



# Unit Testing Makes Me Faster

Convincing Your Boss, Your Coworkers, and Yourself

Jeremy Clark

[www.jeremybytes.com](http://www.jeremybytes.com)

# Bosses Hate Tests

## Production Code

```
1  using ...
10
11 namespace Module.Catalog
12 {
13     public class CatalogViewModel : INotifyPropertyChanged
14     {
15         Fields
30
31         Properties
126
127         Constructors
135
136         Methods
251
252         INotifyPropertyChanged Members
263     }
264 }
265
```

## Test Code

```
1  using ...
11
12 namespace Module.Catalog.Test
13 {
14     [TestClass]
15     public class CatalogViewModelTest
16     {
17         Test Initialization
116
117         Model Initialization
148
149         Catalog Population
182
183         Service Exception
240
241         Catalog Caching
287
288         Filters
355
356         Filter Reset
418
419         Catalog Item Selection
534
535     }
536
```

Is Typing Really  
Our Limitation?



# Different Kinds of Tests

- Unit Testing
- Integration Testing
- Performance Testing
- Exploratory Test
- Penetration Testing
- User Acceptance Testing (UAT)

# What are Unit Tests?

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

*The Art of Unit Testing* by Roy Osherove

Non-Threatening  
Text Here



Threatening  
Text Here



# What are Unit Tests?

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

**automated piece of code**

**a unit of work**

**checks a single  
assumption**



# What Makes Me Faster?

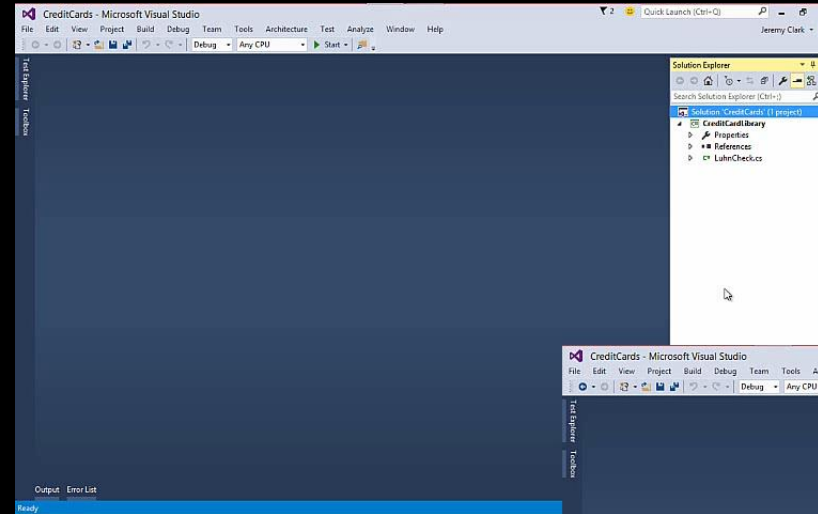
- Confirming Functionality
- Checking Regression
- Pinpointing Bugs
- Documenting Functionality



# Confirming Functionality

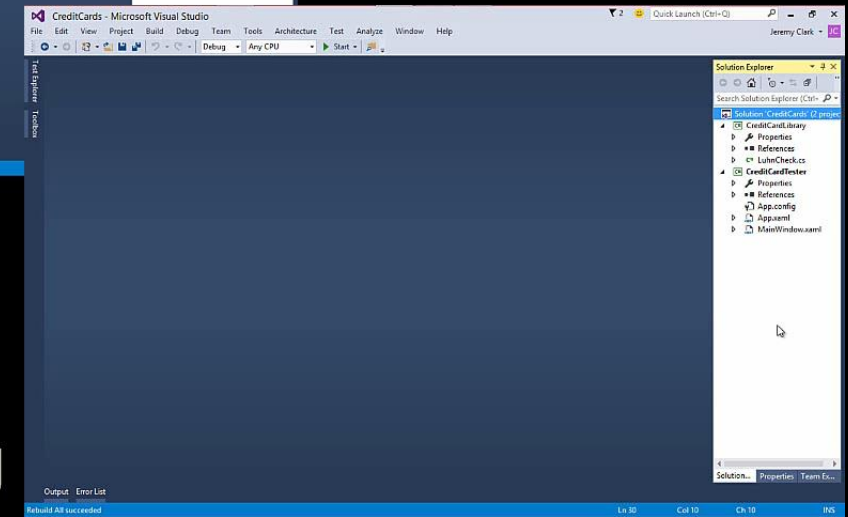
Unit Tests are ***proof*** that my code  
does what I ***think*** it does

