Defense, Space & Security
Lean-Agile Software

From Details to Done

A Test-Driven Approach to Software Development

Steve Jewett
Systems & Software Technology Conference 2011

BOEING is a trademark of Boeing Management Company.
Copyright © 2011 Boeing. All rights reserved.

Topics

- **Moving Tests Forward**
- **3 Rules of Test-Driven Development (TDD)**
- **TDD in Unit, Integration and Acceptance Testing**
- **Comprehensive TDD Process**
- **Pros and Cons of TDD**
- **Q&A**

Copyright © 2011 Boeing. All rights reserved.

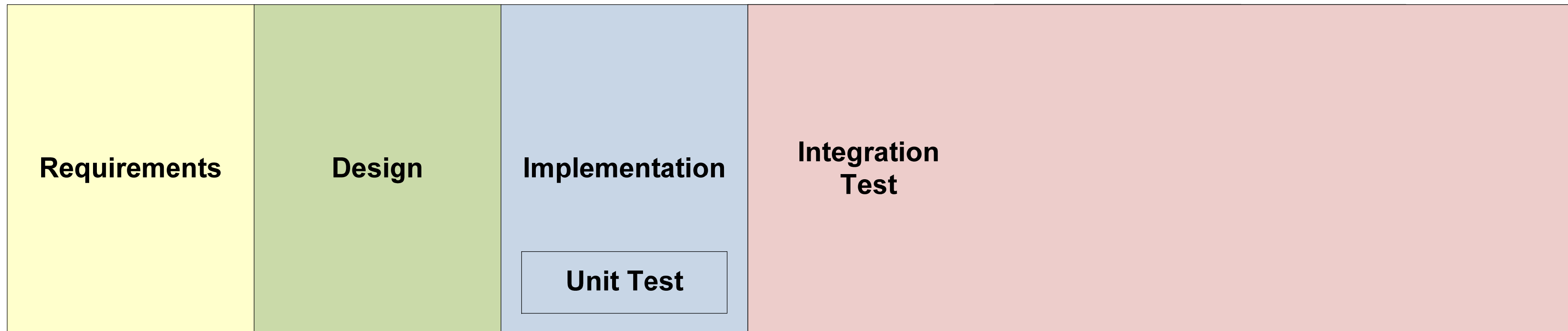
Traditional Development Cycle

Testing Follows Implementation:

Unit tests are executed after modules are completed.

Integration testing follows implementation.

Acceptance testing begins at the end of integration.



Copyright © 2011 Boeing. All rights reserved.

Moving Tests Forward

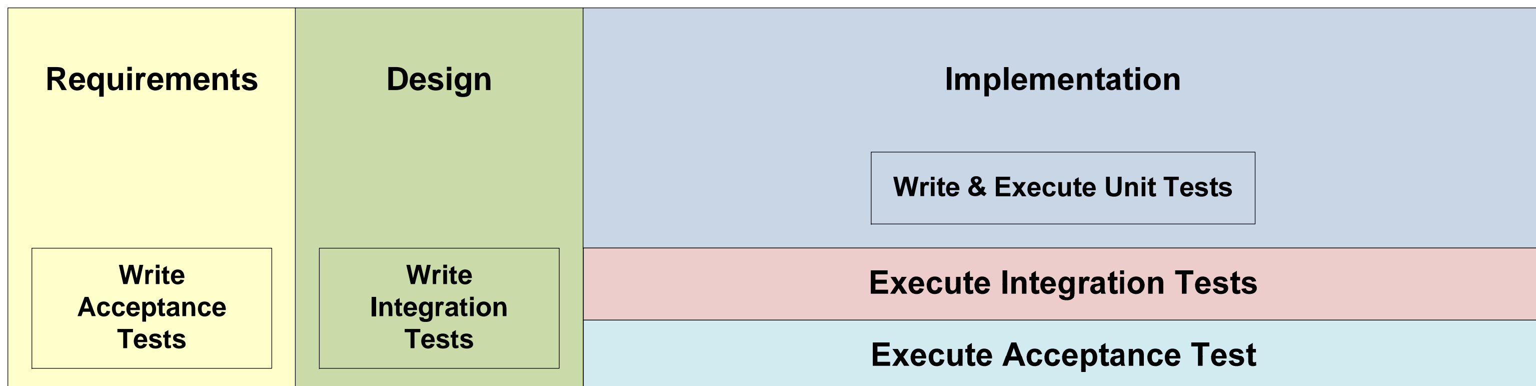
Testing Occurs Before Implementation:

Acceptance tests are developed as part of the requirements.

