Programming with GUTs

@KevlinHenney

kevlin@curbralan.com


WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
97
Collective Wisdom from the Experts
97 Things Every Programmer Should Know
O'REILLY
Edited by Kevlin Henney

When you write unit tests, TDD-style or after your development, you scrutinize, you think, and often you prevent problems without even encountering a test failure.

Michael Feathers
"The Flawed Theory Behind Unit Testing"
http://michaelfeathers.typepad.com/michael_feathers_blog/2008/06/the-flawed-theo.html

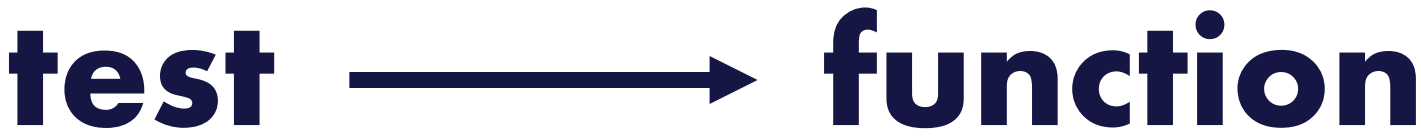
Very many people say "TDD" when they really mean, "I have good unit tests" ("I have GUTs"?). Ron Jeffries tried for years to explain what this was, but we never got a catch-phrase for it, and now TDD is being watered down to mean GUTs.

Alistair Cockburn
"The modern programming professional has GUTs"
http://alistair.cockburn.us/The+modern+programming+professional+has+GUTs

```
size_t ints_to_csv(
    const int * to_write, size_t how_many,
    char * output, size_t length);
```

```
size_t ints_to_csv(
    const int * to_write, size_t how_many, char * output, size_t length)
{
    size_t result = 0;
    if(length != 0)
    {
        if(how_many == 0)
        {
            output[0] = '\0';
        }
        else
        {
            for(size_t which = 0; which != how_many && result != length; ++which)
            {
                result +=
                    snprintf(
                        output + result, length - result,
                        which == 0 ? "%i" : ",%i",
                        to_write[which]);
            }
            result = result > length - 1 ? length - 1 : result;
        }
    }
    return result;
}
```

```
extern "C" size_t ints_to_csv(
    const int * to_write, size_t how_many, char * output, size_t length)
{
    size_t result = 0;
    if(length != 0)
    {
        output[length - 1] = '\0';
        std::ostrstream buffer(output, length - 1);
        for(size_t which = 0; which != how_many; ++which)
            buffer << (which == 0 ? "" : ",") << to_write[which];
        buffer << std::ends;
        result = std::strlen(output);
    }
    return result;
}
```


