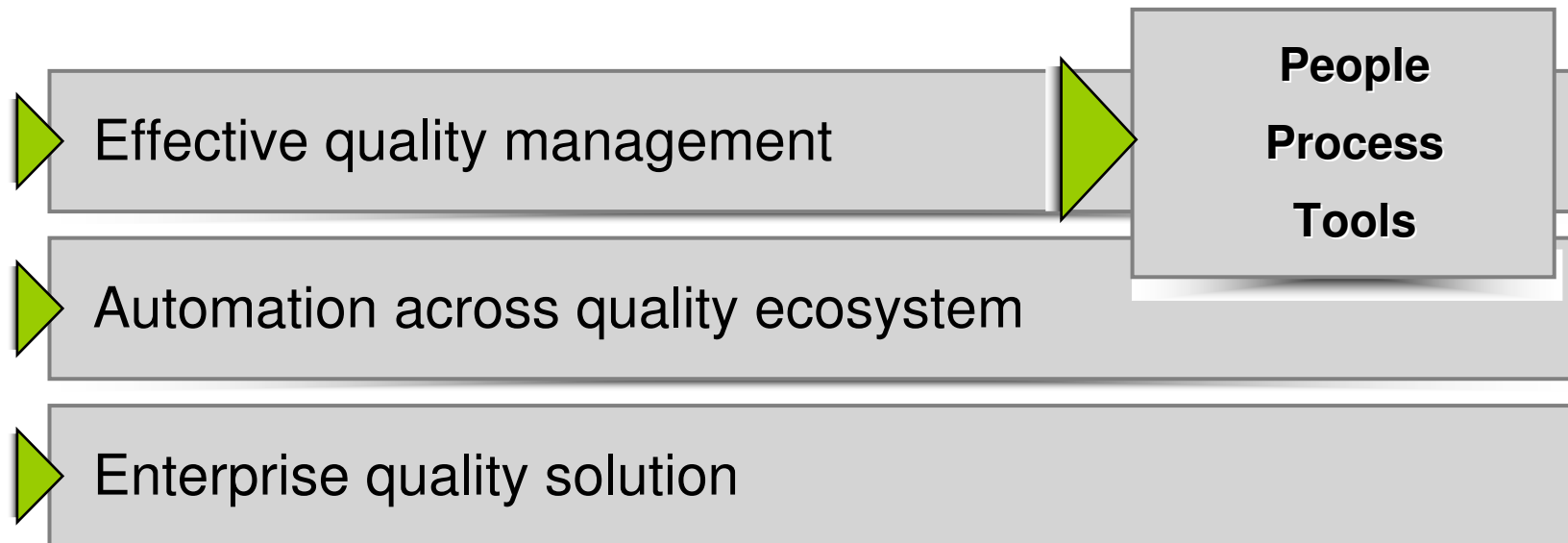
Unit Testing

Love It or Hate It,
You Should Be Doing It Automatically



Optimising Application Delivery

How Assurity Helps



MERCURY™

Agenda

- Unit Testing defined
- Value proposition
- Unit Testing best practices
- Case studies
- Automating unit testing
- Introducing Agitator
- Demo

What Is Unit Testing?

- Testing done by developers during code construction
- Usually called Unit testing, developer testing, testing of single classes, component testing, etc
- Most developers do some form already
 - ✓ Main routines, interactive debuggers, running the application, testing frameworks, etc
 - ✓ Need to do more and better

Why Unit Test?

- Reveals bugs that may escape “black box” testing
- Simplifies Integration
- Facilitates Change
- Documentation
- Encourages good design, such as separation of interface from implementation

Take a Better Approach

Test Bugs Out



Typical software development process

- QA is mainly responsible for quality
- Test late
- Use unverified components
- Deliver and fix defects later

VS.