Integration tests are developed as part of the design.

Unit tests are developed as part of the implementation.

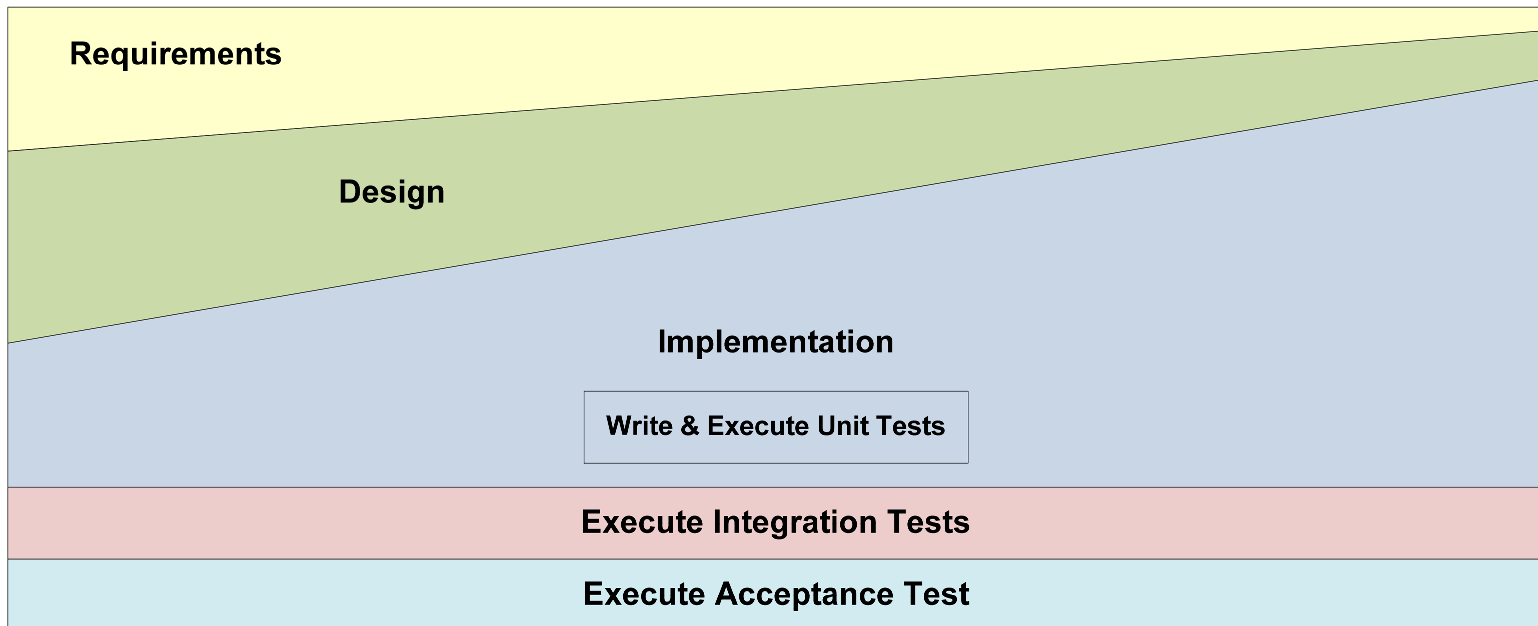
Test are executed throughout implementation; test failures drive what to do next.



Copyright © 2011 Boeing. All rights reserved.

Testing in an Agile Development Cycle

Agile Development is not phase-oriented, so tests are executed throughout the cycle, not just during implementation.



Copyright © 2011 Boeing. All rights reserved.

How Does Testing Drive Development?

Test-Driven Development (TDD) says to create tests first and let them drive implementation. The three rules of TDD demonstrate how to do that.

3 Rules of TDD

1. Do work ~~Write production code~~ only to pass a failing test.
2. Do ~~Write~~ only enough testing ~~test code~~ to fail.
3. Do ~~Write~~ only enough work ~~production code~~ to pass.

