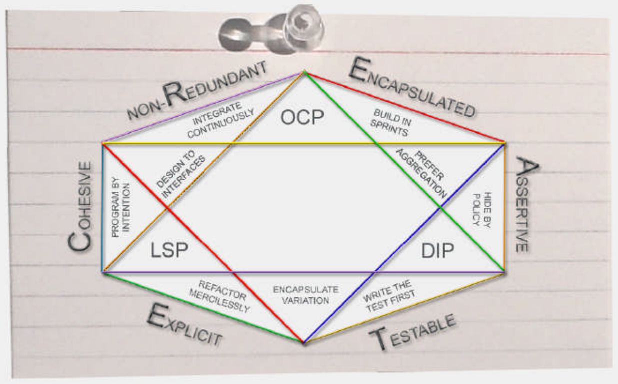
to be agile

Overcoming Test-Driven Damage

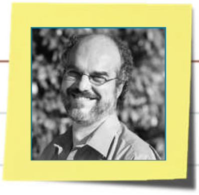


And Effective Ways to Think About TDD

http://ToBeAgile.com
info@ToBeAgile.com

DB20180111

David Scott Bernstein



- Software developer since 1980
- Trained 8,000 developers since 1990
- Published author since 2015
- Website: http://ToBeAgile.com

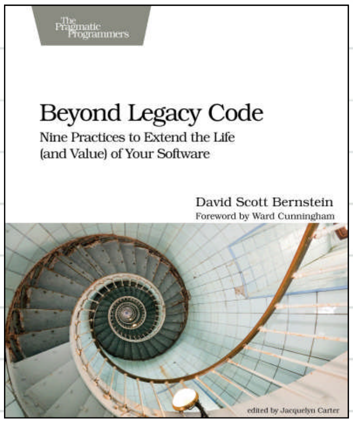
to be agile

2

My Book – Beyond Legacy Code

to be agile
lessen the learning curve

http://ToBeAgile.com
info@ToBeAgile.com


The Pragmatic Programmers

Beyond Legacy Code
Nine Practices to Extend the Life (and Value) of Your Software

David Scott Bernstein
Foreword by Ward Cunningham

edited by Jacquelyn Carter

http://BeyondLegacyCode.com

to be agile

DB20160109 3

Nine Essential Practices

1. **Say What, Why, and for Whom before How**: With a Product Owner defining the next most important features to build, the need for upfront requirements goes away.
2. **Build in Small Batches**: Building incrementally increases feedback, helps simplify the construction of complex systems, and reduces risks.
3. **Integrate Continuously**: Sets up the infrastructure for incremental development.
4. **Collaborate**: Spiking, pairing, and swarming as a team to solve problems and radiate knowledge throughout an organization.
5. **Create CLEAN Code**: Share standards and practices for building software with code qualities that support testability.
6. **Write the Test First**: Drops the cost of building and maintaining software dramatically.
7. **Specify Behaviors with Tests**: Uses tests to define and document behaviors.
8. **Implement the Design Last**: Paying technical debt can pay back dividends in the short term as well as the long term.
9. **Refactor Legacy Code**: Incorporate learning and pay off technical debt.