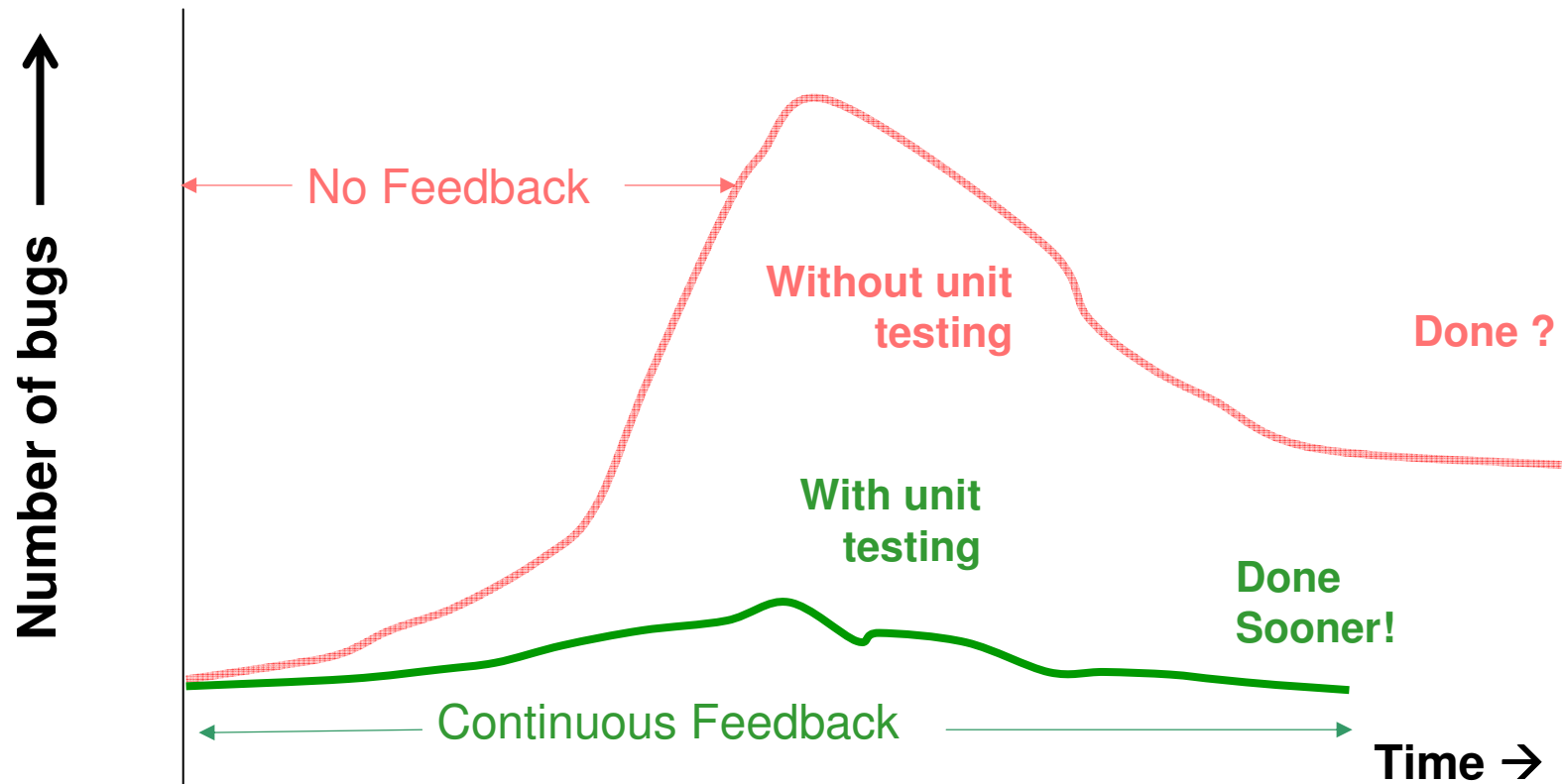
Build Quality In



Mature world-class manufacturing process

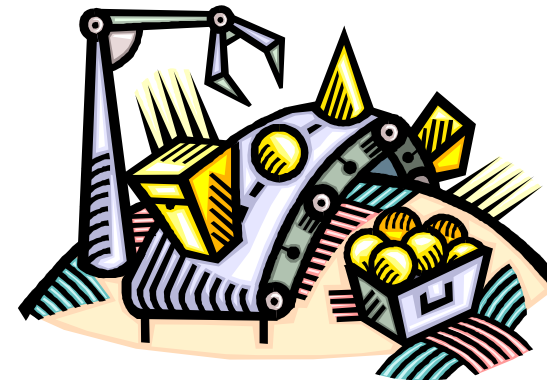
- Everyone is responsible for quality
- Test early and often
- Use verified components
- Stop the assembly line