test
function

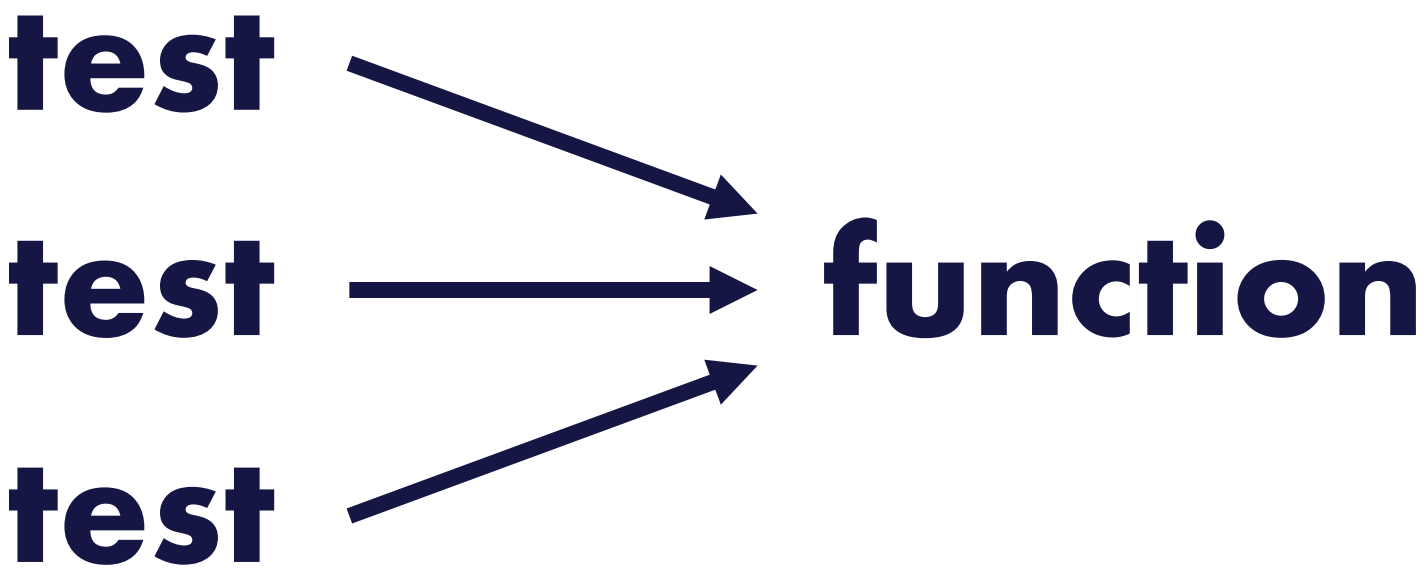
```
void test_ints_to_csv()
{
    size_t written = ints_to_csv(NULL, 0, NULL, 0);
    assert(written == 0);

    const int input[] = { 42 };
    written = ints_to_csv(input, 1, NULL, 0);
    assert(written == 0);

    char output[3] = "+++";
    written = ints_to_csv(NULL, 0, output, sizeof output);
    assert(written == 0);
    assert(output[0] == '\0');

    memcpy(output, "+++", sizeof output);
    written = ints_to_csv(input, 1, output, sizeof output);
    assert(written == 2);
    assert(strcmp(output, "42") == 0);
    ...
}
```

```
void test_ints_to_csv()
{
    // no values from null to null output writes nothing
    size_t written = ints_to_csv(NULL, 0, NULL, 0);
    assert(written == 0);

    // value to null output writes nothing
    const int input[] = { 42 };
    written = ints_to_csv(input, 1, NULL, 0);
    assert(written == 0);

    // no values to sufficient output writes empty
    char output[3] = "+++";
    written = ints_to_csv(NULL, 0, output, sizeof output);
    assert(written == 0);
    assert(output[0] == '\0');

    // positive value to sufficient output writes value without sign
    memcpy(output, "+++", sizeof output);
    written = ints_to_csv(input, 1, output, sizeof output);
    assert(written == 2);
    assert(strcmp(output, "42") == 0);
    ...
}
```

```
void test_ints_to_csv()
{
    // no values from null to null output writes nothing
    {
        size_t written = ints_to_csv(NULL, 0, NULL, 0);
        assert(written == 0);
    }
    // value to null output writes nothing
    {
        const int input[] = { 42 };
        size_t written = ints_to_csv(input, 1, NULL, 0);
        assert(written == 0);
    }
    // no values to sufficient output writes empty
    {
        char output[3] = "+++";
        size_t written = ints_to_csv(NULL, 0, output, sizeof output);
        assert(written == 0);
        assert(output[0] == '\0');
    }
    // positive value to sufficient output writes value without sign
    {
        const int input[] = { 42 };
        char output[3] = "+++";
        size_t written = ints_to_csv(input, 1, output, sizeof output);
        assert(written == 2);
        assert(strcmp(output, "42") == 0);
    }
```

```
void no_values_from_null_to_null_output_writes_nothing()
{
    size_t written = ints_to_csv(NULL, 0, NULL, 0);
    assert(written == 0);
}
void value_to_null_output_writes_nothing()
{
    const int input[] = { 42 };
    size_t written = ints_to_csv(input, 1, NULL, 0);
    assert(written == 0);
}
void no_values_to_sufficient_output_writes_empty()
{
    char output[3] = "+++";
    size_t written = ints_to_csv(NULL, 0, output, sizeof output);
    assert(written == 0);
    assert(output[0] == '\0');
}
void positive_value_to_sufficient_output_writes_value_without_sign()
{
    const int input[] = { 42 };
    char output[3] = "+++";
    size_t written = ints_to_csv(input, 1, output, sizeof output);
    assert(written == 2);
    assert(strcmp(output, "42") == 0);
}
void negative_value_to_sufficient_output_writes_value_with_sign()
{
    const int input[] = { -42 };
    char output[4] = "++++";
    size_t written = ints_to_csv(input, 1, output, sizeof output);
    assert(written == 3);
    assert(strcmp(output, "-42") == 0);
}
void value_to_insufficient_output_writes_truncated_value()
{
    const int input[] = { 42 };
    char output[2] = "++";
    size_t written = ints_to_csv(input, 1, output, sizeof output);
    assert(written == 1);
    assert(strcmp(output, "4") == 0);
}

void multiple_values_to_sufficient_output_writes_comma_separated_values()
{
    const int input[] = { 42, -273, 0, 7 };
    char output[12] = "++++++++++++";
    size_t written = ints_to_csv(input, 4, output, sizeof output);
    assert(written == 11);
    assert(strcmp(output, "42,-273,0,7") == 0);
}
void multiple_values_to_insufficient_output_writes_truncated_value_sequence()
{
    const int input[] = { 42, -273, 0, 7 };
    char output[9] = "+++++++++";
    size_t written = ints_to_csv(input, 4, output, sizeof output);
    assert(written == 8);
    assert(strcmp(output, "42,-273,") == 0);
}
```

```
void no_values_from_null_to_null_output_writes_nothing()
{
    ...
}
void value_to_null_output_writes_nothing()
{
    ...
}
void no_values_to_sufficient_output_writes_empty()
{
    ...
}
void positive_value_to_sufficient_output_writes_value_without_sign()
{
    ...
}
void negative_value_to_sufficient_output_writes_value_with_sign()
{
    ...
}
void value_to_insufficient_output_writes_truncated_value()
{
    ...
}
void multiple_values_to_sufficient_output_writes_comma_separated_values()
{
    ...
}
void multiple_values_to_insufficient_output_writes_truncated_value_sequence()
{
    ...
}
```

