

Software Testing Secrets

For Faster Delivery
and Higher Quality



We are a world-class software delivery firm.



Web Development

Web Applications • Angular • React

We build contemporary web applications that delight users, including a modern user interface and responsive design.



Product Strategy

Technology Strategy • MVP Feature Prioritization

We dive deep to understand your company's business drivers, customer needs, and your vision to ensure your product excels, and help form the plan for your minimum viable product (MVP).



App Development

Mobile • Watch • Desktop • TV • Xamarin

Whether it's Android, iOS, watchOS, tvOS, or the desktop, our applications can reach your users wherever they are—with a rich, platform-specific experience.



Product Design

UI/UX • Visual Design • User Testing

UX architecture ensures the right structure to deliver on user expectations, while UI and visual design make your product beautiful.



Platform Development

APIs • Data • Microservices • AI

By leveraging the latest cloud computing, business analytics, microservice, and artificial intelligence technology, we ensure the highest levels of performance, security, and scalability.



Agile Training & Coaching

Agile Assessments • Scrum Training • DevOps

We teach your teams our agile project management and software engineering secrets perfected over 3+ decades of award-winning product delivery.

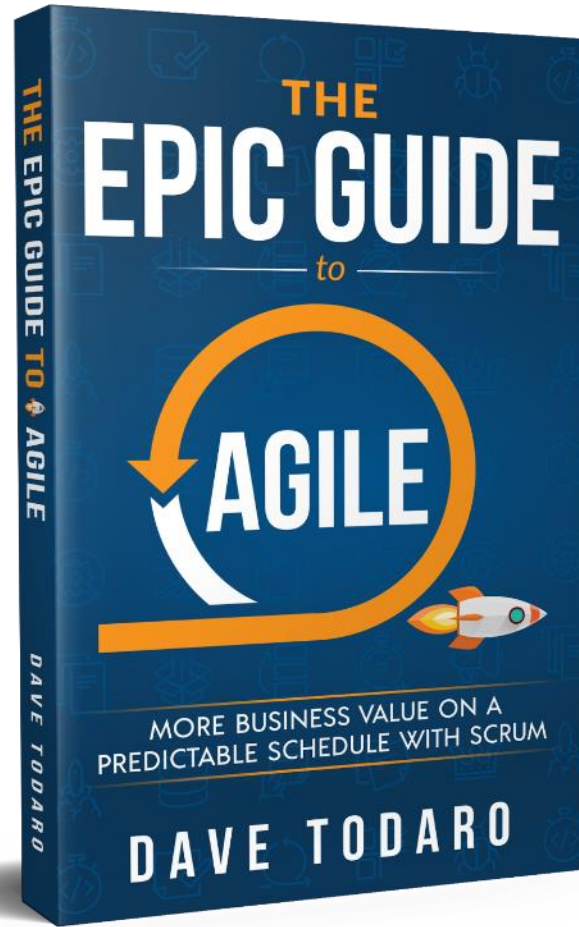
Our clients are market leaders and innovative pioneers.

amadeus

Honeywell

MONSTER





And we wrote
the book.

Dave Todaro

[linkedin.com/in/dtodaro](https://www.linkedin.com/in/dtodaro)