Copyright © 2011 Boeing. All rights reserved.

TDD at the Unit Test Level

Unit tests are created by developers to add functionality to a class or module.

At the unit test level the three rules are manifest in the "red-green-refactor" approach:

Red-Green-Refactor

Write a unit test that fails.

Write production code to make the test pass.

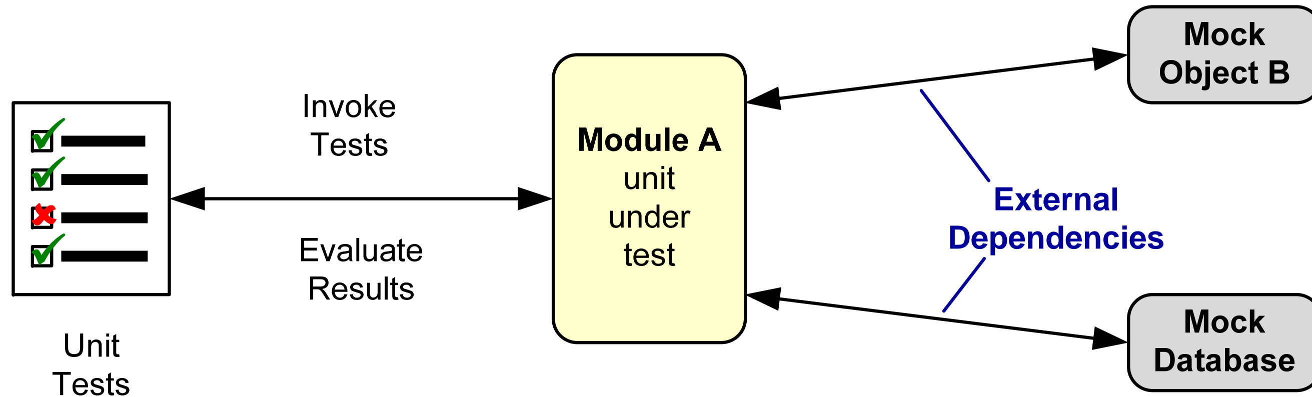
Clean up both test and production code.

for example …

Copyright © 2011 Boeing. All rights reserved.

TDD at the Unit Test Level (cont'd)

Using the Red-Green-Refactor approach, developers create unit tests for individual modules as they add functionality.