# Build Time Comparison



# Test Application

# Unit Testing

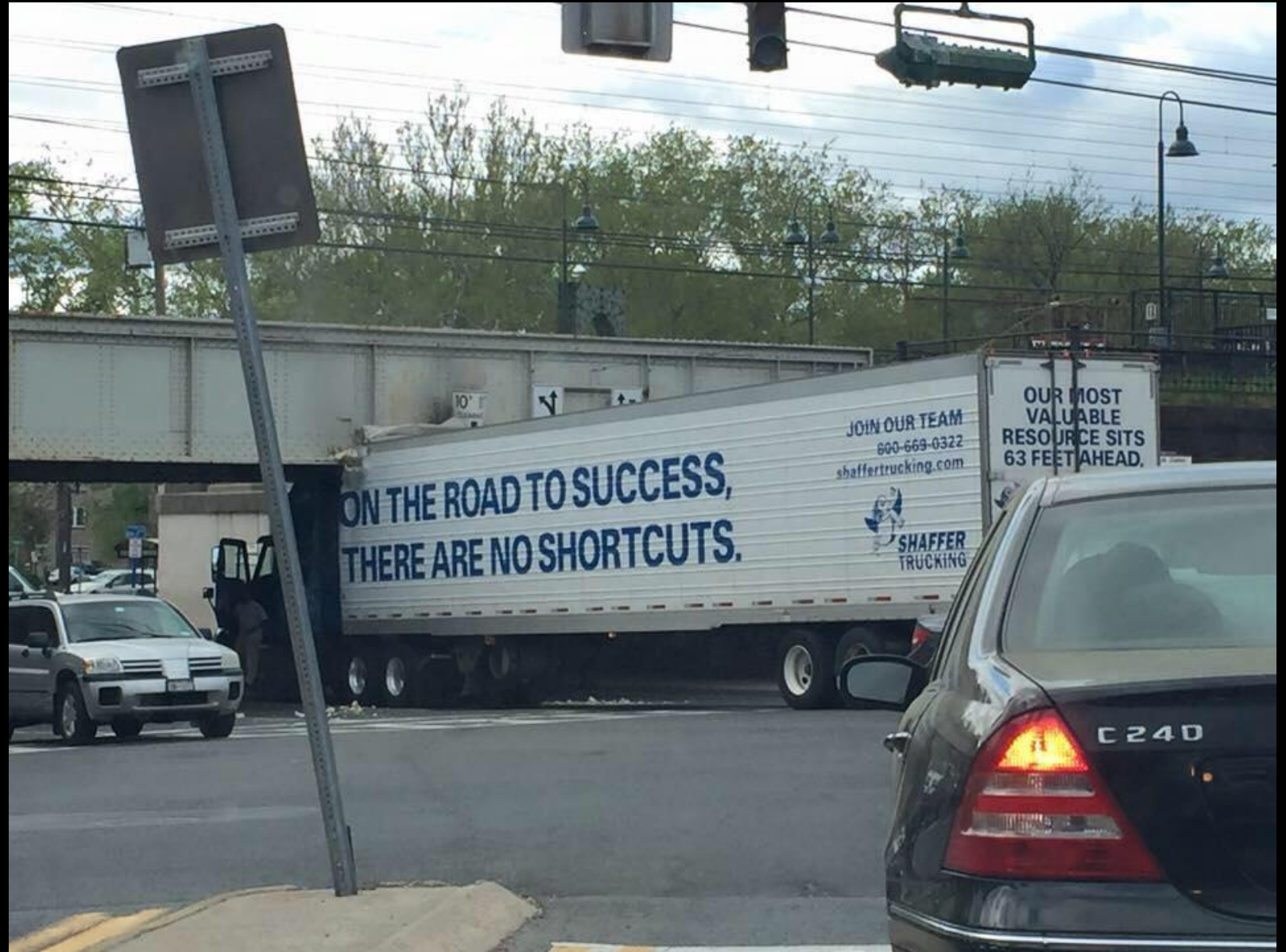




## Disclaimer

We get these advantages when we are **comfortable** writing **good** tests.