Founder & CEO
Ascendle



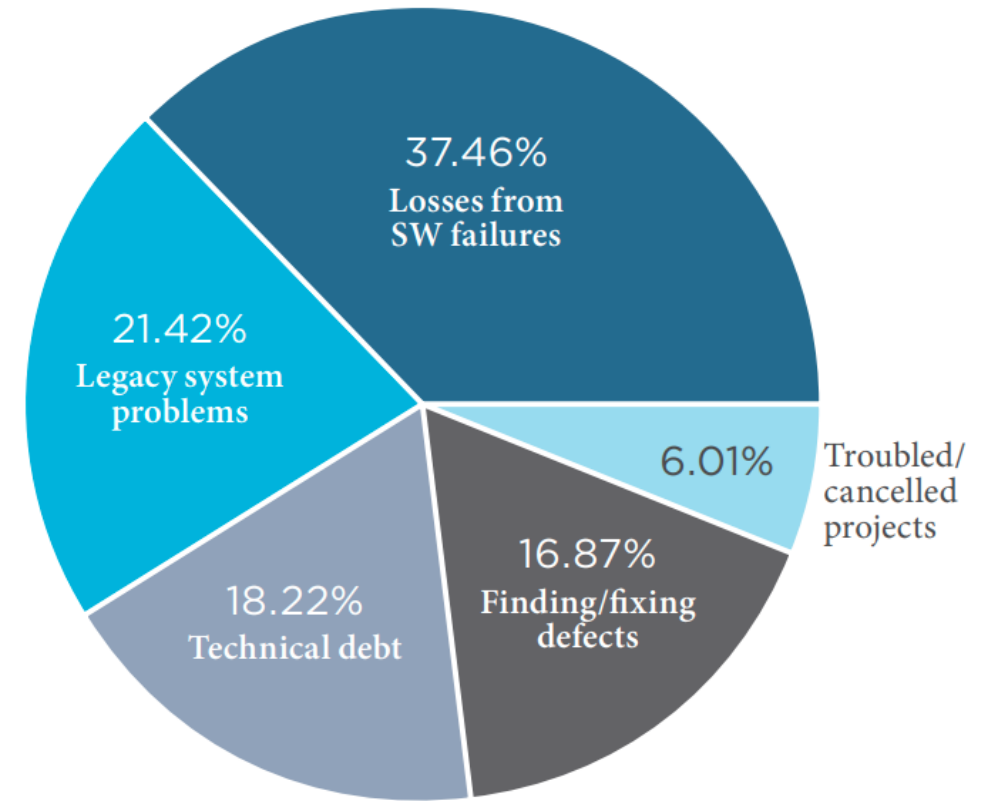


Poll Results

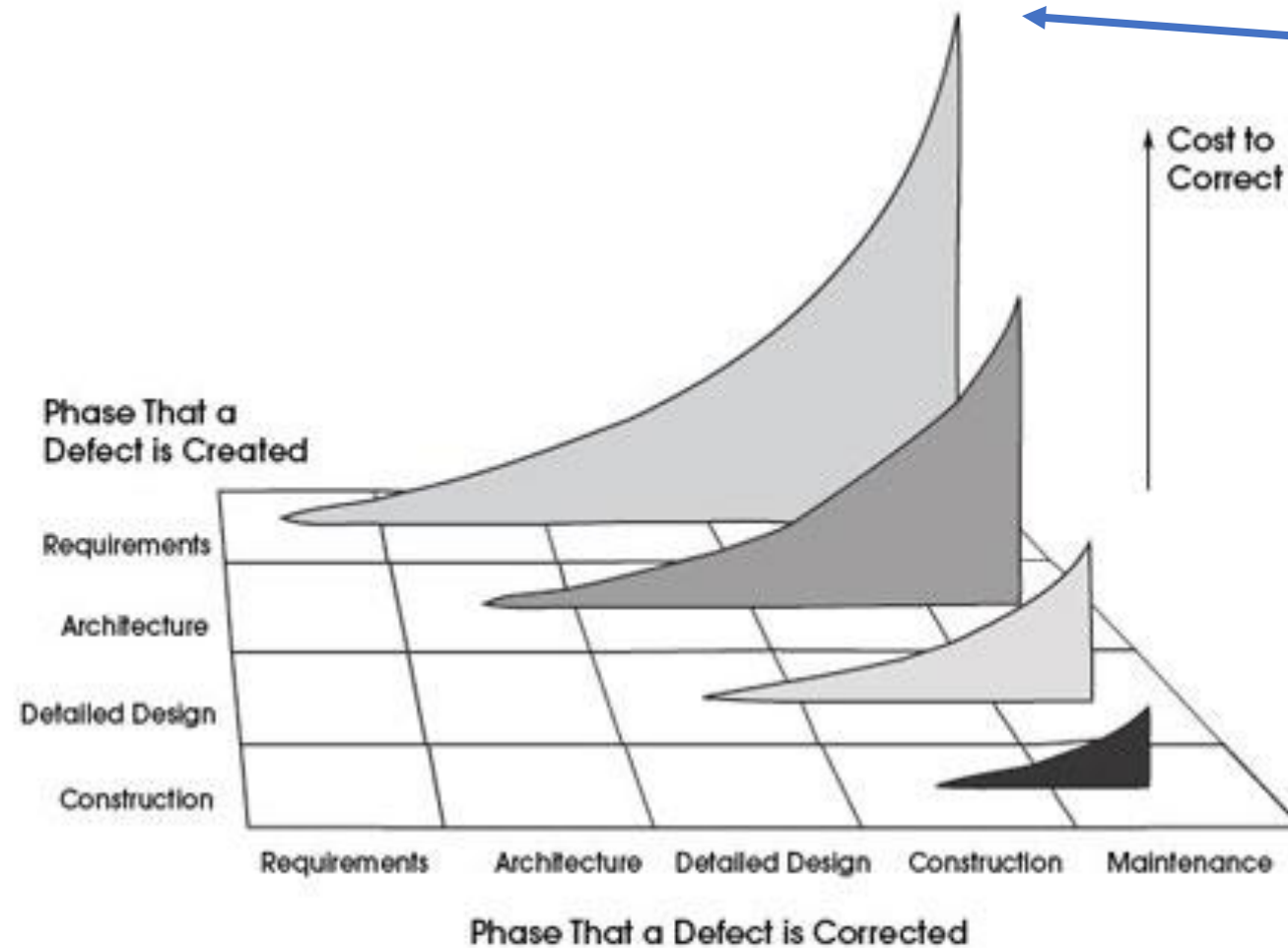
Why Should We Care About Quality?



\$2.8 trillion
in 2018



The Cost of Fixing Defects



50 to 200 times
the cost to
correct vs fixing
close to when it
was introduced



A Quality Standard

At the **conclusion of any sprint**, the product must be deployable to production and available to end users **within one additional sprint**.

*Quality is not an act,
it is a habit.*

- Aristotle

The 7 Habits of Highly Effective Teams

1. Create a clear vision through the use of **user stories** with **acceptance criteria**
2. Write **acceptance tests** to validate the vision and provide a testing strategy
- ③ 3. Implement comprehensive **automated unit testing** and **continuous integration**
4. Perform **peer code reviews** of every line of code
- ⑤ 5. Test manually within **team test environments** that support **automated deployment**, permitting anyone on the team to test
6. **Fix bugs before writing new code** to ensure the product is always potentially shippable
- ⑦ 7. Leverage **automated integration tests** to ensure components work when put together

Notice only 3 of the 7 habits are testing. A comprehensive quality strategy is key.