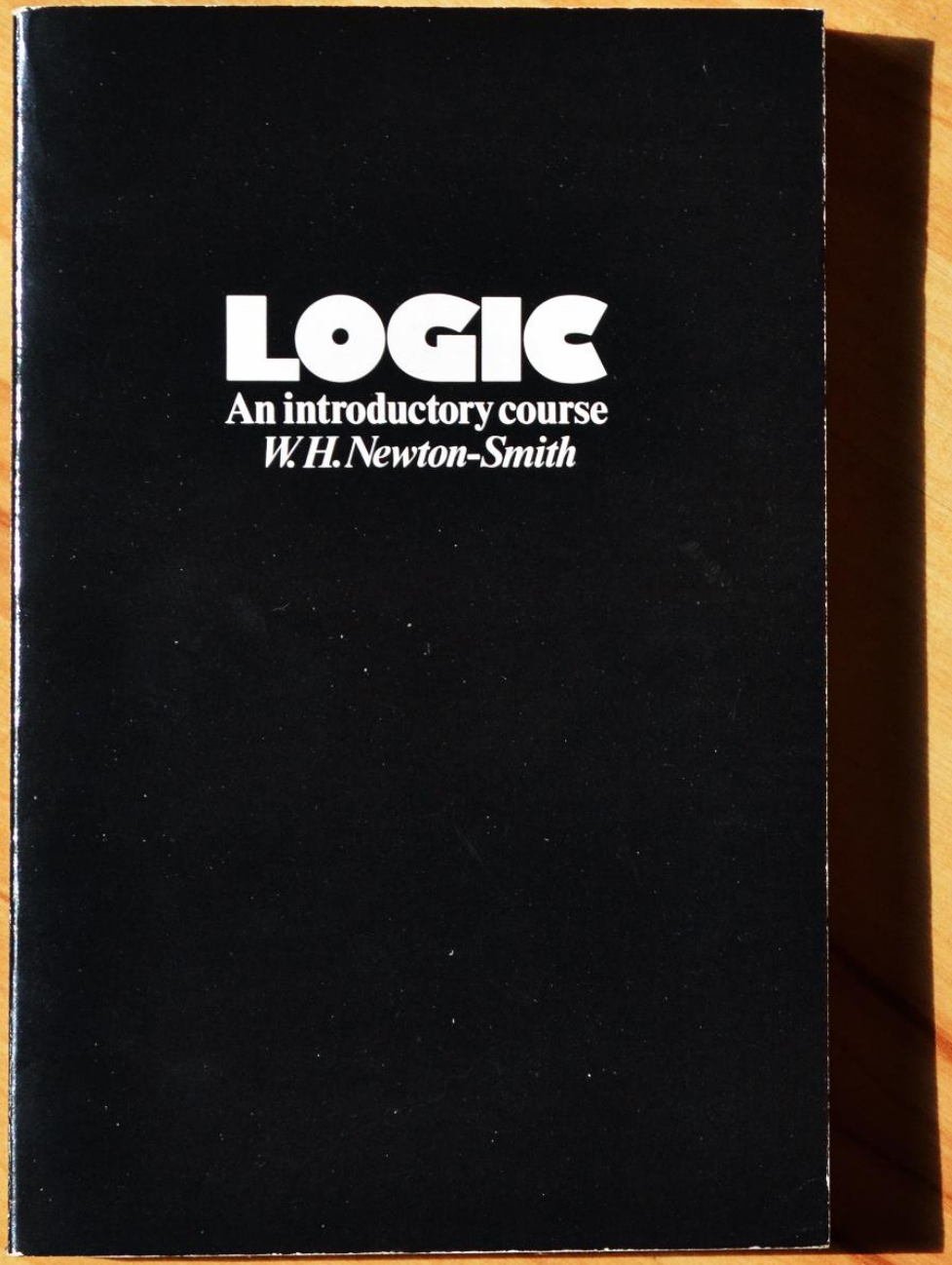

test
test
test
function

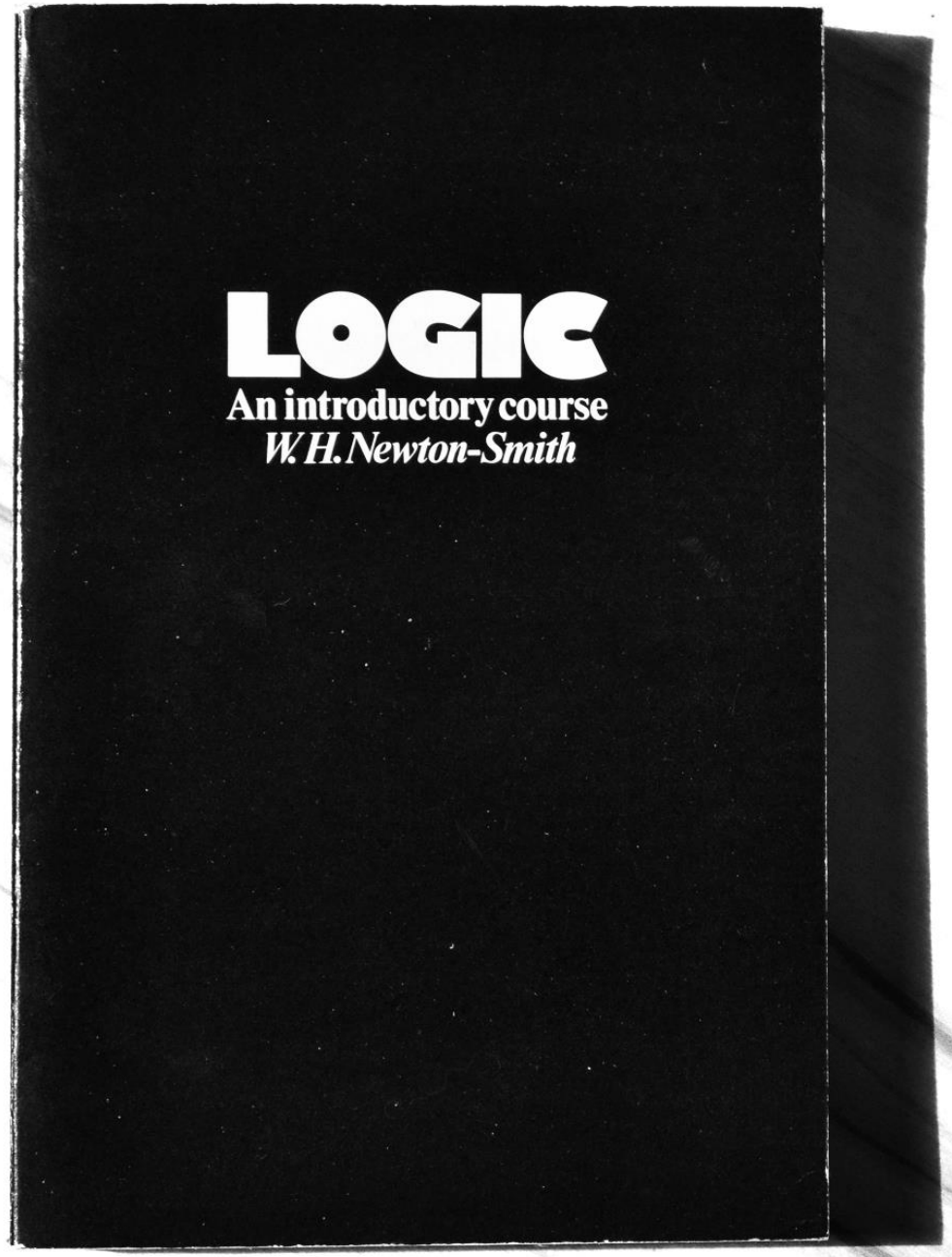
```
size_t ints_to_csv(
    const int * to_write, size_t how_many,
    char * output, size_t length);
```