to be agile

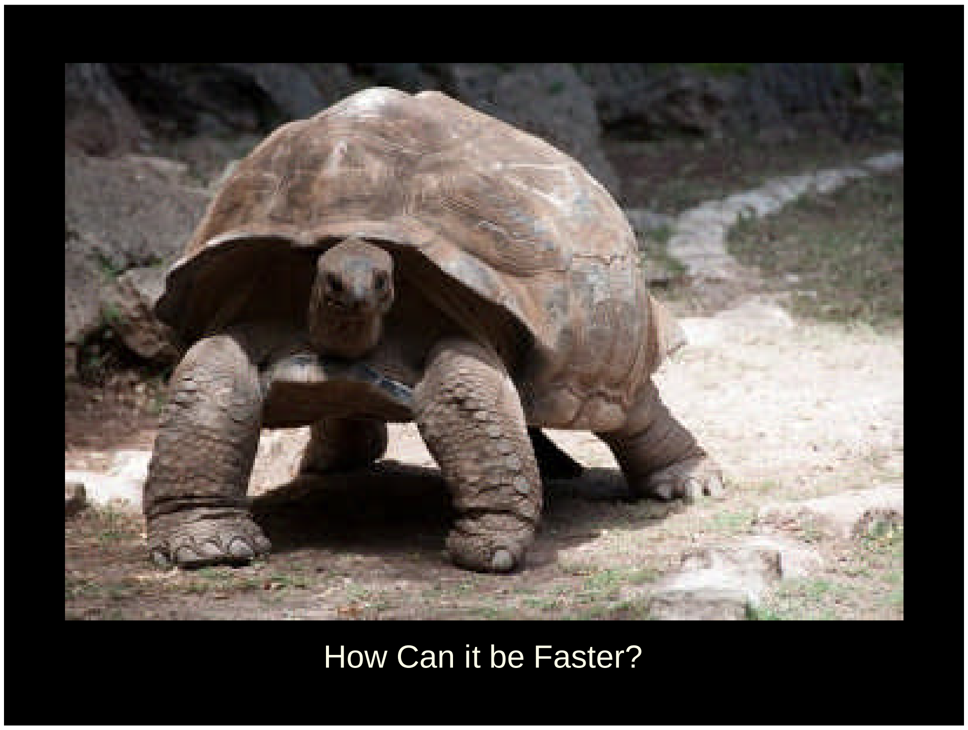
4

Test First Development

to be agile

5

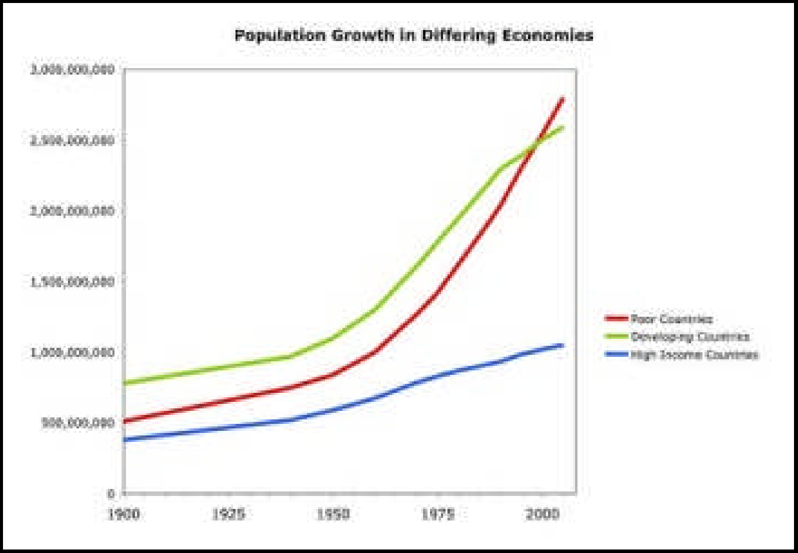
Test First or Test Last?



How Can it be Faster?

What We Stop Doing

Test First



Slow Then Faster

Benefits of TDD

Why TDD Works

TDD Can Fail

create

Code Quality and Tests

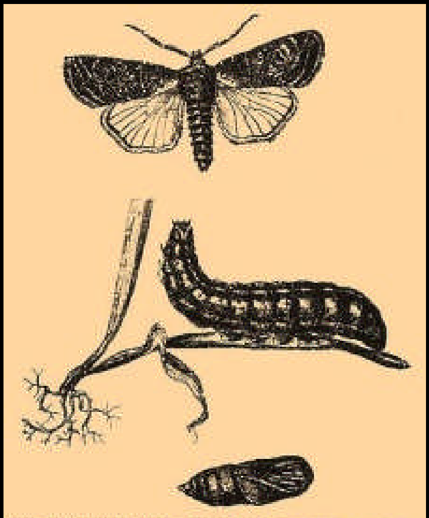
Another Client Told Me

Fast Tests

TDD Does Not Replace QA

What is a Test?