Today's Focus: Testing



Strategy



Test Planning



Manual Testing



Automated Unit Tests



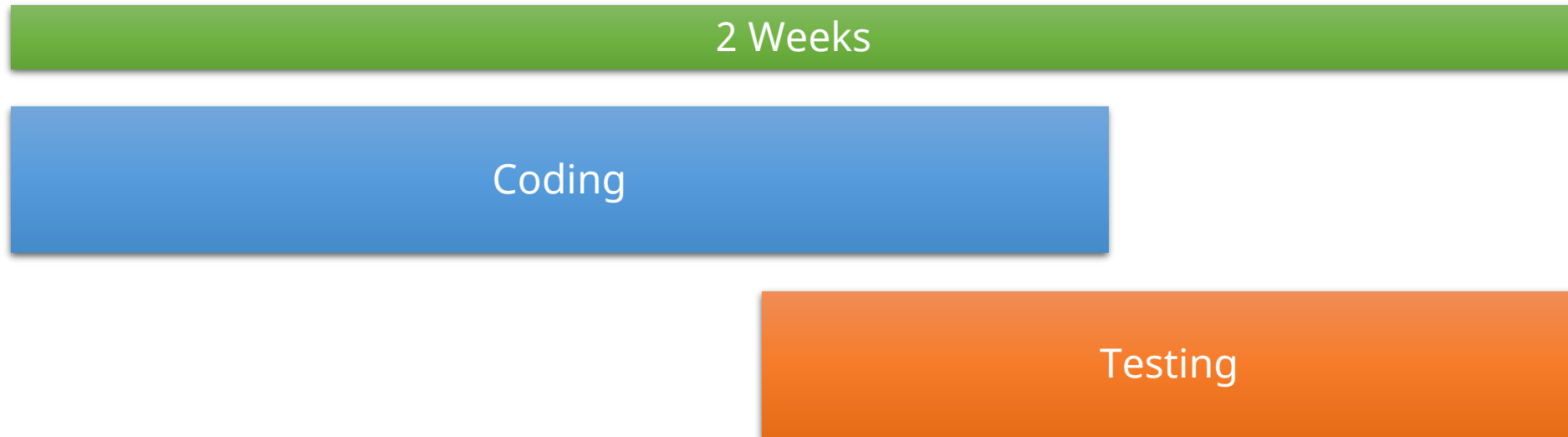
Automated Integration Tests



Strategy

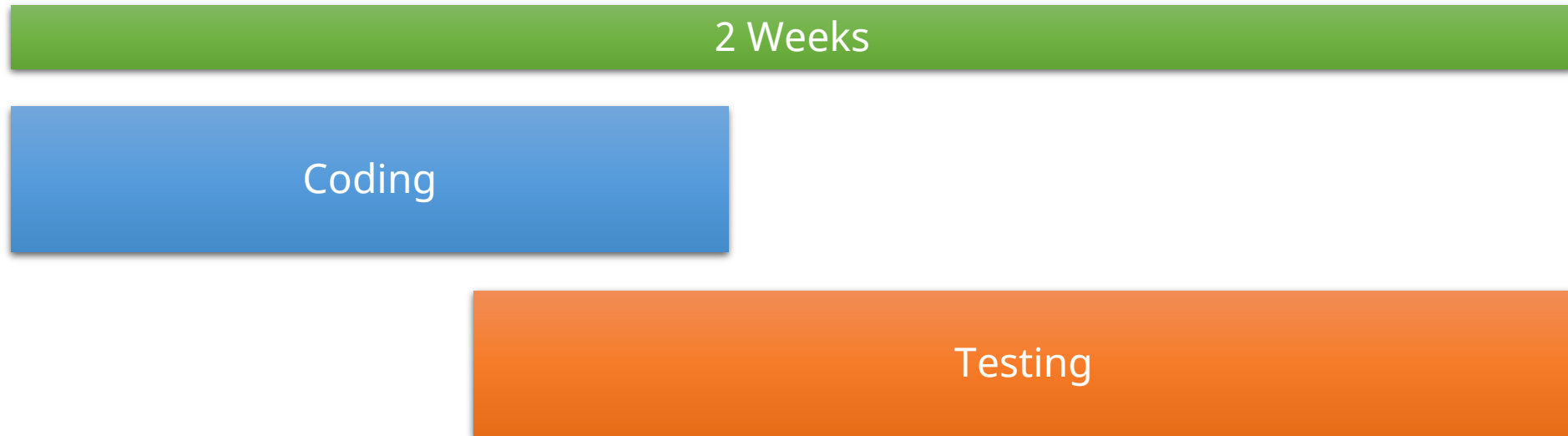
The Challenge With Manual Testing

At the Beginning



The Challenge With Manual Testing

As the Product Grows



The Challenge With Manual Testing

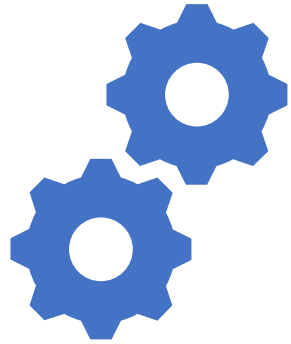
The Tipping Point

2 Weeks

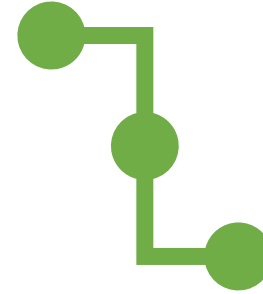
Coding

Testing

Solution: Test Automation

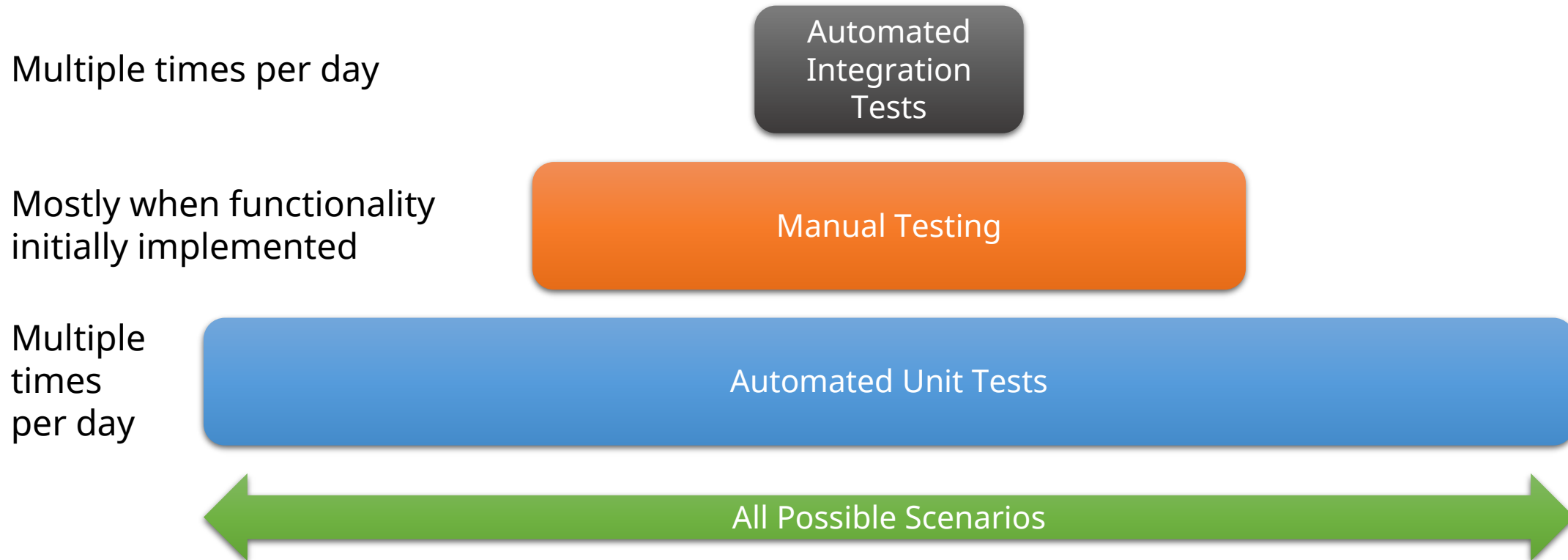


Automated Unit Testing



Automated Integration Testing

A Tiered Testing Strategy





Test Planning



Acceptance Tests

- Scenario-based testing approach
- Sometimes called “specification by example”
- Allows “pre-visualizing” intended behavior of the application
- I suggest Behavior-Driven Development (BDD) Given/When/Then format

Example User Story

As an author I want an indication of misspelled words so I can prevent spelling errors in my document

Acceptance Criteria:

- I can see misspelled words indicated by a wavy red line
- I see no wavy red line after I correct a misspelling





Summary

Indicate Misspelled Words

Background

Given I have the word processor application running
And I have a document open

Scenarios

Scenario 1: One word is misspelled

Given I am typing a new sentence in my document

When I type the word *helol*
And I press the space bar

Then I should see a wavy red line appear under the word *helol*

Scenario 2: Misspelling is corrected

Given I have the word *helol* in the document
And there is a wavy red line under *helol*

When I change *helol* to *hello*
And I press the space bar

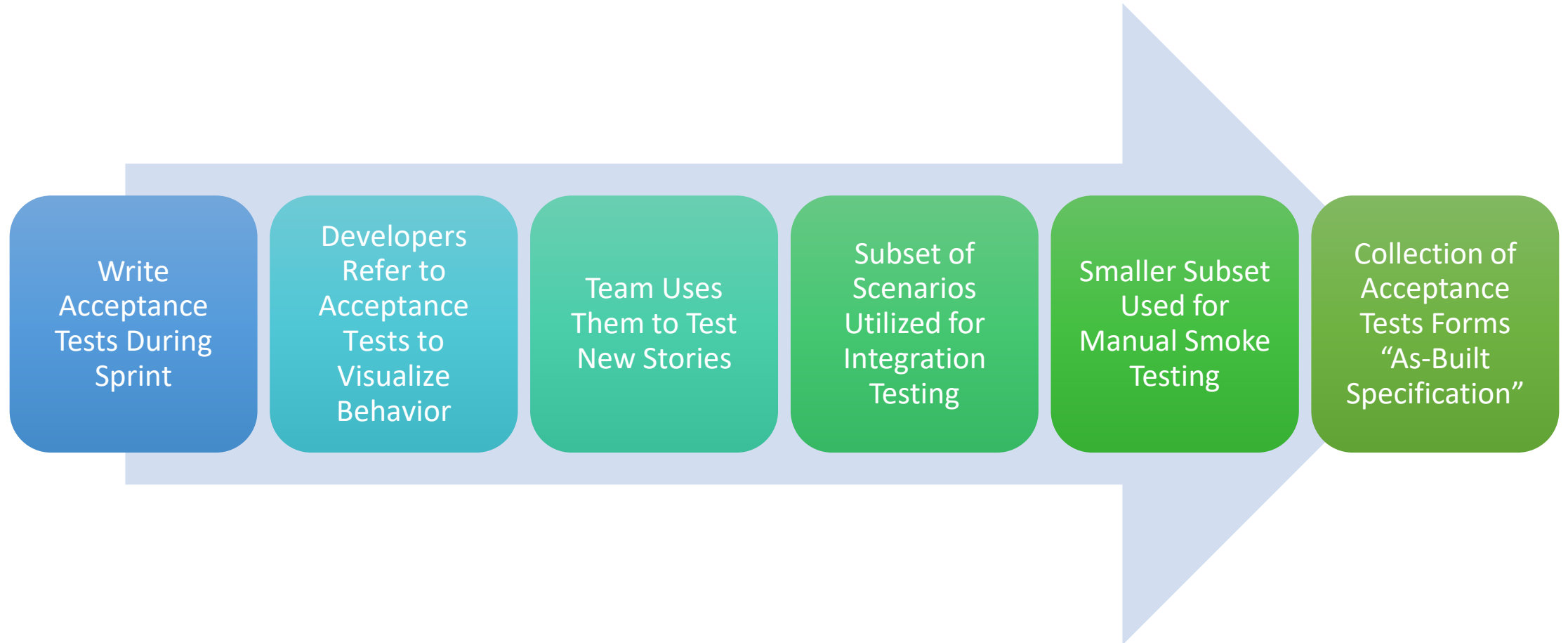
Then I should see the red line disappear from the word *hello*