- *No values from null to null output writes nothing*
- *Value to null output writes nothing*
- *No values to sufficient output writes empty*
- *Positive value to sufficient output writes value without sign*
- *Negative value to sufficient output writes value with sign*
- *Value to insufficient output writes truncated value*
- *Multiple values to sufficient output writes comma separated values*
- *Multiple values to insufficient output writes truncated value sequence*

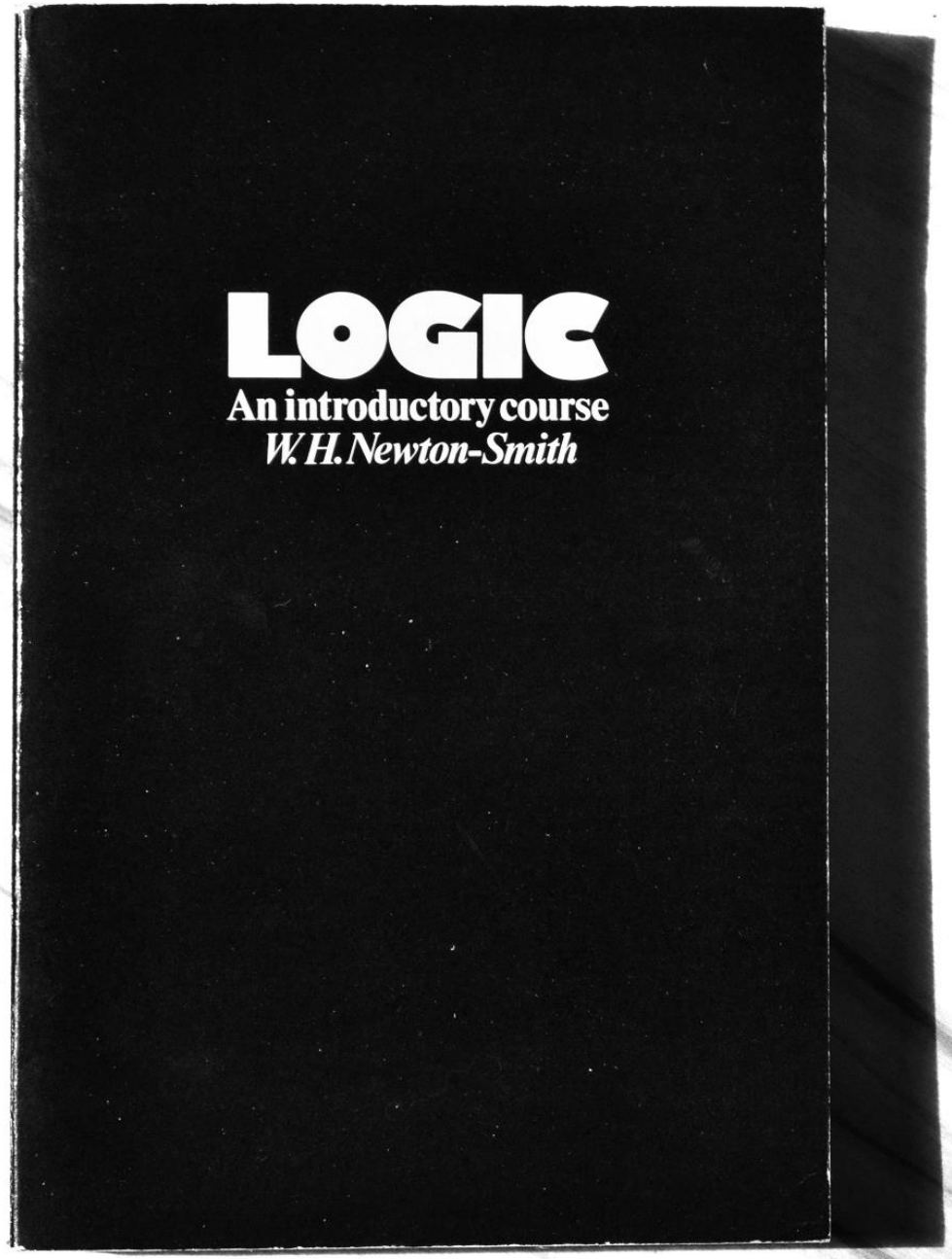
Tests that are not written with their role as specifications in mind can be very confusing to read. The difficulty in understanding what they are testing can greatly reduce the velocity at which a codebase can be changed.