What a Test Tests



Фиг. 2. Озимая совка въ стадіяхъ гусеницы, куколки и бабочки

It Becomes a Test

The Dual Role

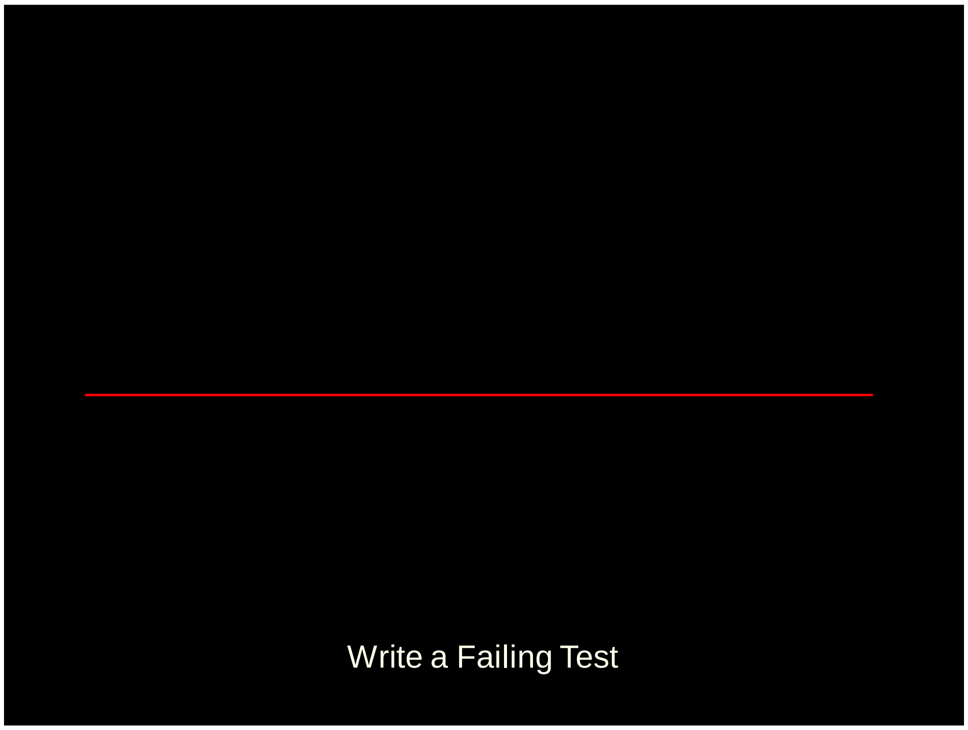
Drive Development with Tests

TDD Metaphors

An Introduction

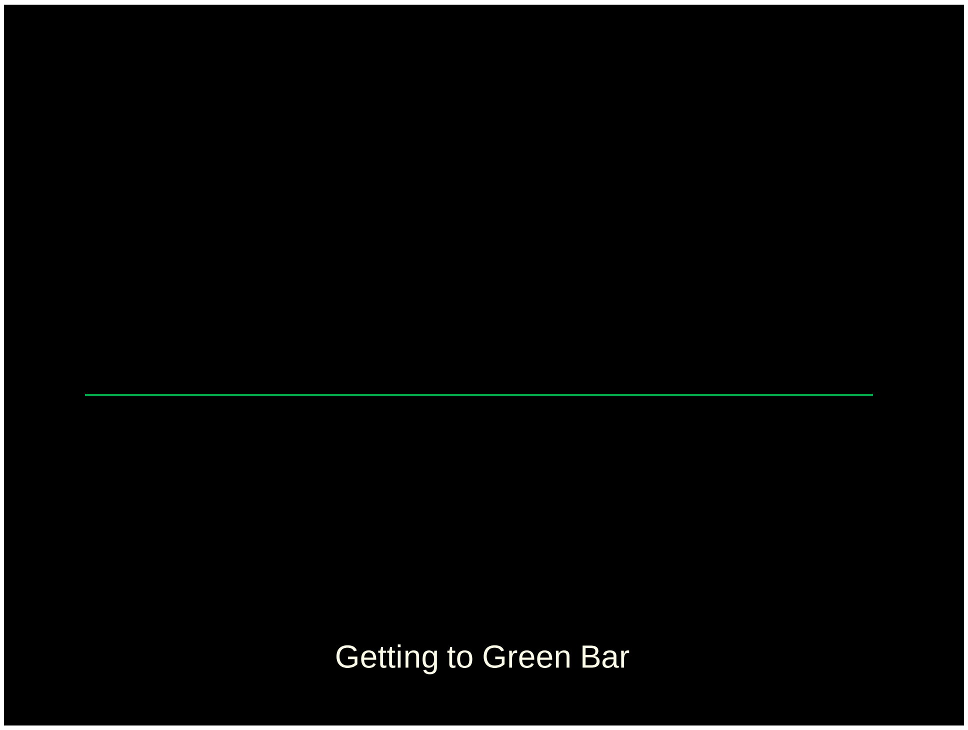
Three Steps in Test First

Red Bar, Green Bars



Write a Failing Test

Celebrate the Red Bar



Getting to Green Bar

Being Green

Refactor Code

Refactor Tests

How Many is Enough?

On the Three Steps

For Example

1 + 1 = 2

Java/C#

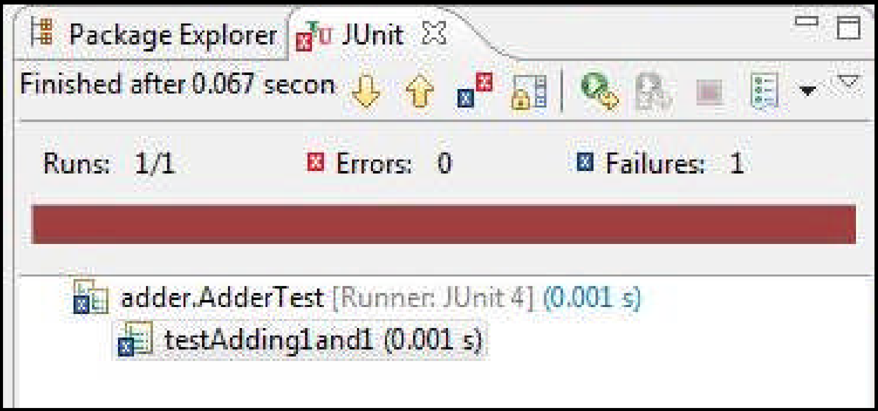
- Suppose I want to write an adder class
- I'd start by writing a failing test

```
Adder adder = new Adder();
assertEquals("1+1=2", 2, adder.add(1,1), .1);
```