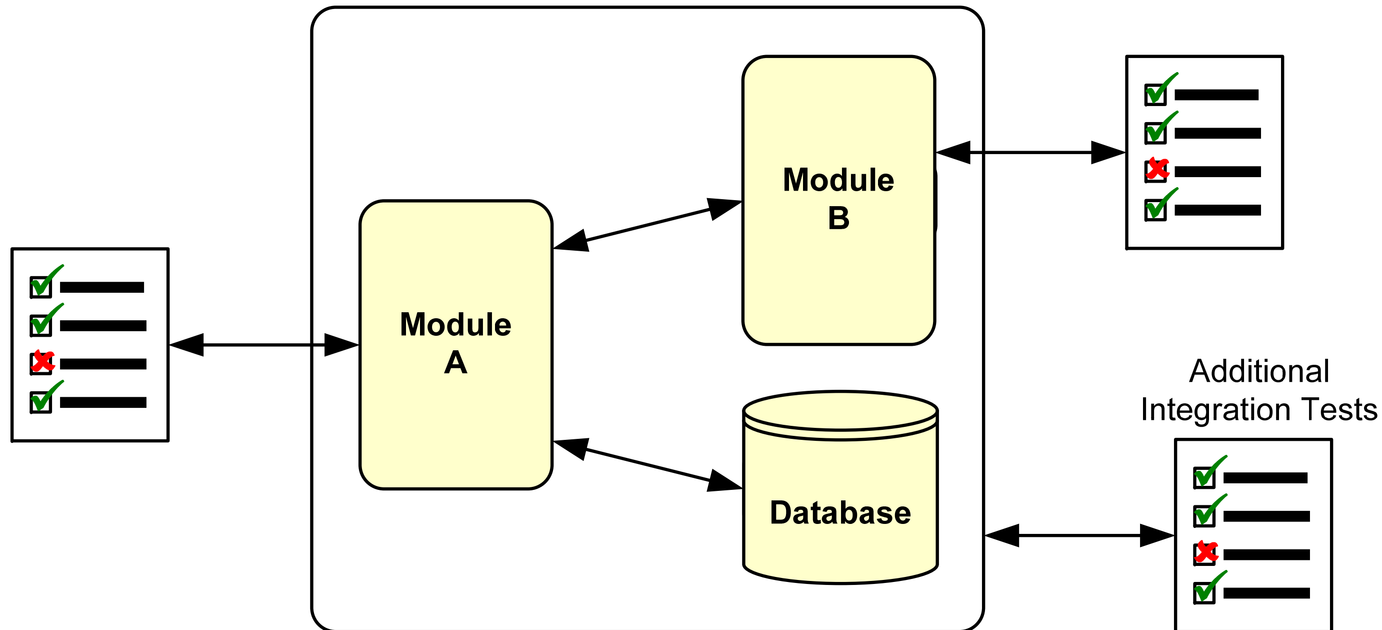


External dependencies are handled by creating mock objects.

Copyright © 2011 Boeing. All rights reserved.

TDD at the Integration Test Level

Initial integration tests are the unit tests with real components replacing mock objects.

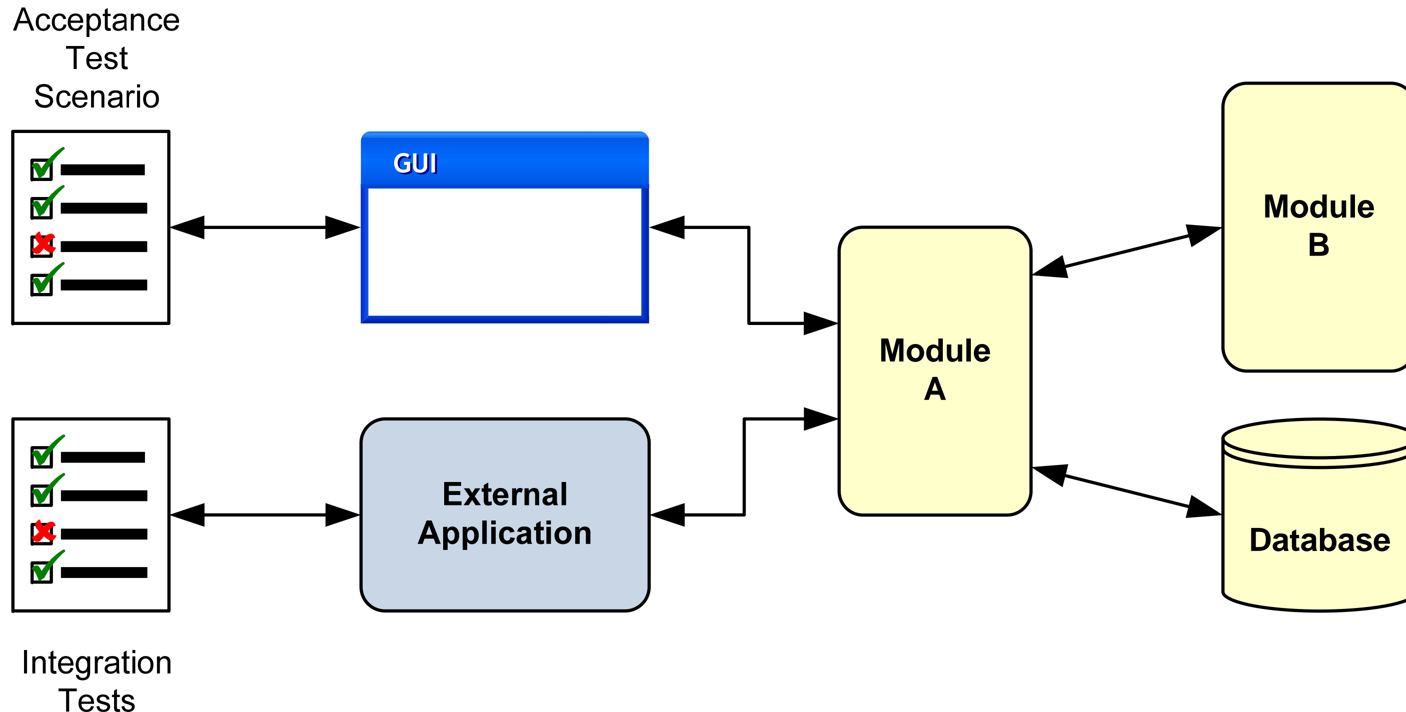


Additional integration tests may be needed to address scaling, loading or speed.