Nat Pryce and Steve Freeman
"Are Your Tests Really Driving Your Development?"


LOGIC
An introductory course
W. H. Newton-Smith



Propositions are vehicles for stating how things are or might be.



Thus only indicative sentences which it makes sense to think of as being true or as being false are capable of expressing propositions.

```
public static boolean isLeapYear(int year) ...
```

```
yearsNotDivisibleBy4...

yearsDivisibleBy4ButNotBy100...

yearsDivisibleBy100ButNotBy400...

yearsDivisibleBy400...
```

Years_not_divisible_by_4_...

Years_divisible_by_4_but_not_by_100_...

Years_divisible_by_100_but_not_by_400_...

Years_divisible_by_400_...

Years_not_divisible_by_4_should_not_be_leap_years

Years_divisible_by_4_but_not_by_100_should_be_leap_years

Years_divisible_by_100_but_not_by_400_should_not_be_leap_years

Years_divisible_by_400_should_be_leap_years

Kevlin Henney
@KevlinHenney

Follow

Test names should reflect outcome not aspiration: doesn't make sense to see "X should give Y" as a result; on passing, result is "X gives Y"

2:22 PM - 27 Jun 2013

16 RETWEETS **7** FAVORITES

https://twitter.com/KevlinHenney/status/350242801868484608

Years_not_divisible_by_4_are_not_leap_years

Years_divisible_by_4_but_not_by_100_are_leap_years

Years_divisible_by_100_but_not_by_400_are_not_leap_years

Years_divisible_by_400_are_not_leap_years

Years_not_divisible_by_4_are_not_leap_years

Years_divisible_by_4_but_not_by_100_are_leap_years

Years_divisible_by_100_but_not_by_400_are_not_leap_years

Years_divisible_by_400_are_not_leap_years

Years_not_divisible_by_4_are_not_leap_years

Years_divisible_by_4_but_not_by_100_are_leap_years

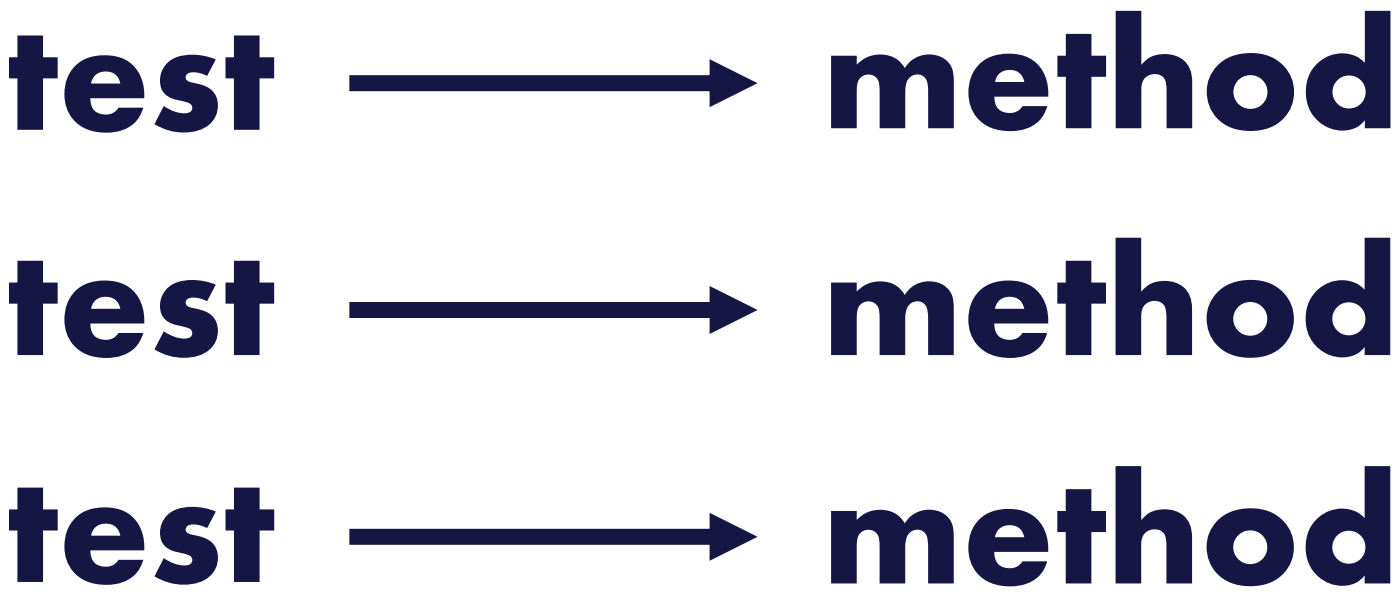
Years_divisible_by_100_but_not_by_400_are_not_leap_years