Value of Unit Testing



Best Practices: Have a Process and Stick to It!

- Have a process
 - ✓ Any process is better than none
 - ✓ Make sure it is not “shelfware”
- ALL processes say you should unit test
 - ✓ (they’re just not that specific about how)
- Developers should feel the process is helping, not hindering them
- Managers should support the team and the process



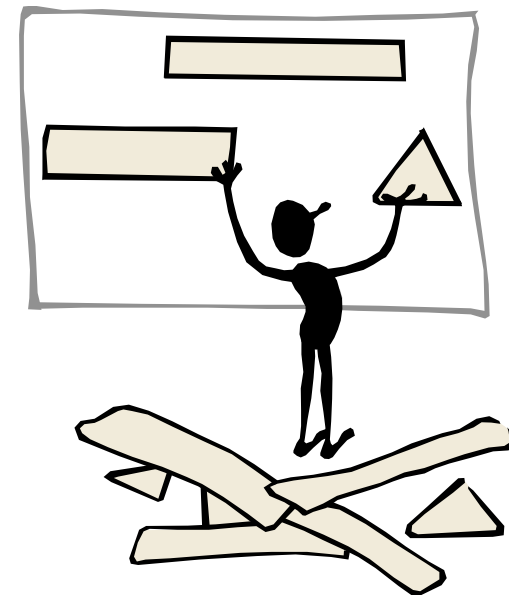
Best Practices: Test Early and Test Often

- Consider writing the tests first (Test-Driven Development)
- Start testing as soon as you start writing the code (if not before)
- Make sure the tests are run frequently
- The earlier you know about a problem the earlier you can fix it



Best Practices: Refactor, Refactor, Refactor

- Avoid complexity
 - ✓ Break large classes into smaller ones
 - ✓ Break large methods into smaller ones
- Simplifying the code makes it easier to understand and test
- Eases automation
- Example...
 - ✓ Servlet “doExecute” methods



Best Practices: Remember It Is a Group Effort

- Make your class more testable and usable by others
- Make test helpers that can be used by others to create test instances of your class
- Just because your class works doesn't mean you have made it easy for other to use
 - ✓ Document!
 - ✓ Throw appropriate exceptions
 - ✓ Make any messages meaningful



Best Practices: Use Mock Objects Effectively

- Mock Objects should:
 - 1) help make tests portable and easy to run, and
 - 2) help create assertions.
- Typically used for services such as logging, database, caching services, configuration managers etc.
- Implemented by jMock, EasyMock, StrutsTestCase, Agitator mocks, etc.
- In general, real classes ought to be invoked as much as is practical.

Best Practices: Use “Extreme Feedback”

- Try to fix bugs as soon as they are identified
- If not possible, make a plan for when to fix
- “Don’t live with Broken Windows”
- [Dashboard](#)
- [Lava Lamps](#)
- If you can’t measure you can’t manage it
- If you can’t manage it, you can’t improve it