- I'd then stub it out so it compiles

```
public class Adder {
    public int add(p1, p2) {
        return 0;
    }
}
```

to be agile



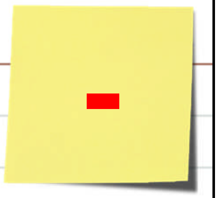
```
Finished after 0.067 secon

Runs:  1/1        Errors:  0        Failures:  1

adder.AdderTest [Runner: JUnit 4] (0.001 s)
    testAdding1and1 (0.001 s)
```

Red Bar

A Mad Dash



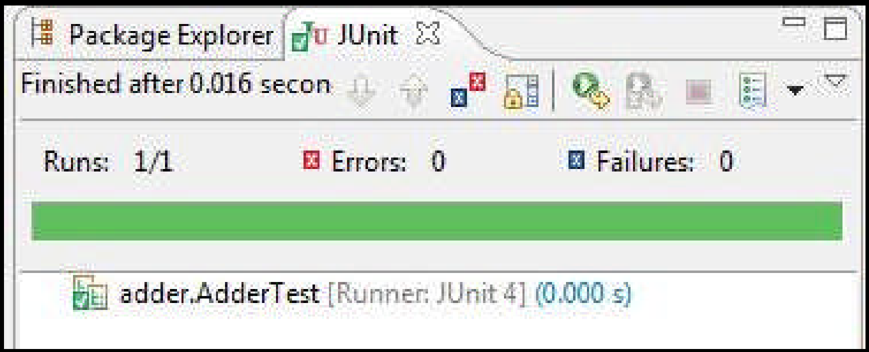
Java/C#

- What's the fastest way to get to the green bar?
  - Return 2

```
public class Adder {
    public int add(p1, p2) {
        return 2;
    }
}
```

to be agile

37



Package Explorer JUnit

Finished after 0.016 secon

Runs: 1/1 Errors: 0 Failures: 0

adder.AdderTest [Runner: JUnit 4] (0.000 s)

Green Bar

Let’s Try Another

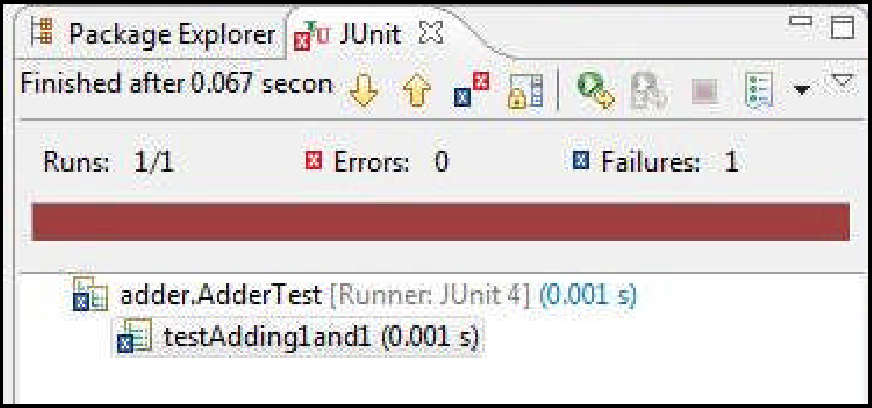
Java/C#

assertEquals(“2+2=4”, 4, adder.add(2,2), .1);

- And what happens?