Years_divisible_by_400_are_not_leap_years

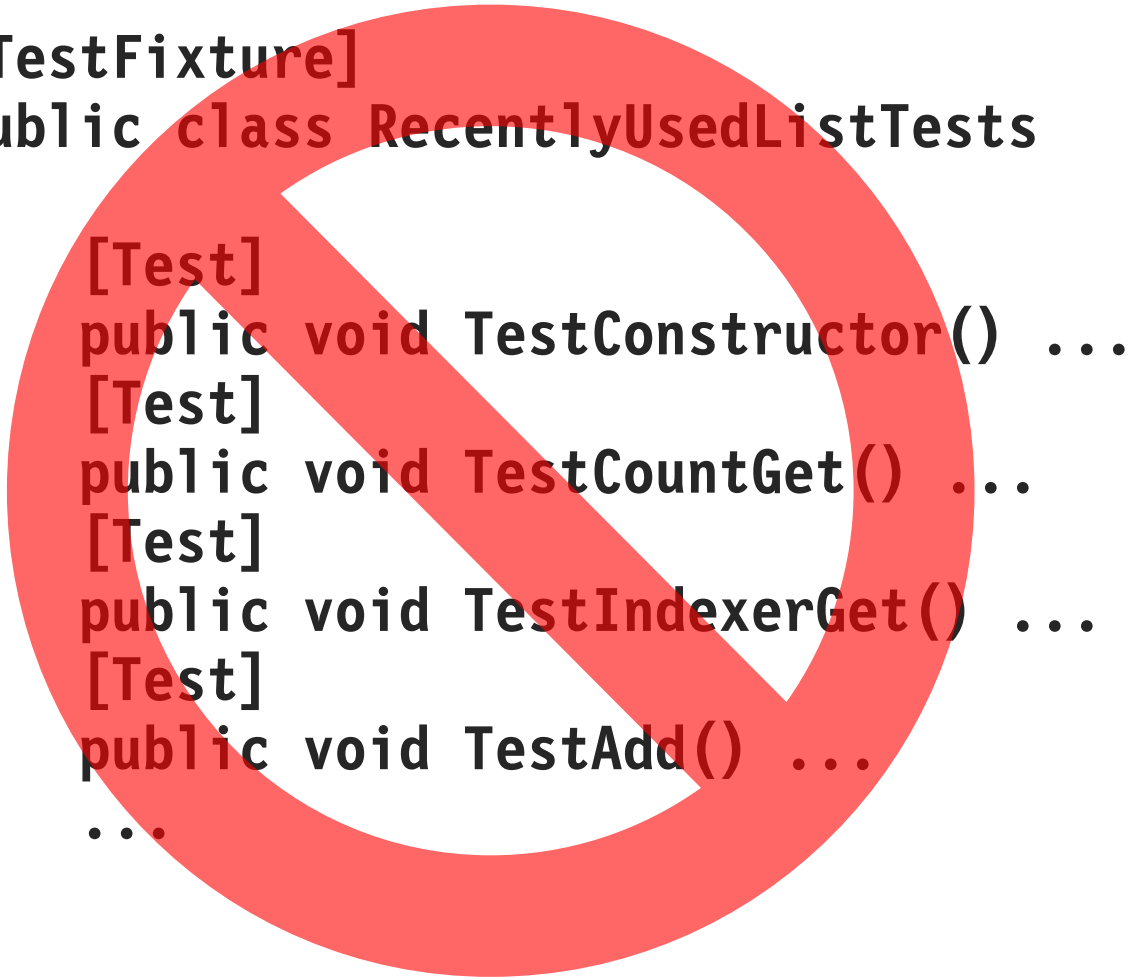
A test case should be just that: it should correspond to a single case.

```
public class Leap_year_spec
{
    public static class A_year_is_a_leap_year
    {
        @Test public void If_it_is_divisible_by_4_but_not_by_100() ...
        @Test public void If_it_is_divisible_by_400() ...
    }
    public static class A_year_is_not_a_leap_year
    {
        @Test public void If_it_is_not_divisible_by_4() ...
        @Test public void If_it_is_divisible_by_100_but_not_by_400() ...
    }
}
```