# Realistic Expectations



# Checking Regression

The screenshot displays the Visual Studio IDE with the Test Explorer on the left and the LuhnCheckTests.cs file open in the main editor. The Test Explorer shows 15 passed tests for the LuhnCheck class, including tests for invalid numbers (returning false) and valid numbers (returning true). The code in the main editor implements the LuhnCheck class with a PassesLuhnCheck method that calculates the checksum using a DELTAS array.

**Test Explorer - Passed Tests (15)**

| Test Name  | Duration |
|--|----------|
| PassesLuhnCheck_OnInvalidNumber_ReturnsFalse("-01233454567")     | < 1 ms   |
| PassesLuhnCheck_OnInvalidNumber_ReturnsFalse("123")              | < 1 ms   |
| PassesLuhnCheck_OnInvalidNumber_ReturnsFalse("7147894289")       | 8 ms     |
| PassesLuhnCheck_OnInvalidNumber_ReturnsFalse("9876543210987654") | < 1 ms   |
| PassesLuhnCheck_OnInvalidNumber_ReturnsFalse("abc")              | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("3530111333300000")    | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("3566002020360505")    | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("371449635398431")     | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("378282246310005")     | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("4012888888881881")    | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("4111111111111111")    | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("5105105105100")       | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("555555555554444")     | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("6011000990139424")    | < 1 ms   |
| PassesLuhnCheck_OnValidNumber_ReturnsTrue("601111111111117")     | < 1 ms   |

```
public static class LuhnCheck
{
    public static bool PassesLuhnCheck(string cardNumber)
    {
        try
        {
            int[] DELTAS = new int[] { 0, 1, 2, 3, 4, -4, -3, -2, -1 };
            int checksum = 0;
            char[] chars = cardNumber.ToCharArray();
            for (int i = chars.Length - 1; i > -1; i--)
            {
                int j = ((int)chars[i]) - 48;
                checksum += j;
                if (((i - chars.Length) % 2) == 0)
                    checksum += DELTAS[j];
            }

            return ((checksum % 10) == 0);
        }
        catch (Exception)
        {
            return false;
        }
    }
}
```

# Regression Comparison

```
public static class LuhnCheck
{
    try
    {
        int[] DELTAS = new int[] { 0, 1, 2, 3, 4, -4, -3, -2, -1, 0 };
        int checksum = 0;
        char[] chars = cardNumber.ToCharArray();
        for (int i = chars.Length - 1; i > -1; i--)
        {
            int j = ((int)chars[i]) - 48;
            checksum += j;
            if (((i - chars.Length) % 2) == 0)
                checksum += DELTAS[j];
        }
        return ((checksum % 10) == 0);
    }
    catch (Exception)
    {
        return false;
    }
}
```

# Test Application

# Unit Testing

Test Explorer Summary:

- Failed Tests (2):
  - PassesLuhnCheck\_OrInvalidNumber... 1 ms
  - PassesLuhnCheck\_OrInvalidNumber... 11 ms
- Passed Tests (18):
  - PassesLuhnCheck\_OrInvalidNumber... < 1 ms
  - PassesLuhnCheck\_OrInvalidNumber... 9 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms
  - PassesLuhnCheck\_OrValidNumber... < 1 ms

Summary: Last Test Run Failed (Total Run Time: 0:00:01)  
2 Tests Failed  
2 Tests Passed

# Pinpointing Bugs

The screenshot displays the Visual Studio interface. On the left, the Test Explorer window shows a list of tests. The top section, 'Failed Tests (6)', is highlighted with a red bar. The tests listed are:

- ✘ CatalogViewModel\_OnInitialization\_ModelsPopulated (161 ms)
- ✘ ModelSelectedItems\_AddToSelectionWithExistingPerson\_SelectionIsUnchanged (2 ms)
- ✘ ModelSelectedItems\_AddToSelectionWithNewPerson\_PersonAdded (3 ms)
- ✘ ModelSelectedItems\_RemoveFromSelectionWithExistingPerson\_PersonRemoved (1 ms)
- ✘ ModelSelectedItems\_RemoveFromSelectionWithNewPerson\_SelectionIsUnchan... (1 ms)
- ✘ ModelSelectedPeople\_OnClearSelection\_IsEmpty (1 ms)

The bottom section, 'Passed Tests (13)', lists 13 tests that passed successfully. A 'Summary' section at the bottom indicates 'Last Test Run Failed (Total Run Time 0:00:03)' with 6 tests failed and 13 tests passed.