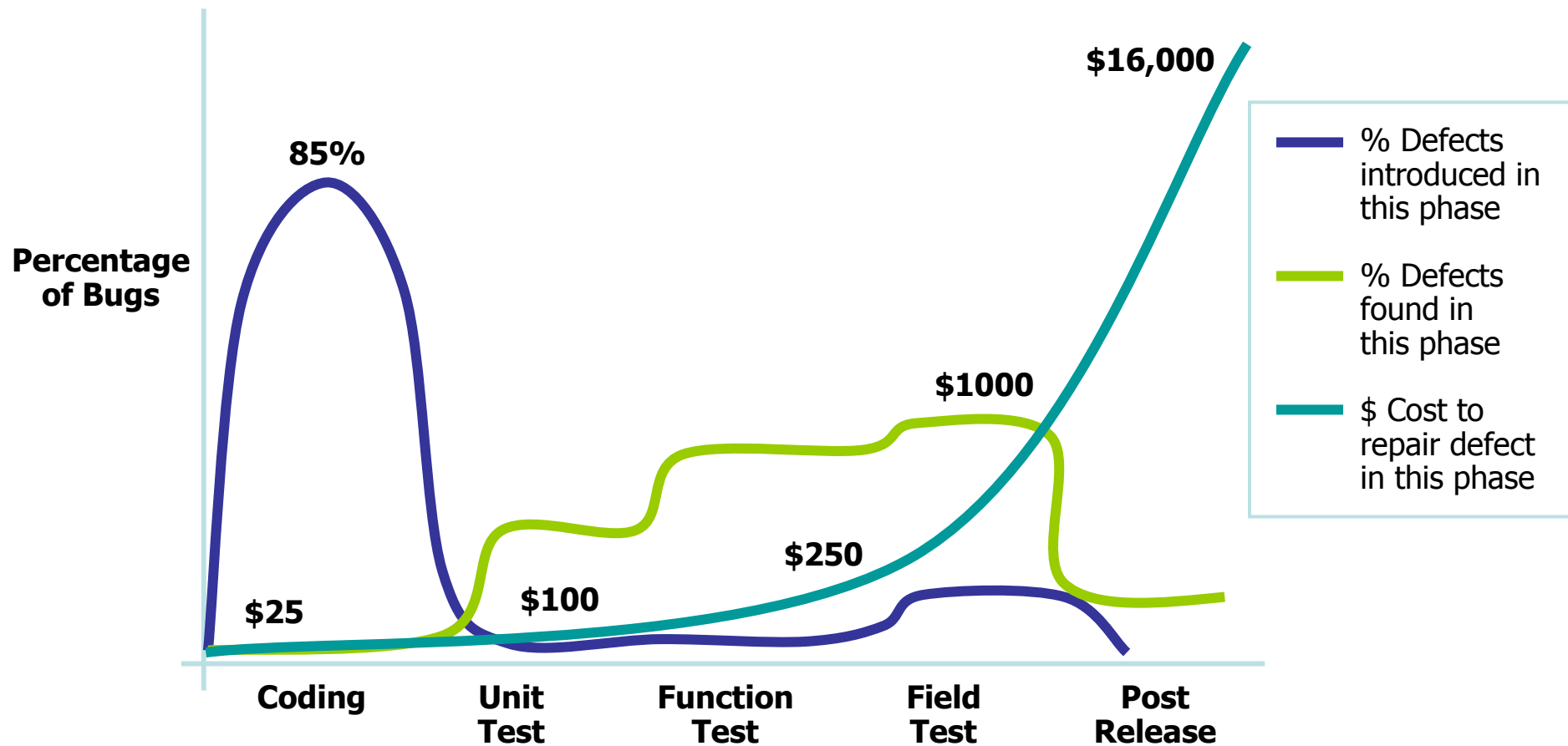
Case Studies: Industry Studies

- Industry data backs up the theory that finding bugs earlier is much cheaper
 - ✓ The Economic Impacts of Inadequate Infrastructure for Software Testing – the “NIST” report
 - ✓ Finding a bug in System Testing is 90 times more expensive to fix than finding it the requirements
 - ✓ Library of references:
 - www.agitar.com/downloads
 - [“Software Quality Articles and References”](#)

Life Cycle Stage	Baziuk (1995) Study Costs to Repair when Found	Boehm (1976) Study Costs to Repair when Found
Requirements	1X	0.2Y
Design		0.5Y
Coding		1.2Y
Unit Testing		
Integration Testing		
System Testing	90X	5Y
Installation Testing	90X-440X	15Y
Acceptance Testing	440X	
Operation and Maintenance	470X-880X	