to be agile

39

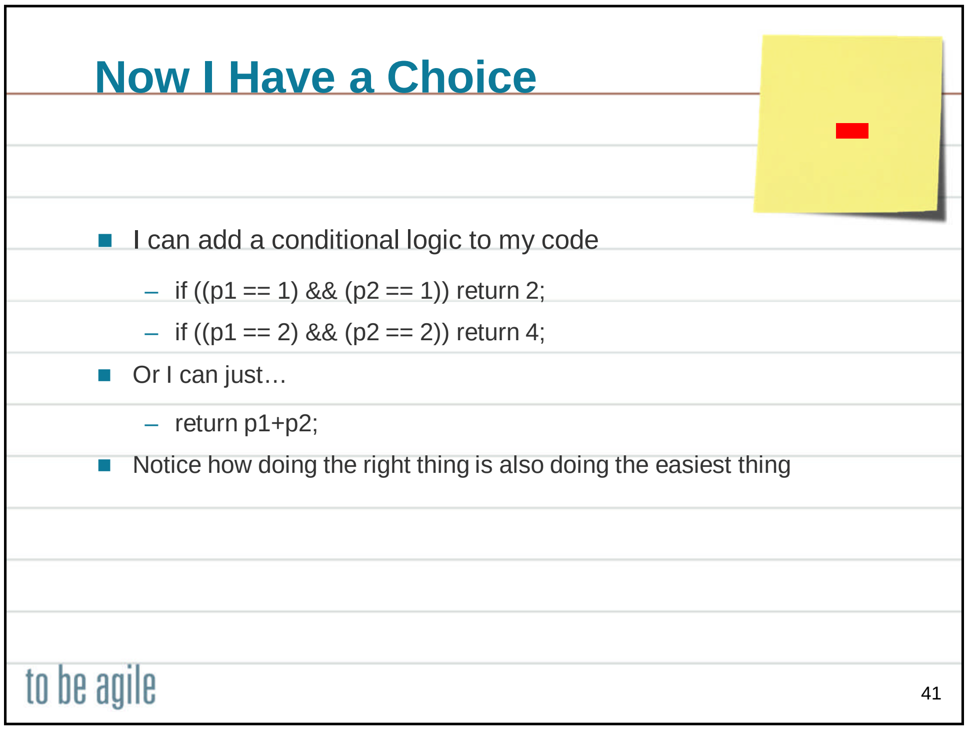

Package Explorer | JUnit

Finished after 0.067 secon

Runs: 1/1   Errors: 0   Failures: 1

adder.AdderTest [Runner: JUnit 4] (0.001 s)
testAdding1and1 (0.001 s)

Red Bar

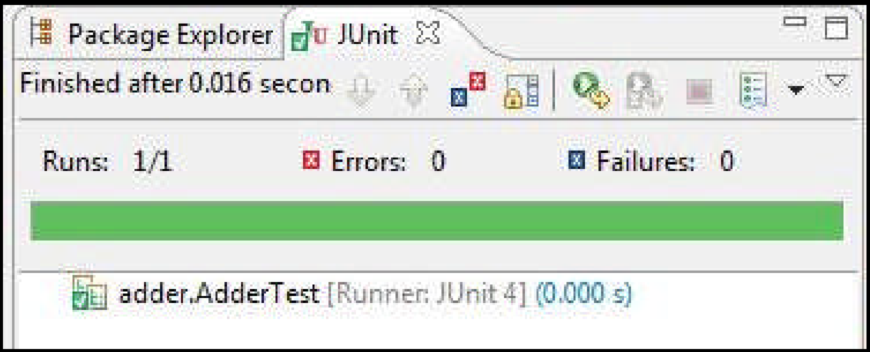


Now I Have a Choice

- I can add a conditional logic to my code
  - if ((p1 == 1) && (p2 == 1)) return 2;
  - if ((p1 == 2) && (p2 == 2)) return 4;
- Or I can just…
  - return p1+p2;
- Notice how doing the right thing is also doing the easiest thing

to be agile

41

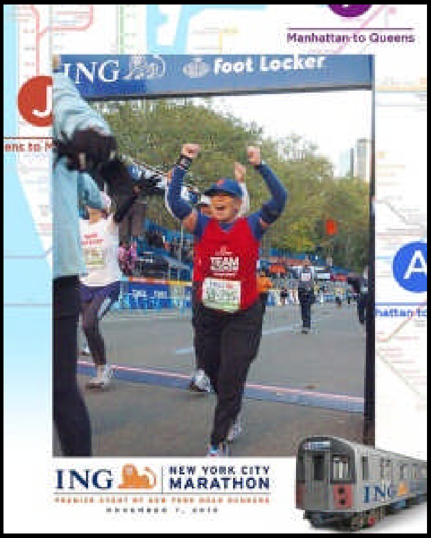


Finished after 0.016 secon

Runs: 1/1 Errors: 0 Failures: 0

adder.AdderTest [Runner: JUnit 4] (0.000 s)

Green Bar



Test as Design

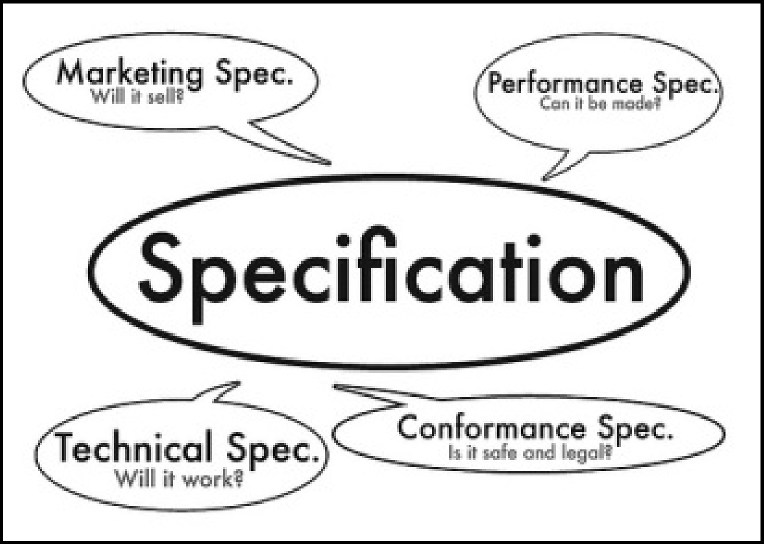
3

Uncle Bob’s Laws of TDD

What Makes a Good Test?

