# Creating Clear and Simple Tests That Emphasize Readability



Rusty Divine
SOFTWARE CONSULTANT

@CornerPosts cornerpostsoftware.com



# Core Concepts



Make unit tests short and clean

Clearly explain expectations and assumptions

Become a star tester with these simple tips!



## Demo



Code review of poorly named tests

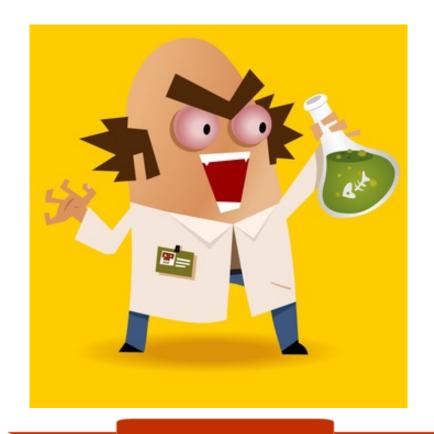


Read code >> Write code

Consistent, meaningful names

Clear and simple tests

Precise test scenarios



**Your Successor** 



# Make Test Names Consistent and Meaningful



# Conventions in Signage









# Three Part Naming

Unit of work Initial condition **Expected result** 





UnitofWork\_InitialCondition\_ExpectedResult
UserLogsIn\_WithValidCredentials\_RedirectsToHome
UserLogsIn\_WithBadPassword\_ReturnsError
UserLogsIn\_FailsThreeTimes\_LocksOutAccount

Scan test names quickly

Same unit of work sorts together

Read like business rules



#### Demo



Naming tests consistently

Creating a unit test template



# Create Clear and Simple Tests



I make a small change to the code

I just broke 50 tests with one small change

This test is 100 lines of code

What is this test even testing?





# Test Organization

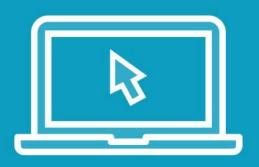
Dry and damp

Arrange, act, assert

Fluent assertions



#### Demo



Arrange, act, assert pattern

Factory and builder methods for dependencies and data

Fluent Assertions for improved readability



### Precise Test Scenarios



```
Public void ReviewSaveTest()
..snip..
Assert.IsNotNull(result);
Assert.AreEqual(3, result.Count);
Assert.IsNotNull(input.DateUpdated);
Assert.AreNotEqual(input.DateUpdated,
input.DateCreated);
Mock.Assert(() =>
_reviewDal.Submit(), Occurs.Once());
```

 Asserting too many expectations makes it harder to find why the test actually failed

```
SavingReview_WhenValid_ReturnsReviews {
Assert.IsNotNull(result);
Assert.AreEqual(3, result.Count);
SavingReview_WhenValid_SetsDateUpdate {
Assert.IsNotNull(input.DateUpdated);
Assert.AreNotEqual(input.DateUpdated,
input.DateCreated);
SavingReview_WhenValid_CallsReviewSubmit {
Mock.Assert(() => _reviewDal.Submit(),
Occurs.Once());
```

■ Three separate tests improve precision

## High Precision

Test one expectation per test

Multiple asserts on same object can be OK

Test should point to precise location of problem



## Summary



Consistency improves readability

Organization improves maintainability

Precision improves fidelity