On the right, the code editor shows the `CatalogViewModel.cs` file. The `Initialize()` method is expanded, and a red box highlights the `GetModelFromContainer()` call. The code is as follows:

```
#region Methods

public void Initialize()
{
    _service = GetServiceFromContainer();
    GetModelFromContainer();

    RefreshCatalog();
}

private CatalogOrder GetModelFromContainer()
{
    if (!_container.IsRegistered<CatalogOrder>("CurrentOrder"))
        throw new MissingFieldException(
            "CurrentOrder is not available from the DI Containe");
    return _container.Resolve<CatalogOrder>("CurrentOrder");
}

private IPersonService GetServiceFromContainer()
{
    if (!_container.IsRegistered<IPersonService>())
        throw new MissingFieldException(
            "IPersonService is not available from the DI Contai");
    return _container.Resolve<IPersonService>();
}

public void RefreshCatalog()...

private void RefreshCatalogFromService()...
```



# Documenting Functionality

- ✓ Catalog\_FilterDoesNotInclude00s\_00sRecordsIsNotIncluded
- ✓ Catalog\_FilterDoesNotInclude70s\_70sRecordsIsNotIncluded
- ✓ Catalog\_FilterIncludes00s\_00sRecordsIsIncluded
- ✓ Catalog\_FilterIncludes70s\_70sRecordsIsIncluded
- ✓ CatalogService\_OnRefreshAndCacheExpired\_ServicesCalledTwice
- ✓ CatalogService\_OnRefreshAndCacheNotExpired\_ServicesCalledOnce
- ✓ CatalogViewModel\_OnInitialization\_CatalogsPopulated
- ✓ CatalogViewModel\_OnInitialization\_ModelsPopulated
- ✓ CatalogViewModel\_OnInitializationAndCurrentOrderMissing\_ThrowsException
- ✓ CatalogViewModel\_OnInitializationAndPersonServiceMissing\_ThrowsException

✓ Filters\_OnRefreshAndCacheExpired\_AreResetToDefaults

✓ Filters\_OnRefreshAndCacheNotExpired\_AreResetToDefaults

✓ ModelSelectedItem\_AddToSelectionWithExistingPerson\_SelectionIsUnchanged

✓ ModelSelectedItem\_AddToSelectionWithNewPerson\_PersonAdded



## Disclaimer

We get these advantages when we are **comfortable** writing **good** tests.



# Good Unit Tests

- Maintainable
- Dependable
- Runnable

# Qualities of a Good Test

## Maintainable

- Not Tricky
- Easy to Read
- Easy to Write
- Well-Named

## Dependable

- Consistent Results
- Isolated
- Continued Relevance
- Tests the Right Things

## Runnable

- FAST

# Michael C. Feathers on Speed

“A unit test that takes 1/10<sup>th</sup> of a second to run is a slow unit test.”

“Unit tests run fast. If they don't run fast, they aren't unit tests.”