Characteristics of a Good Test

What is a Unit?



Test Semantics

Instrumentation

Instead of Doing This…

```
@Test
    public void testConstructor() {

    User user = new User("Clark", "Kent", "user@example.com",
              "Superman", "kryptonite");
         assertEquals("Clark", user.firstName());
         assertEquals("Kent", user.lastName());
         assertEquals("user@example.com", user.eMail());
         assertEquals("Superman", user.userName());
         assertEquals("kryptonite", user.password());
    }
```

to be agile

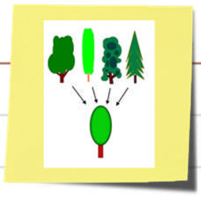
50

Do This

```
 @Test
     public void testRetrievingParametersAfterConstruction() {
           private static final String firstName = "Clark";
           private static final String lastName = "Kent";
           private static final String eMail = "user@example.com";
           private static final String userName = "Superman";
           private static final String password = "kryptonite";

           User user = new User(firstName, lastName, eMail, userName,
password);
           assertEquals(firstName, user.firstName());
           assertEquals(lastName, user.lastName());
           assertEquals(eMail, user.eMail());
           assertEquals(userName, user.userName());
           assertEquals(password, user.password());
     }
```

Generalizations



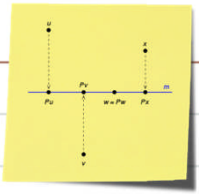
- Instrumentation can also specify generalizations:

```
public void testAddition() {
          private int anyInt = 1;
          private int theResult = 2;

          Adder adder = new Adder();
          assertEquals(theResult, adder.add(anyInt, anyInt));
     }
```

Instrumentation Benefits

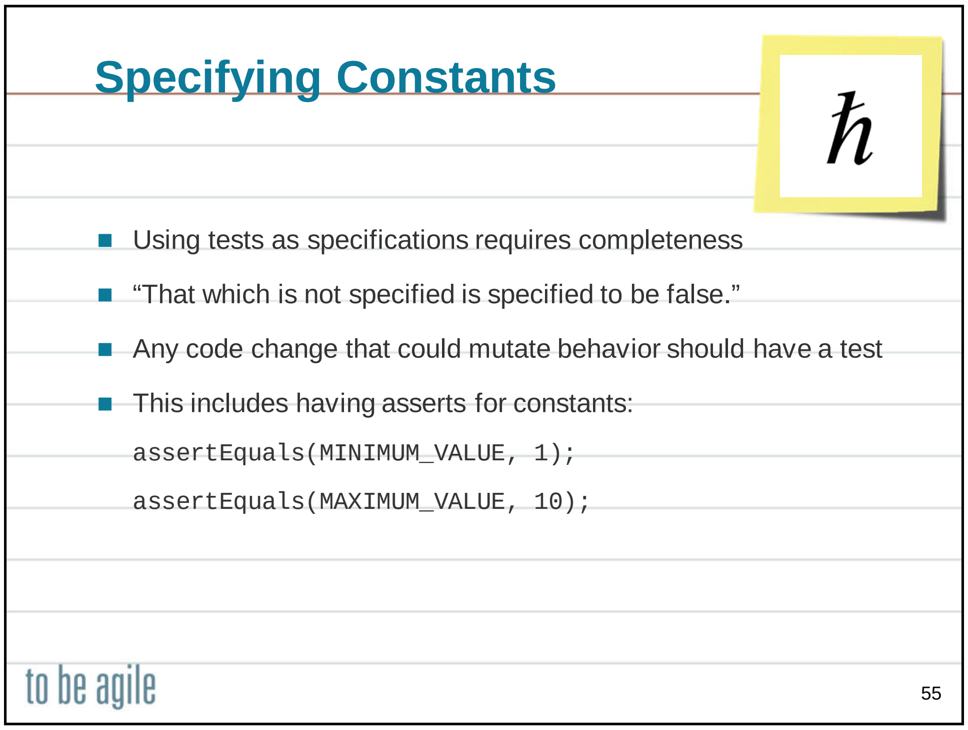
Defining a Linear Range



- How many assertions do you need to specify a linear range?
- 4… or 3
- For example, to validate value is within a range from MINIMUM_VALUE to MAXIMUM_VALUE:

```
assertEquals(value, MINIMUM_VALUE – 1); // exception
assertEquals(value, MINIMUM_VALUE);     // valid
assertEquals(value, MAXIMUM_VALUE);     // superfluous?
assertEquals(value, MAXIMUM_VALUE + 1); // exception
```