Table 1-5. Relative Costs to Repair Defects when Found at Different Stages of the Life-Cycle

Case Studies: It Pays to Find Bugs in Development



Source: Applied Software Measurement, Capers Jones, 1996

Case Studies: More Evidence of the Benefits

A JavaPOS software development project in the IBM Retail Store Solutions group experienced a 50% reduction in defects identified during functional verification testing

Source: Michael Maximilien and Laurie Williams,
“Assessing Test-Driven Development at IBM,”

http://collaboration.csc.ncsu.edu/laurie/Papers/MAXIMILIEN_WILLIAMS.PDF

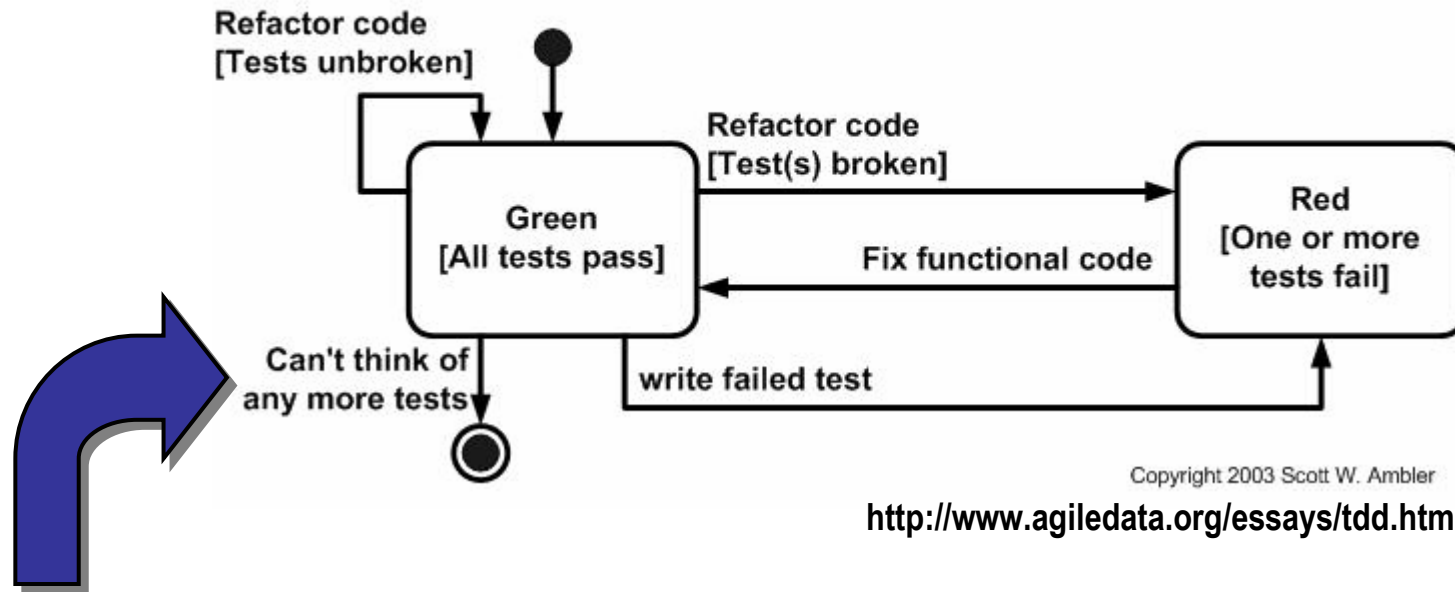
Case Studies: Manual Unit Testing Successes

- Many organizations have been successful with JUnit – particularly when used as part of a Test Driven methodology
 - ✓ Kent Beck, Martin Fowler, many others...
 - ✓ ThoughtWorks
 - ✓ TDD, TFD, FDD, etc...
- It takes skill and commitment to succeed in the long term
- You can still overlook things...



Case Studies : Manual Unit Testing – Cautionary Tale

Figure 2. Testing via the xUnit Framework.