Copyright © 2011 Boeing. All rights reserved.

TDD at the Acceptance Test Level

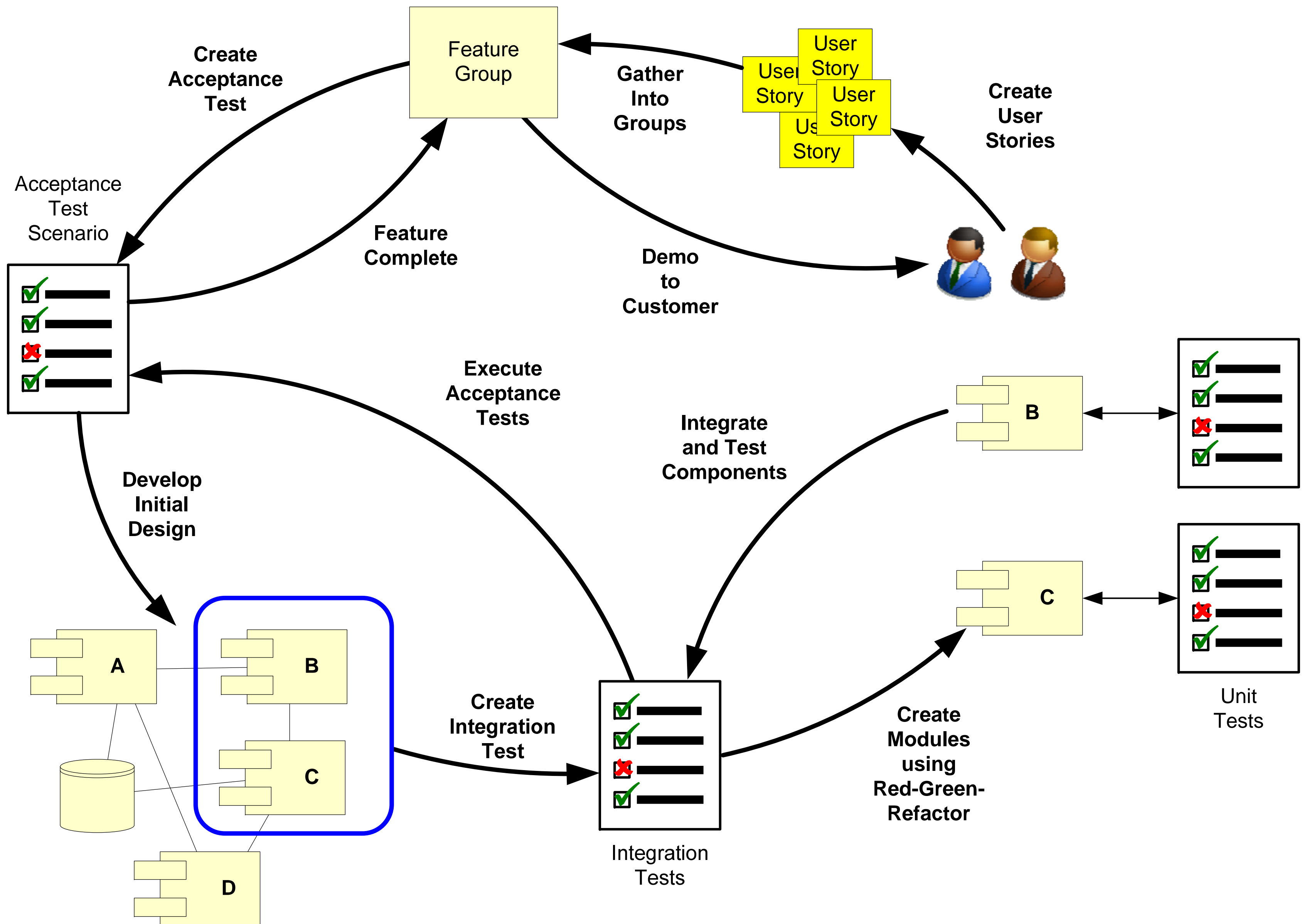
Acceptance tests may take the form of use case scenarios executed via a user interface …



… or they may be the integration tests from an external application.