Benefits

- Bridge between acceptance criteria of a story and implementation
- Allows “beta testing” acceptance criteria of a story
- Forces thinking through “gotchas” or exceptional scenarios early
- Provides raw material for:
 - Implementation
 - Unit tests
 - Manual tests
 - Integration tests
 - Documentation

Typical Workflow





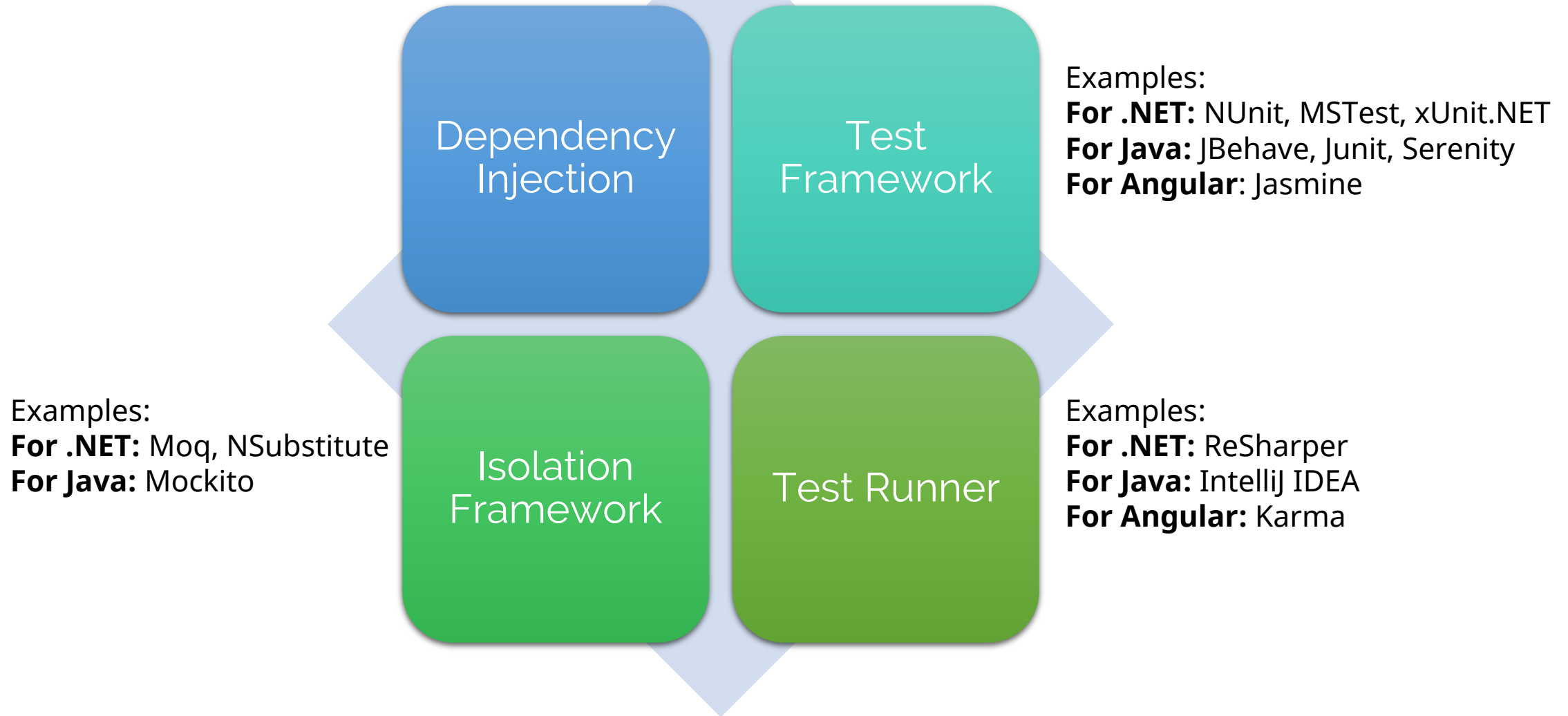
Automated Unit Tests

Unit vs Integration Testing

Unit Testing	Integration Testing
Tests a single logical concept in the system	Tests two or more dependent software modules as a group
Able to be fully automated	Often requires configuration and set-up to automate
Has full control over all the pieces running (isolated)	Often tests interaction with resources you may not fully control
Runs in memory (no database or file access, for example)	May include calls to databases or file systems
Can be run in any order if part of a test suite	Typically runs as a series of related steps – i.e. insert a record in the database, update it, delete it
Consistently returns the same result	May not always return the same result due to external dependencies
Runs quickly	Runs more slowly than unit tests as scope and resource usage are larger
Is readable, maintainable, and trustworthy	Requires more effort to maintain



Unit Testing: Components





Potential dependencies

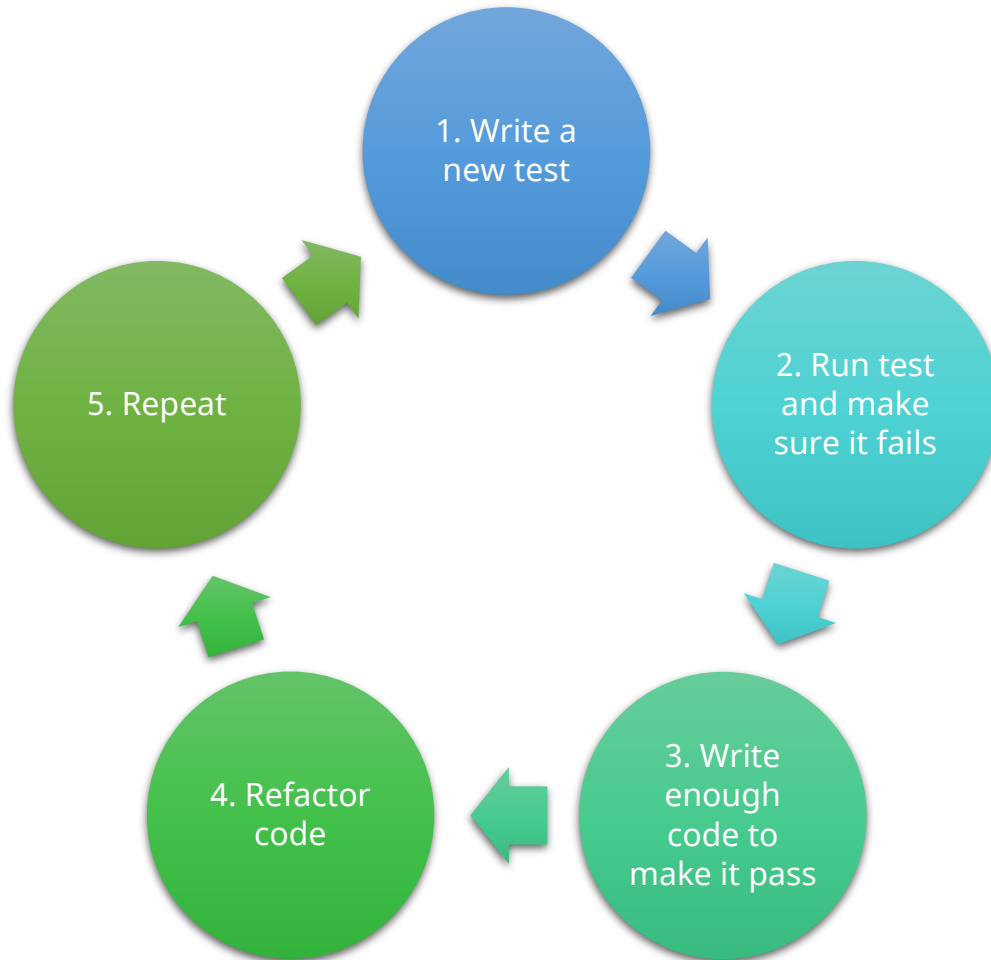
- External APIs
- Database
- File system
- System clock
- Many more