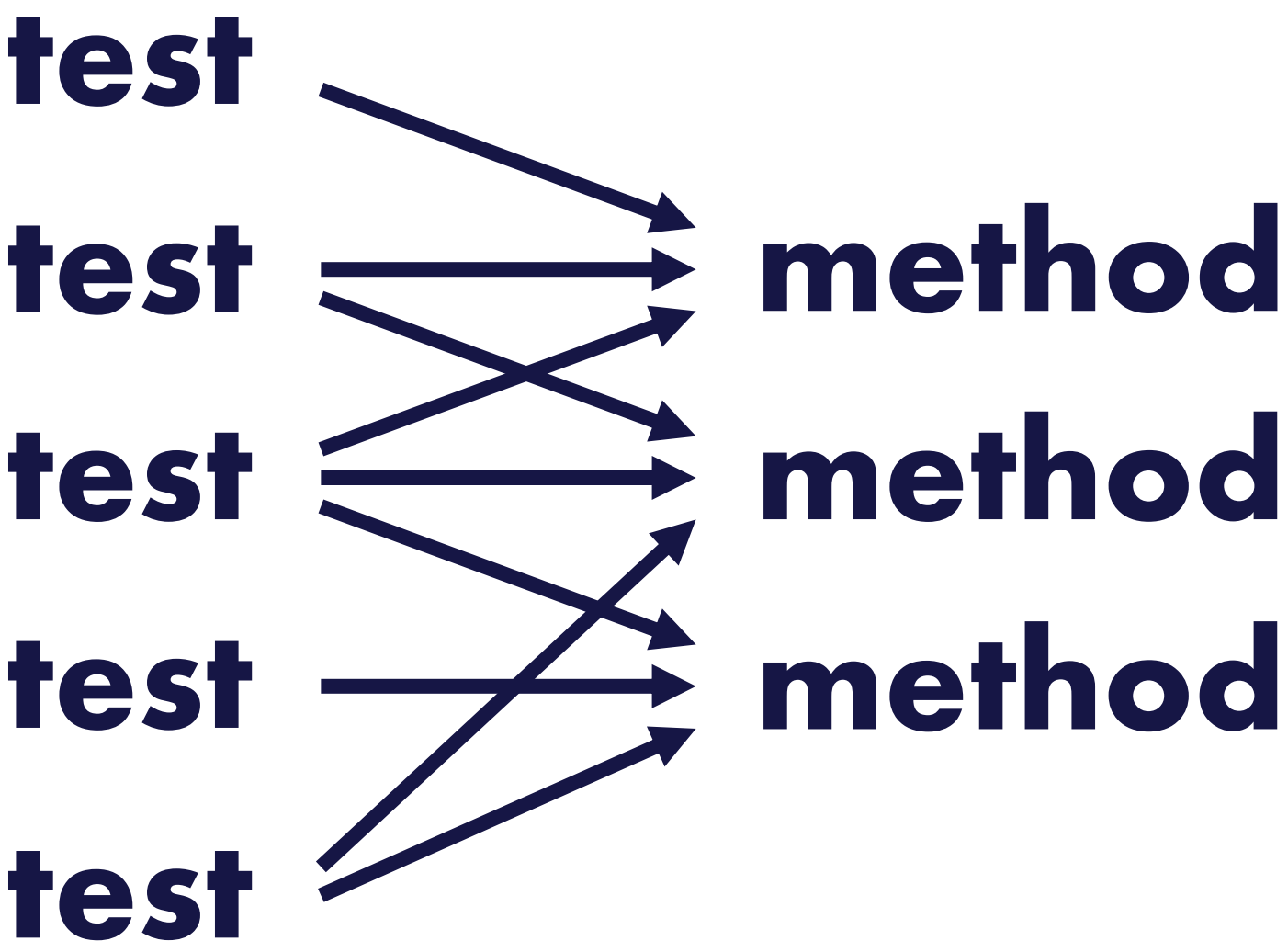
```
public class Leap_year_spec
{
    public static class A_year_is_a_leap_year
    {
        @Test public void If_it_is_divisible_by_4_but_not_by_100() ...
        @Test public void If_it_is_divisible_by_400() ...
    }
    public static class A_year_is_not_a_leap_year
    {
        @Test public void If_it_is_not_divisible_by_4() ...
        @Test public void If_it_is_divisible_by_100_but_not_by_400() ...
    }
}
```


test
method
test
method
test
method

```
public class RecentlyUsedList
{
    ...
    public RecentlyUsedList() ...
    public int Count
    {
        get...
    }
    public string this[int index]
    {
        get...
    }
    public void Add(string newItem) ...
    ...
}
```

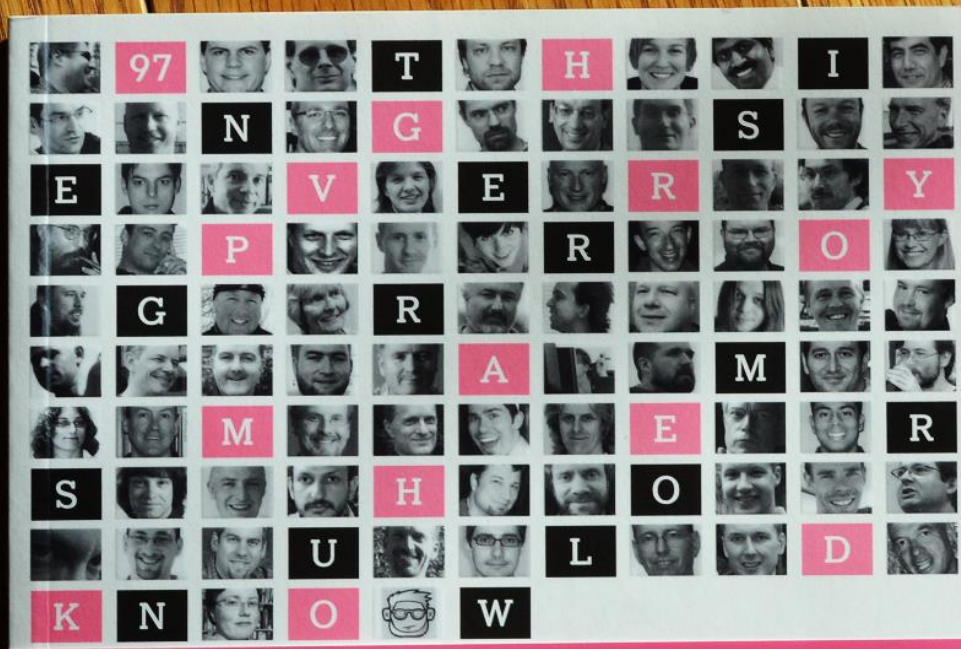


```
[TestFixture]
public class RecentlyUsedListTests
{
    [Test]
    public void TestConstructor() ...
    [Test]
    public void TestCountGet() ...
    [Test]
    public void TestIndexerGet() ...
    [Test]
    public void TestAdd() ...
    ...
}
```