- “Can’t think of any more tests?”
- You only write tests for the things you thought might be a problem

Case Studies: Faults of Omission

- Jet Fighter test story

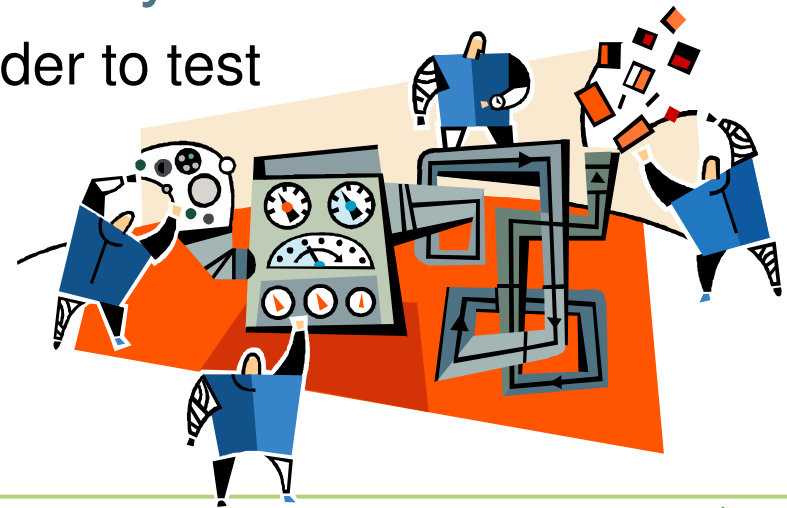
Pseudo code:

```
while (true) {  
    command = getCommand();  
    switch(command) {  
        case RAISE_LANDING_GEAR:  
            if (isLandingGearUp()) {  
                raiseLandingGear();  
            }  
        case ....  
    }
```

- In a study “22% to 54% of faults were omissions”
(<http://www.testing.com/writings/omissions.html>)

Intelligently Automating Test Creation Increases the Benefits

- Exploratory testing with all sorts of valid and invalid inputs
- Discovers subtle bugs outside of the “happy path”, eg.
 - ✓ Data values that were not considered
 - ✓ Sequences that were not anticipated
- Helps you think about the design of your code
 - ✓ Discovers which elements are harder to test (and probably harder to maintain)
- Makes tests easier to maintain
- Helps you become a better coder and unit tester



Quotes

- **Kent Beck** – “Kevin Lawrence ... used Agitator to find a bug in the JUnit Money example code that had sat there undiscovered for seven years.

I ran the tests for Spider through the Dashboard and learned about holes in my testing technique--that was the moment that convinced me to join Agitar (I work there one day a week). “

- **J. B. Rainsberger** – “This is exactly how I expect to use Agitator: to help me find the tests I missed. When the day comes that I'm on a full-time Java project (if that day ever comes again), I will insist on Agitator.”