to be agile



Specifying Constants

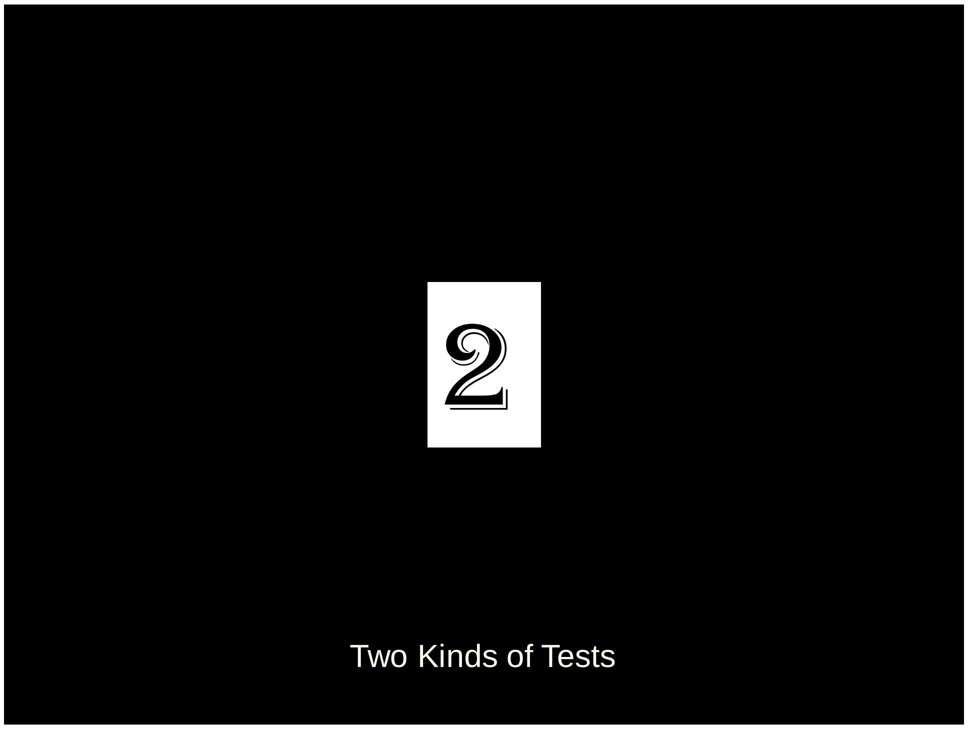
ℏ

- Using tests as specifications requires completeness
- “That which is not specified is specified to be false.”
- Any code change that could mutate behavior should have a test
- This includes having asserts for constants:

```
assertEquals(MINIMUM_VALUE, 1);
assertEquals(MAXIMUM_VALUE, 10);
```

to be agile

55

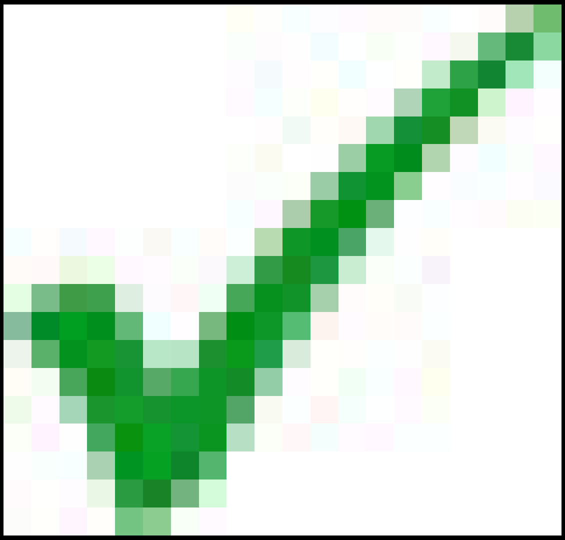


2

Two Kinds of Tests

Boundary Testing

Workflow Testing


Testable Code

Don’t “Test Until Bored”