Potential problems if tests aren't isolated

- API is down
- Database password expired
- File System is missing an expected file
- System Clock causes test to yield a different result
- The unit test runs slowly

Recommended Approach

Test-Driven Development (TDD)



Benefits

- Ensures tests will fail if code breaks
- Design is based on desired behavior
- Forces developer to test before committing
- Tests document expected behavior of code

The Process



the art of

UNIT TESTING

with examples in C#

SECOND EDITION



FOREWORDS BY
Michael Feathers
Robert C. Martin

 MANNING

ROY OSHEROVE

Red

Write failing unit test

Log Analyzer Example – Checking for bad filename

```
public class LogAnalyzer
{
    public bool IsValidLogFileName(string fileName)
    {
        return false;
    }
}
```

Red

Write failing unit test

```
using System;  
using NUnit.Framework;
```

```
namespace LogAn.UnitTests
```

```
{  
    [TestFixture]  
    public class LogAnalyzerTests  
    {  
        [Test]  
        public void IsValidLogFileName_GoodExtension_ReturnsTrue()  
        {  
            LogAnalyzer analyzer = new LogAnalyzer();  
  
            bool result = analyzer.IsValidLogFileName("filewithgoodextension.slf");  
  
            Assert.True(result);  
        }  
    }  
}
```

[TestFixture]

public class LogAnalyzerTests

{

[Test]

Unit of Work

Scenario

Expected Behavior

public void IsValidLogFileName_GoodExtension_ReturnsTrue()

{

LogAnalyzer analyzer = new LogAnalyzer();

Arrange

bool result = analyzer.IsValidLogFileName("filewithgoodextension.slf");

Act

Assert.True(result);

Assert

}

}

}

Red

Run the failing unit test

```
[Test]
public void IsValidLogFileName_GoodExtension_ReturnsTrue()
{
    LogAnalyzer analyzer = new LogAnalyzer();

    bool result = analyzer.IsValidLogFileName("filewithgoodextension.slf");
}
```

Unit Test Sessions

IsValidLogFileName_GoodExtension_ReturnsTrue

1 0 1 0 0

pe to search

- LogAnalyzerTests (1 test) Failed: One or more child tests had errors: 1 test failed
- IsValidLogFileName_GoodExtension_ReturnsTrue Failed: Expected: True

IsValidLogFileName_GoodExtension_ReturnsTrue failed

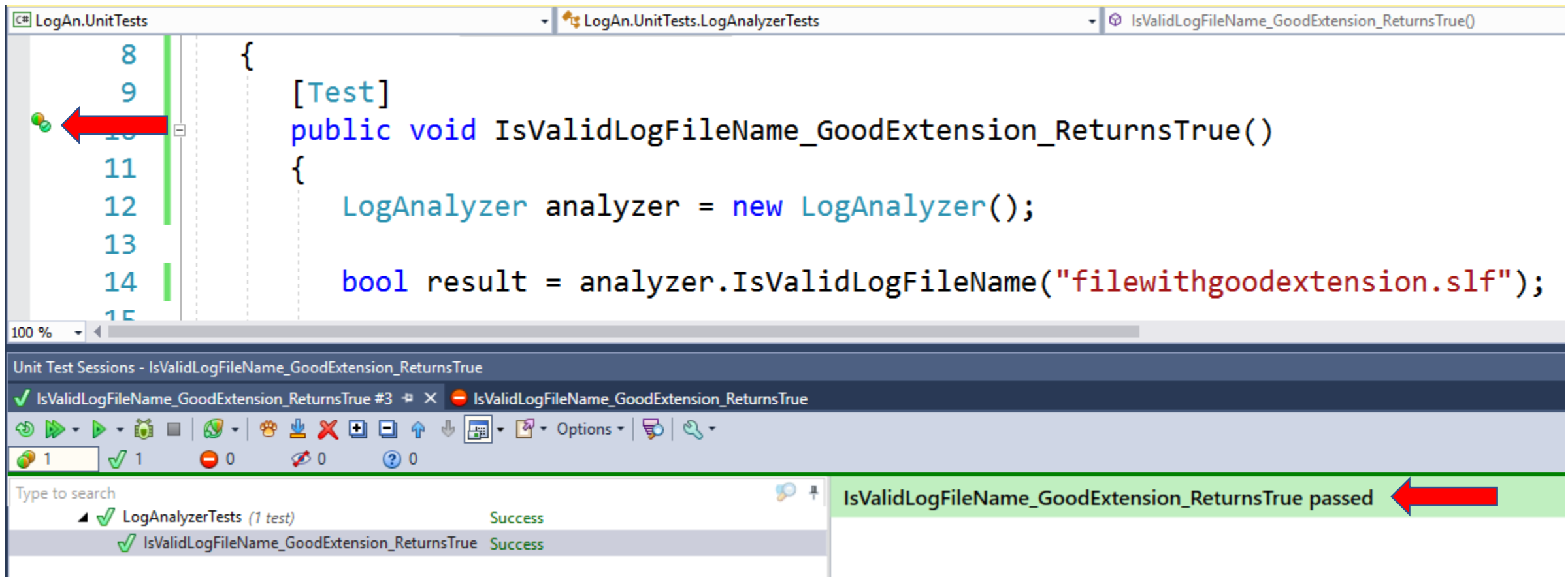
Expected: True
But was: False

Green

Write enough code to make it pass

```
public class LogAnalyzer
{
    public bool IsValidLogFileName(string fileName)
    {
        if (!fileName.EndsWith(".slf"))
        {
            return false;
        }
        return true;
    }
}
```


Green Run the modified test



The screenshot shows the Visual Studio IDE with the following components:

- Code Editor:** Displays the `IsValidLogFileName_GoodExtension_ReturnsTrue()` test method in `LogAnalyzerTests.cs`. A red arrow points to the test icon (a green circle with a white checkmark) on the left margin.
- Unit Test Explorer:** Located at the bottom, it shows the test session `IsValidLogFileName_GoodExtension_ReturnsTrue`. The test `IsValidLogFileName_GoodExtension_ReturnsTrue #3` is marked with a green checkmark and the word `Success`.
- Test Results:** A green banner at the bottom right displays the message `IsValidLogFileName_GoodExtension_ReturnsTrue passed`, with a red arrow pointing to it.

```
8 {  
9 [Test]  
10 public void IsValidLogFileName_GoodExtension_ReturnsTrue()  
11 {  
12     LogAnalyzer analyzer = new LogAnalyzer();  
13  
14     bool result = analyzer.IsValidLogFileName("filewithgoodextension.slf");  
15 }
```

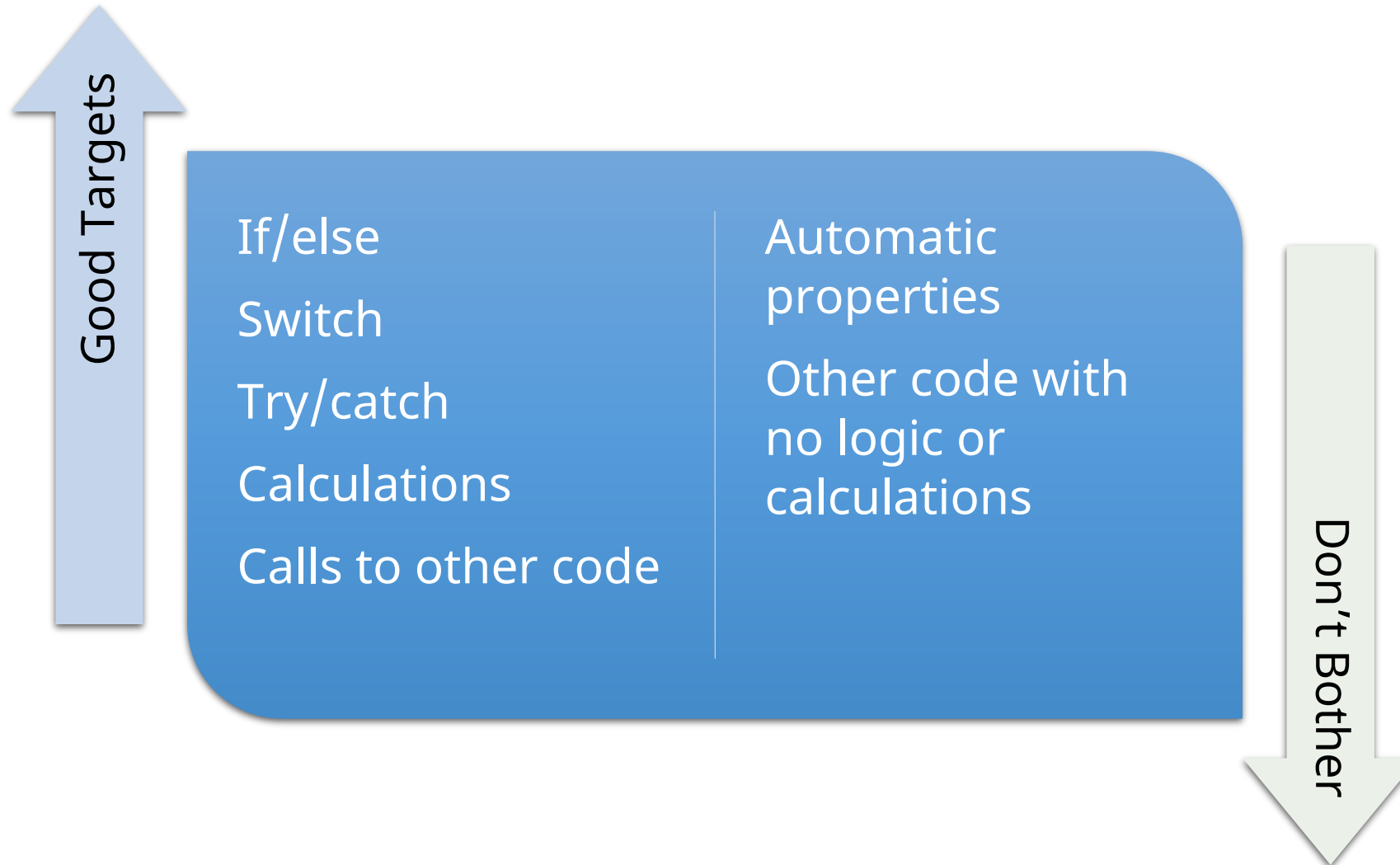
Refactor

Refactor the code

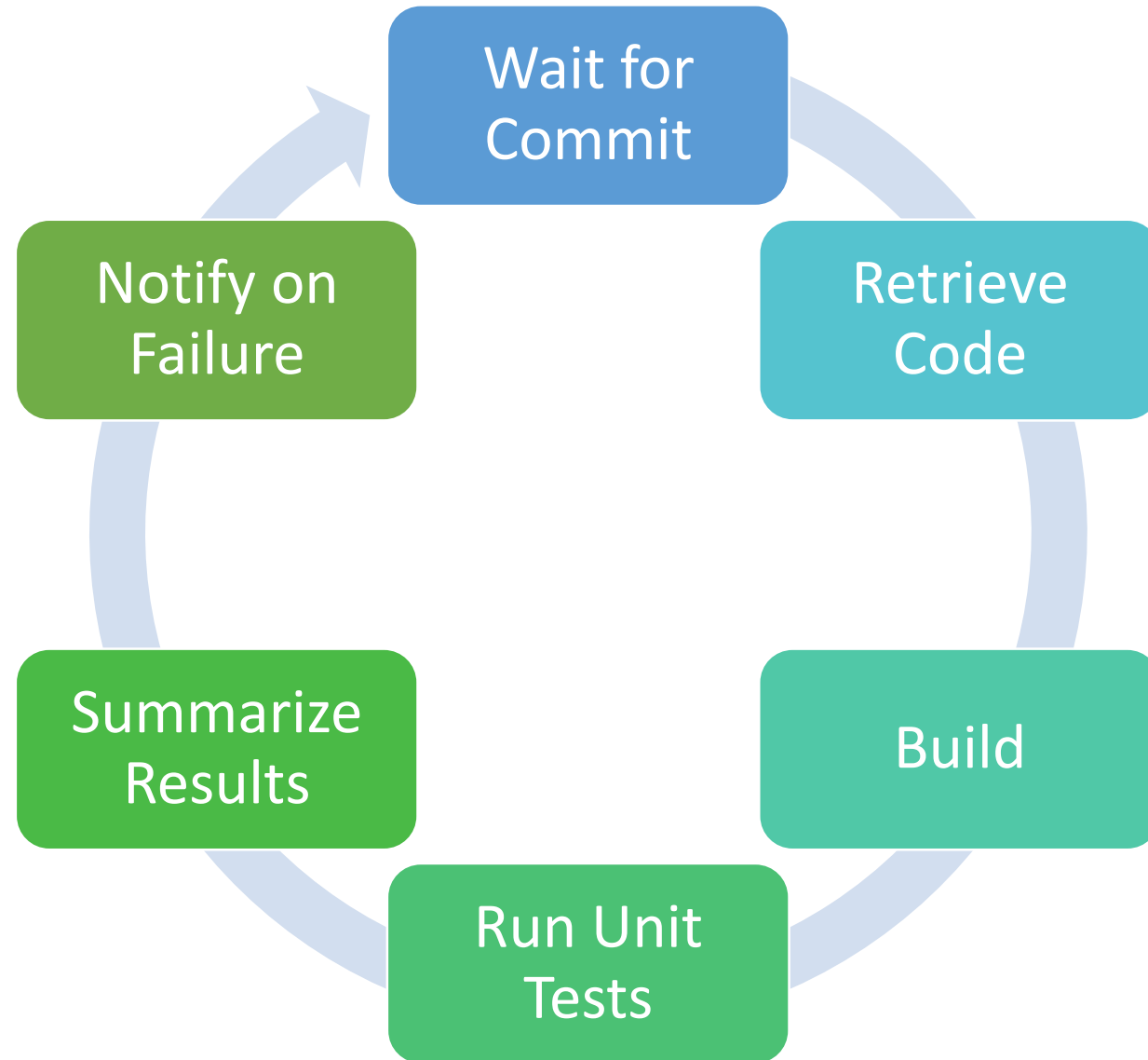
Examples:

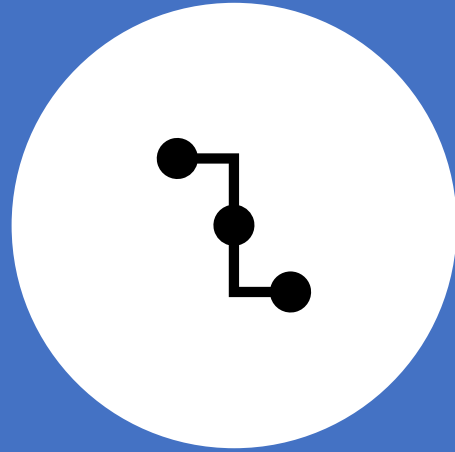
- Make code cleaner
- Add comments
- Make more efficient

What to Unit Test



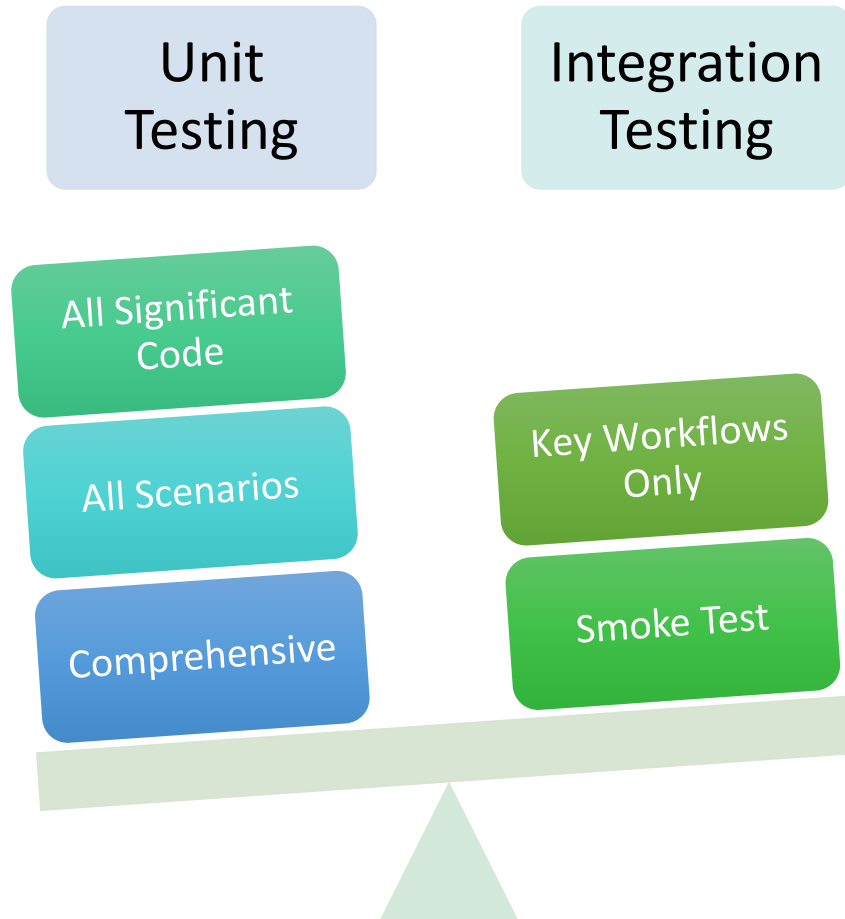
Continuous Integration





Automated Integration Tests

Balancing Unit and Integration Testing





What Should **Not** Be Covered by Automated Integration Tests?

- Don't try to test every possible scenario – that's much better suited for unit testing
- Focus on key workflows
- **Good target:** Ordering workflow for an e-commerce application
- **Bad target:** Every possible combination of state/county/city tax calculations

User Interface vs API-Driven Integration Testing

1

UI-driven testing can be:

- Time-consuming to create
- Brittle and needing lots of care and feeding

2

Solution: minimize UI-driven testing when alternatives exist

- Unit testing as a foundation
- API-driven integration testing as a next step

3

Don't abandon UI-driven testing altogether

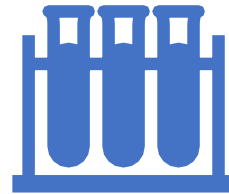
- You need to make sure the UI is still talking to the API!

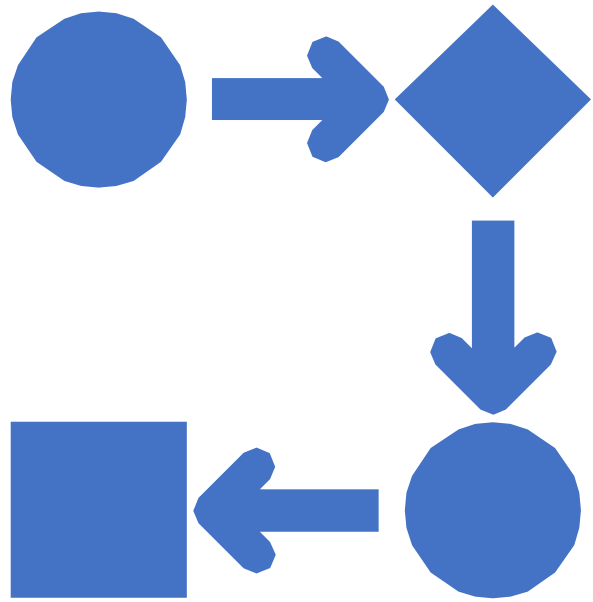


Manual Testing

Manual Testing Strategy

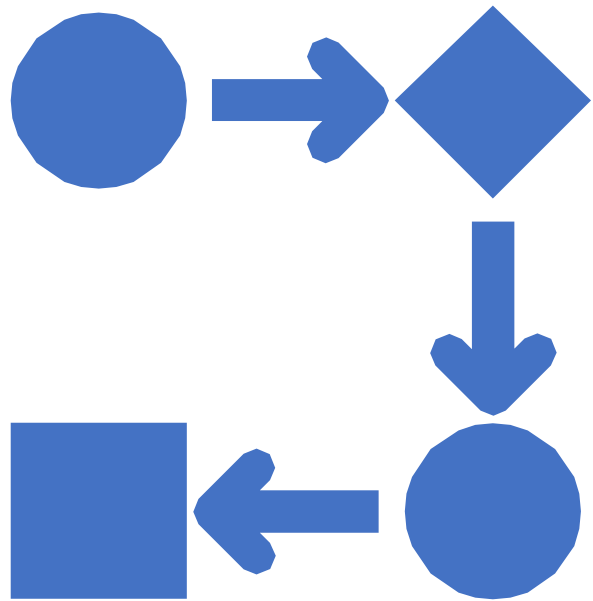
- Manual testing doesn't go away
- Test automation is the safety net
- But you still need to “touch and feel” the application