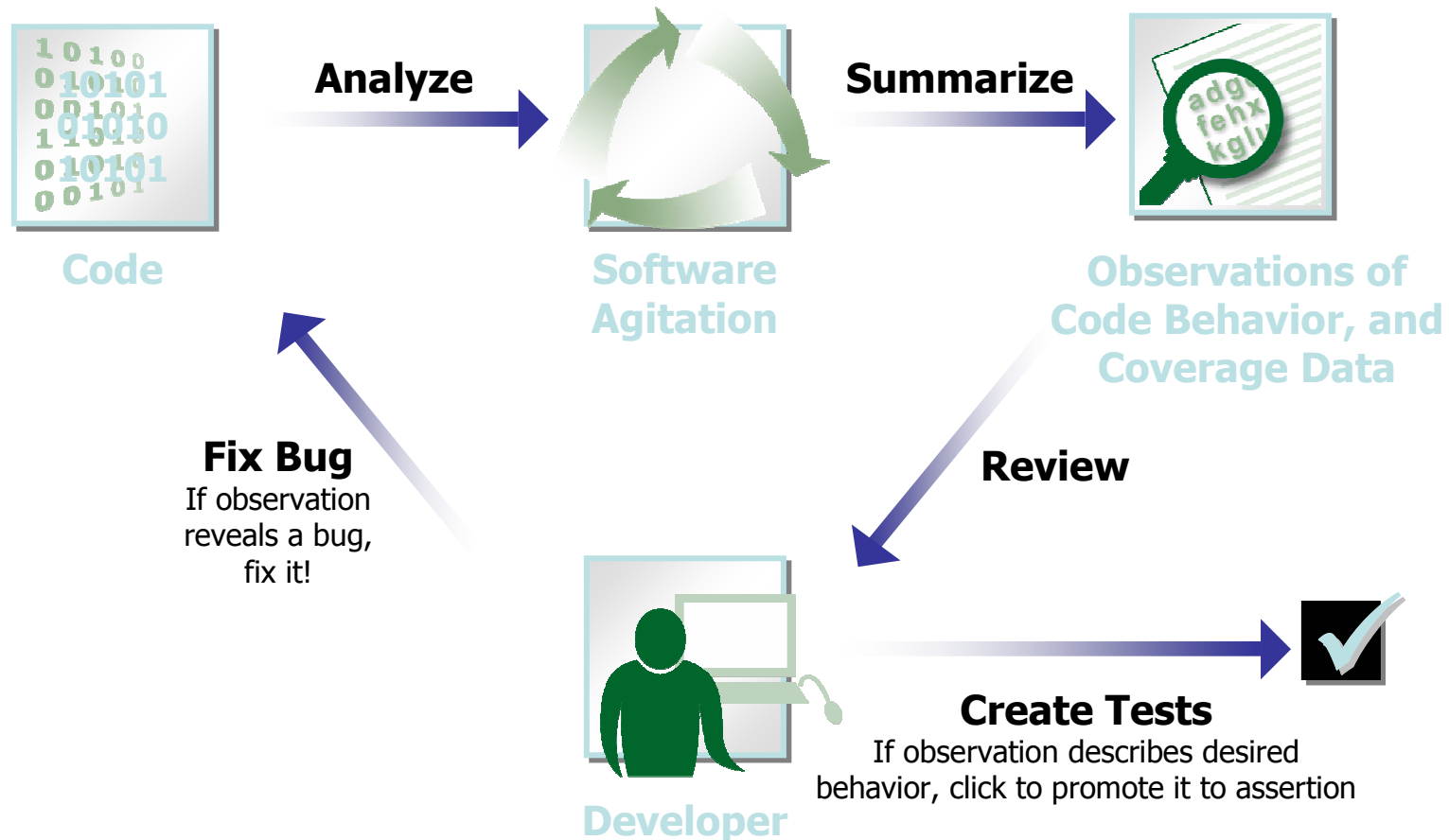
Case Studies:

abeBooks - Resulting Benefits

- Lower defects for committed code
- Quicker turn around time for maintenance work and defect fixes
- More maintainable code, better structure within the architecture for even the small things
- QA became more efficient, testing what wasn't already tested
- Improved communication between developers and QA team

...first 100% uptime quarter

Software Agitation Validates Intended Code Behavior



Demo



Demo - Bowling Game

- 1 game = 10 frames

- Standard Frame

2	5
7	

- Spare Frame

8	/	4	
14			

- Strike Frame

	X	8	1
19		28	

- Foul

7	F
7	

Summary:

Unit Testing with JUnit and Agitator

- JUnit is great for:
 - ✓ helping you design using TDD
 - ✓ crafting very specific test cases
- Agitator is great for:
 - ✓ exploratory testing - with all sorts of valid and invalid inputs
 - ✓ helping you think about corner cases and omissions
 - ✓ discovering and enforcing class and method invariants
- JUnit + Agitator combines all of these benefits
- Agitar Management Dashboard combines JUnit and Agitator results

Thank you

Nigel Charman

nigel.charman@assurity.co.nz



www.assurity.co.nz

www.junit.org

www.agitar.com

www.developertesting.com