test
test
test
test
test
method
method
method

```
namespace RecentlyUsedList_spec
{
    [TestFixture]
    public class A_new_list
    {
        [Test] public void Is_empty() ...
    }

    [TestFixture]
    public class An_empty_list
    {
        [Test] public void Retains_a_single_addition() ...
        [Test] public void Retains_unique_additions_in_stack_order() ...
    }

    [TestFixture]
    public class A_non_empty_list
    {
        [Test] public void Is_unchanged_when_head_item_is_readded() ...
        [Test] public void Moves_non_head_item_to_head_when_it_is_readded() ...
    }

    [TestFixture]
    public class Any_list
    {
        [Test] public void Rejects_addition_of_null_items() ...
        [Test] public void Rejects_indexing_past_its_end() ...
        [Test] public void Rejects_negative_indexing() ...
    }
}
```

```
namespace RecentlyUsedList_spec
{
    [TestFixture]
    public class A_new_list
    {
        [Test] public void Is_empty() ...
    }

    [TestFixture]
    public class An_empty_list
    {
        [Test] public void Retains_a_single_addition() ...
        [Test] public void Retains_unique_additions_in_stack_order() ...
    }

    [TestFixture]
    public class A_non_empty_list
    {
        [Test] public void Is_unchanged_when_head_item_is_readded() ...
        [Test] public void Moves_non_head_item_to_head_when_it_is_readded() ...
    }

    [TestFixture]
    public class Any_list
    {
        [Test] public void Rejects_addition_of_null_items() ...
        [Test] public void Rejects_indexing_past_its_end() ...
        [Test] public void Rejects_negative_indexing() ...
    }
}
```



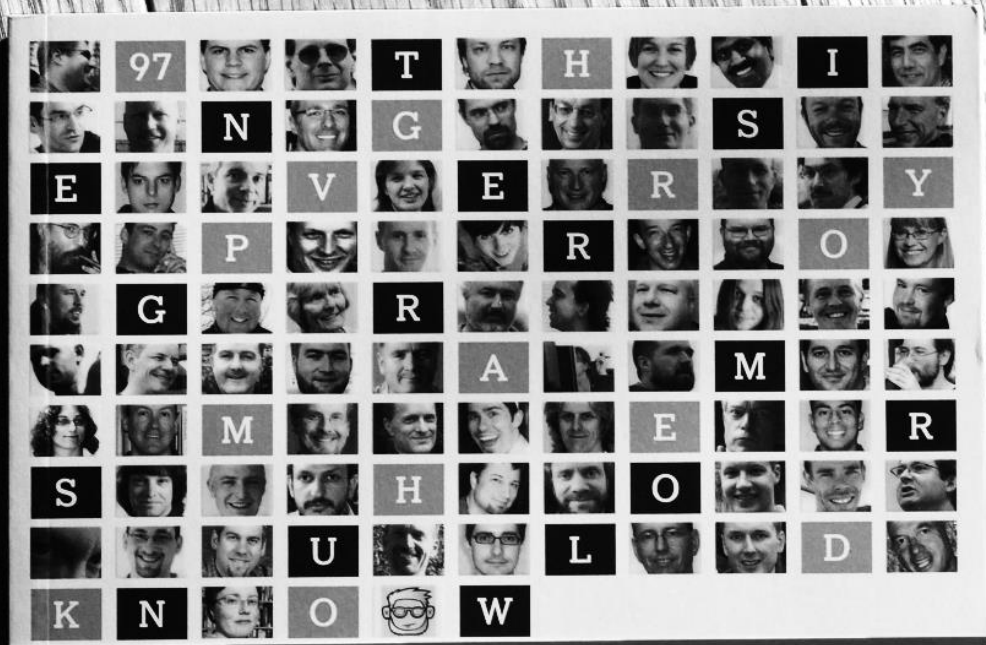
프로그래머가 알아야 할 97 가지

Collective Wisdom from the Experts

"프로그래머의, 프로그래머에 의한, 프로그래머를 위한 책!!"

이 땅의 모든 프로그래머에게 전하는 현장의 지혜를 담은 107가지 의미있는 이야기

Kevlin Henney 엮음 / 손영수, 김수현, 최현미 외 옮김



Gerard Meszaros
"Write Tests for People"

So who should you be writing the tests for? For the person trying to understand your code.

Good tests act as documentation for the code they are testing. They describe how the code works. For each usage scenario, the test(s):

- **Describe the context, starting point, or preconditions that must be satisfied**
- **Illustrate how the software is invoked**
- **Describe the expected results or postconditions to be verified**

```
namespace RecentlyUsedList_spec
{
    [TestFixture]
    public class A_new_list ...

    [TestFixture]
    public class An_empty_list
    {
        [Test]
        public void Retains_a_single_addition(
            [Values("Prague", "Oslo", "Bristol")] string addend)
        {
            var items = new RecentlyUsedList();   // Given...

            items.Add(addend);                    // When...

            Assert.AreEqual(1, items.Count);      // Then...
            Assert.AreEqual(addend, list[0]);
        }
        [Test] public void Retains_unique_additions_in_stack_order() ...
    }

    [TestFixture]
    public class A_non_empty_list ...

    [TestFixture]
    public class Any_list ...
}
```

Less unit testing dogma.
More unit testing karma.

Alberto Savoia
"The Way of Testivus"
http://www.artima.com/weblogs/viewpost.jsp?thread=203994