Copyright © 2011 Boeing. All rights reserved.

Driving Development with Tests



Copyright © 2011 Boeing. All rights reserved.

From Details to Done

- Develop acceptance test scenarios from groups of related features.
- Develop integration tests for components of a simple, initial design.
- Develop unit tests and components using the red-green-refactor approach and mock objects.
- Integrate components by replacing mock objects with actual components and executing unit and integration tests.
- Execute acceptance test scenarios to ensure all functionality is complete.

Copyright © 2011 Boeing. All rights reserved.

Benefits of Test-Driven Development

Test-Driven Development

Testable Designs
Creates inherently testable designs

Complete Test Suite
Creates a test suite that can be retained for regression testing

Reduced Scope Creep
Fights developer-induced scope creep by limiting efforts to what needs to be developed

Lean Code – Simple Designs
Emphasis on writing just enough code drives lean and simple solutions

Definition of Done
Up front test definition provides a concrete "definition of done"

Customer Acceptance Tests
Allows customers to write acceptance level tests without needing to understand technical details

Copyright © 2011 Boeing. All rights reserved.

Drawbacks of Test-Driven Development

Test-Driven Development

Paradigm Shift/Learning Curve
Can affect productivity due to a lack of necessary skills and experience, as well as resistance to culture change

Drop in Perceived Productivity
Feature productivity is traded off for stability, quality and maintainability

Simple Designs
Creates a solution, but not necessarily the best or most efficient solution

Exhaustive Testing Not Addressed
Difficult and/or inefficient for projects requiring exhaustive testing

Not a Silver Bullet
Bad Requirements → Bad Tests → Bad Software

Copyright © 2011 Boeing. All rights reserved.

End

BOEING is a trademark of Boeing Management Company.
Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example – Hello World

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
   }
}
```

Compilation Error

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
    [TestMethod]
    public void TestDisplayHelloWorld()
    {
        Greeter myGreeter = new Greeter()
    }
}
```

Greeter

```
public class Greeter
{
    Greeter()
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), "Hello World")
   }
}
```

Compilation Error

Greeter

```
public class Greeter
{
   Greeter()
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
    [TestMethod]
    public void TestDisplayHelloWorld()
    {
        Greeter myGreeter = new Greeter()
        Assert(myGreeter.getGreeting(), "Hello World")
    }
}
```

Test Failure

Greeter

```
public class Greeter
{
    Greeter()

    String getGreeting()
    {
        return ""
    }
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), "Hello World")
   }
}
```

Greeter

```
public class Greeter
{
   Greeter()

   String getGreeting()
   {
      return "Hello World"
   }
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), "Hello World")
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   Greeter()

   String getGreeting()
   {
      return greeting
   }
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), expectedGreeting)
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   Greeter()

   String getGreeting()
   {
      return greeting
   }
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = new Greeter()
      Assert(myGreeter.getGreeting(), expectedGreeting)
   }
}
```

Compilation Error

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   private Greeter()

   static Greeter GetInstance()
   {
      return new Greeter()
   }

   String getGreeting()
   {
      return greeting
   }
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Greeter myGreeter = Greeter.GetInstance()
      Assert(myGreeter.getGreeting(), expectedGreeting)
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   private Greeter()

   static Greeter GetInstance()
   {
      return new Greeter()
   }

   String getGreeting()
   {
      return greeting
   }
}
```

Copyright © 2011 Boeing. All rights reserved.

Very Simple TDD Example (cont'd)

Greeter_Test

```
[Test]
public class Greeter_Test
{
   const String expectedGreeting = "Hello World"

   [TestMethod]
   public void TestDisplayHelloWorld()
   {
      Assert(Greeter.GetInstance().getGreeeting,
             expectedGreeting)
   }
}
```

Greeter

```
public class Greeter
{
   const String greeting = "Hello World"

   private Greeter()

   static Greeter GetInstance()
   {
      return new Greeter()
   }

   String getGreeting()
   {
      return greeting
   }
}
```

Copyright © 2011 Boeing. All rights reserved.