# Qualities of a Good Test

## Maintainable

- Not Tricky
- Easy to Read
- Easy to Write
- Well-Named

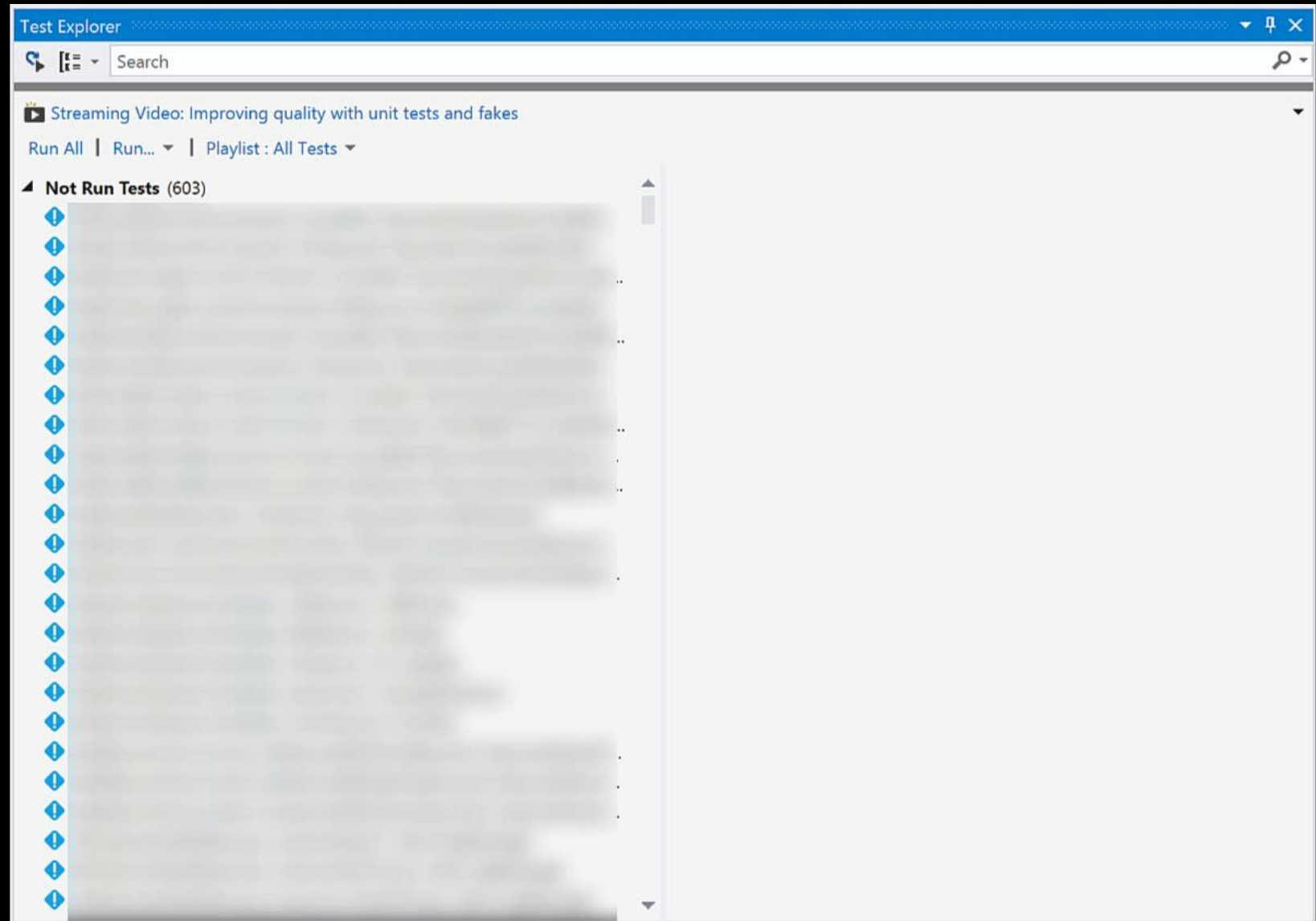
## Dependable

- Consistent Results
- Isolated
- Continued Relevance
- Tests the Right Things

## Runnable

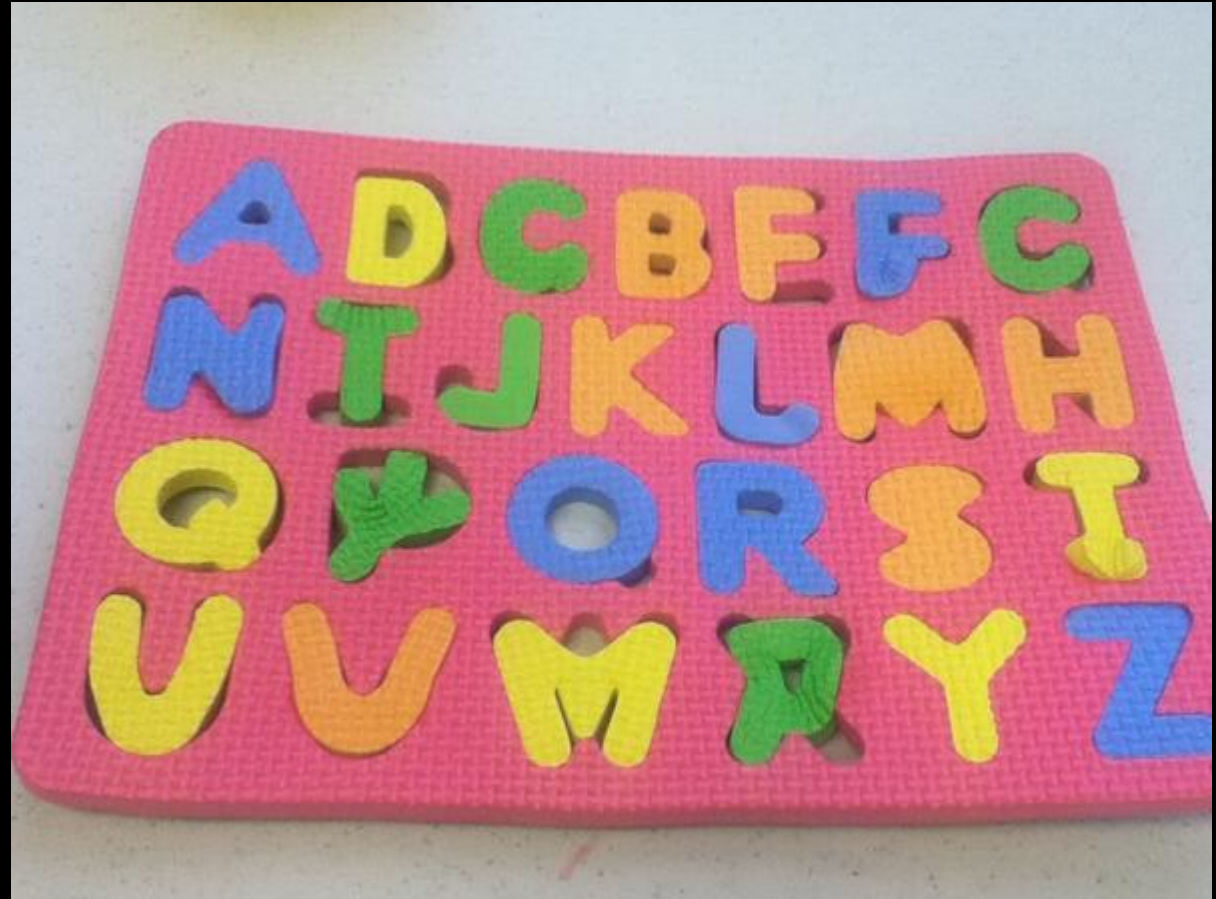
- FAST
- Single Click
- Repeatable
- Failure Points to the Problem

# Isolated and Fast



# Code Coverage

100% Code Coverage  
is not a guarantee





# Conversations about Code Coverage

“What parts of your application are  
okay *not* to test?”

# The Stahl Standard

“What parts of your application do your users **not** care about?”

-Barry Stahl

# Know the Goals

- Don't do the right thing for the wrong reason.
- Unit testing will not fix bad development practices.



<http://www.jenders.com/2012/01/08/thief-almost-caught-on-camera-stealing-thin-lg-television/>



## Martin Fowler on Fear

“Don’t let the fear that testing can’t catch all bugs stop you from writing the tests that will catch most bugs.”

*Refactoring* by Martin Fowler et al.

# References

- *The Art of Unit Testing with Examples in C#* – Roy Osherove
- *Refactoring* – Martin Fowler et al.
- *Working Effectively with Legacy Code* – Michael C. Feathers
  
- *Test-Driven Development by Example* – Kent Beck
- *Refactoring to Patterns* – Joshua Kerievsky
- *Agile Principles, Patterns, and Practices in C#* – Robert C. Martin & Micah Martin
- *Code Complete* – Steve McConnell
- *Beautiful Testing* – Edited by Tim Riley & Adam Goucher

# What Makes Me Faster?

- Confirming Functionality
- Checking Regression
- Pinpointing Bugs
- Documenting Functionality



Thank You!

Jeremy Clark

- <http://www.jeremybytes.com>
- [jeremy@jeremybytes.com](mailto:jeremy@jeremybytes.com)
- @jeremybytes