When to Test Manually During Each Sprint

- Developers step through new code
- Testing new stories in the sprint
- Product owner acceptance testing

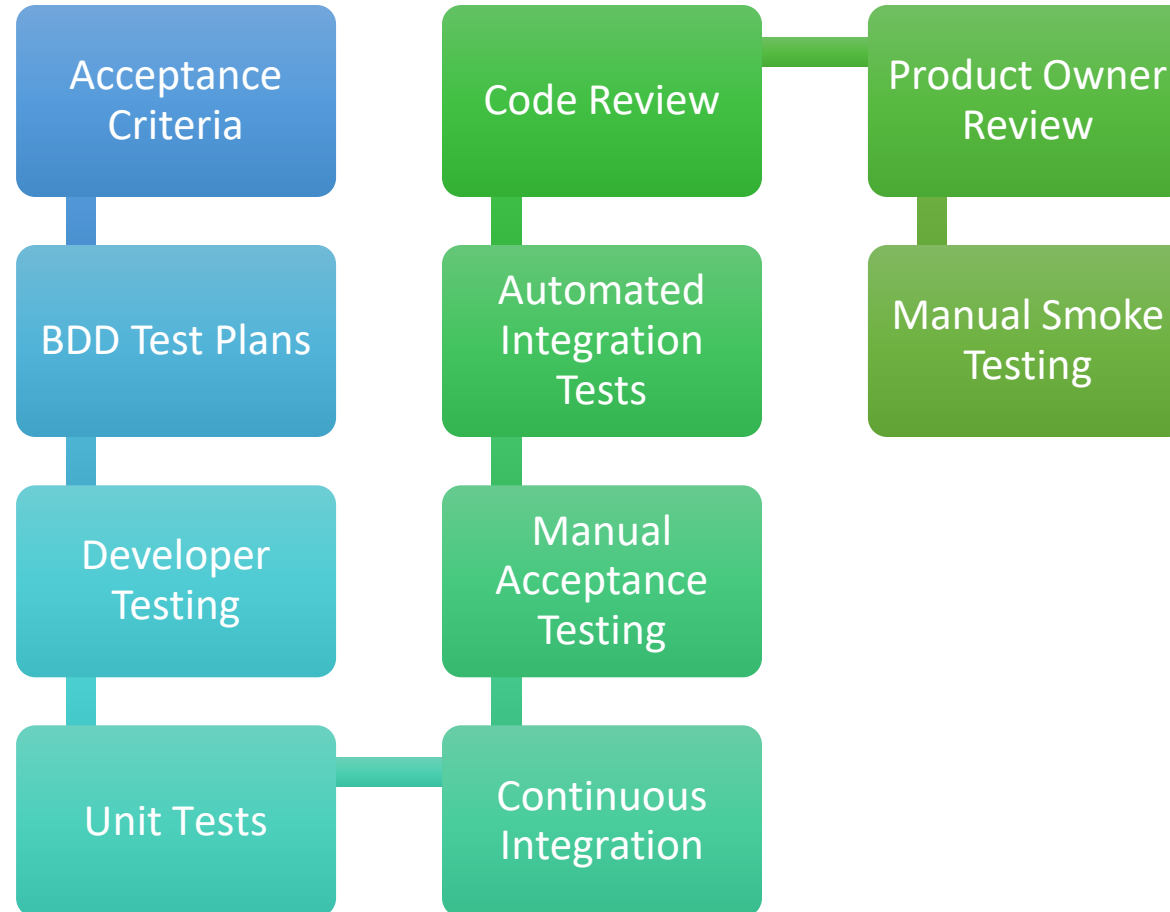


When to Test Manually Before a New Release

- Smoke testing
 - Lightweight “sanity check”
- Time box
 - 2 to 4 hours
 - Focused on critical workflows

Putting it All Together

10 Elements of Quality Our Teams Use



Key Takeaways

- Low software quality incurs significant cost
- High quality isn't "free," but it speeds overall development
- Testing is only part of the picture
 - Inspecting requirements is important
 - Code inspections can detect bugs in $\frac{1}{4}$ of the time required for testing
- Use unit tests for comprehensive coverage and integration tests for key workflows
- Manual testing is still important, but used more during initial development and for pre-release sanity checks



Further Reading

Chapter 10

Introduction to Software Construction

Chapter 11

Creating a Shippable Increment Requires Team Test Environments

Chapter 13

Automated Unit Testing: The Foundation of Code Quality

Chapter 14

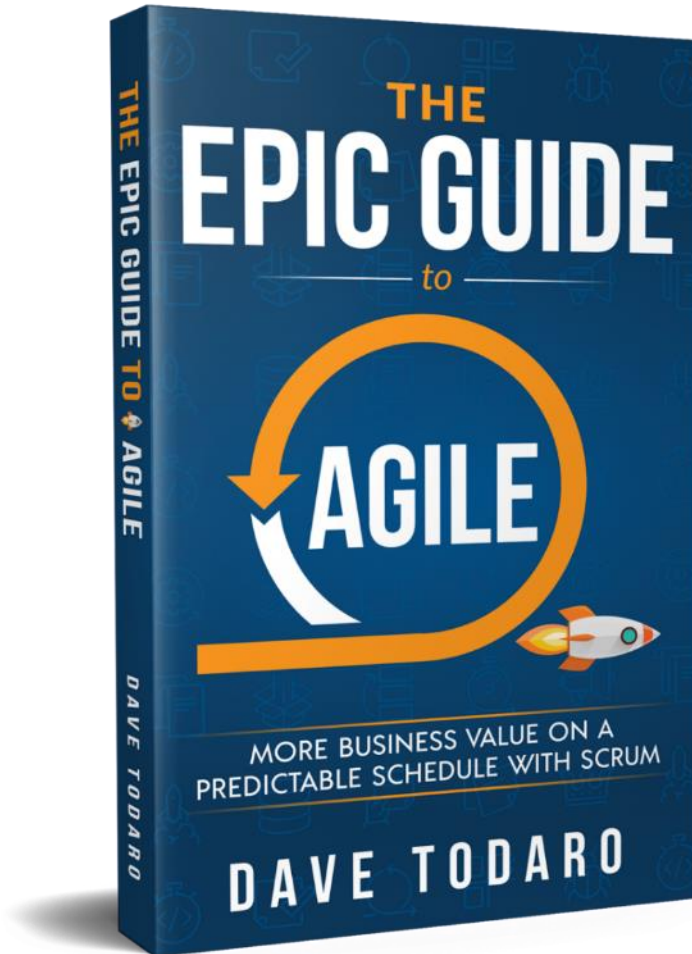
Acceptance Testing: Making Sure Everything Works Together

Chapter 27

A “Zero Defects” Approach to Dealing with Bugs

Chapter 30

Confidently Releasing Your Product into Production



Questions?



Dave Todaro

[linkedin.com/in/dtodaro](https://www.linkedin.com/in/dtodaro)

dave@ascendle.com