TDD is Not Enough

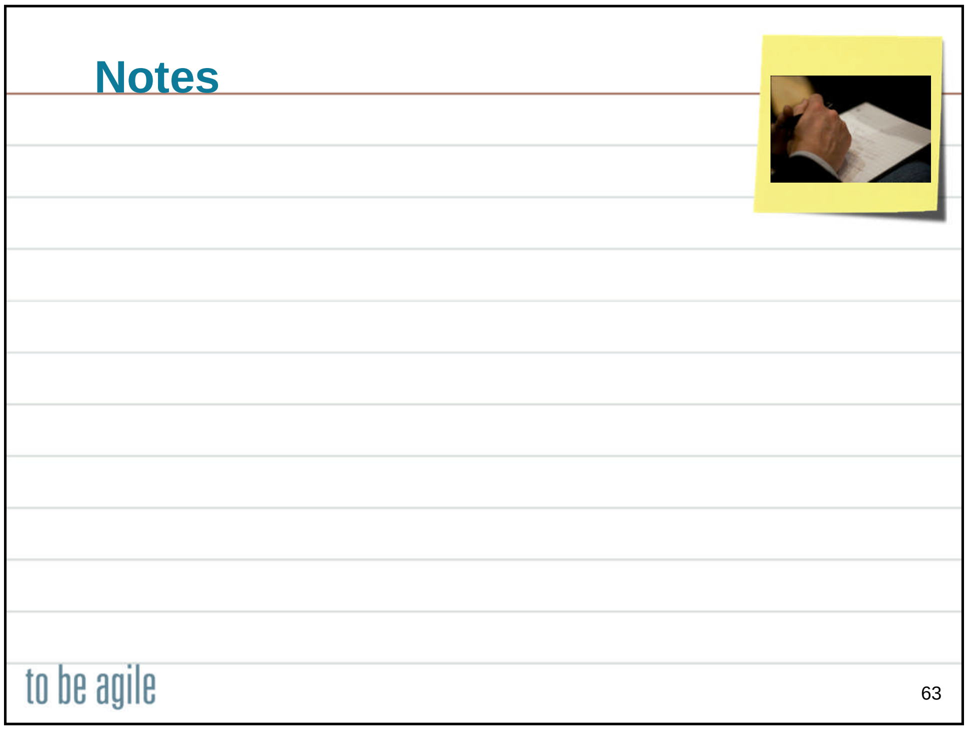
Thank You!

Please fill out your feedback forms!

- We have just scratched the surface, to learn more:
  - Read my blog: http://ToBeAgile.com/blog
  - Sign up for my newsletter: http://ToBeAgile.com/signup
  - Follow me on Twitter (@ToBeAgile)
  - Read my book:
    - *Beyond Legacy Code: Nine Practices to Extend the Life (and Value) of Your Software* (available from http://BeyondLegacyCode.com)
  - Attend my one of my Certified Scrum Developer trainings
    - See http://ToBeAgile.com/training for my public class schedule
    - Or contact me to arrange a private class for your organization
  - Visit http://ToBeAgile.com for more information

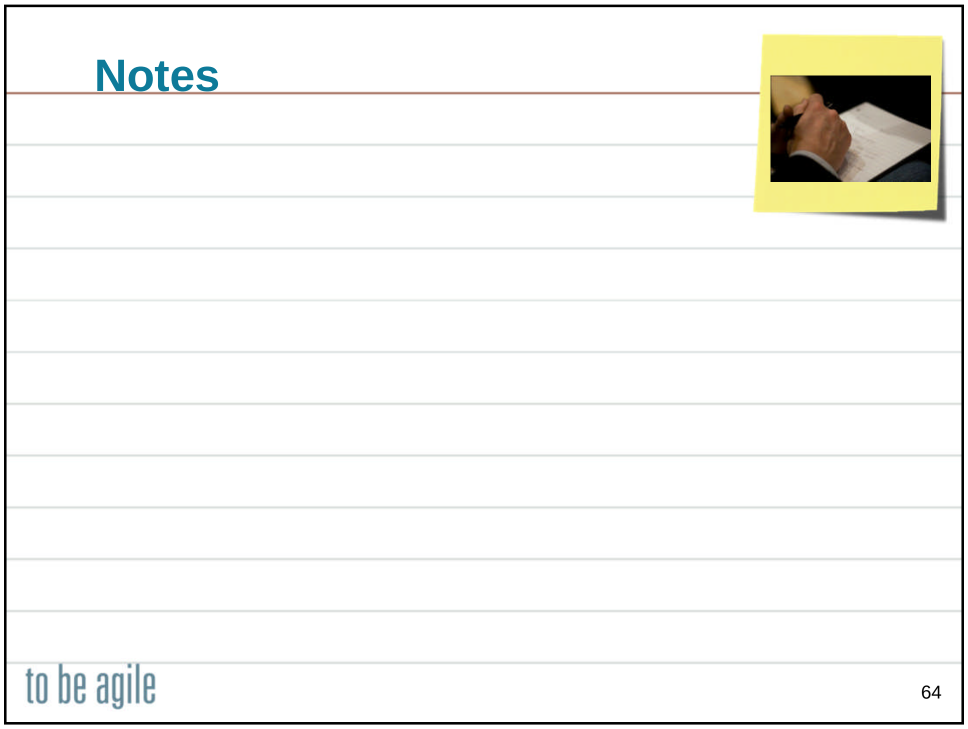
to be agile

62



Notes

63



Notes

64