# The Pros and Cons of White-Box Testing


**Zoran Horvat**

PRINCIPAL CONSULTANT AT CODING HELMET

@zoranh75 csharpmentor.com

```
public void AddTargetPoints(IMyList toList, int count)
{
    if (count < 0)
    {
        throw new ArgumentException($"{nameof(count)} < 0");
    }

    for (int i = 0; i < count; i++)
    {
        toList.Append(3 + 2*i);
    }
}
```

# White-box Testing Defined

**Testing internal structure of the method**

**Cover all distinct paths through the method**

**Cover each of the execution branches in the method**

```
public void AddTargetPoints(IMyList toList, int count)
{
    if (count < 0)
    {
        throw new ArgumentException($"{nameof(count)} < 0");
    }

    for (int i = 0; i < count; i++)
    {
        toList.Append(3 + 2*i);
    }
}
```

2 branches

2 branches

# Path vs. Branch Coverage

**Total number of paths grows exponentially**

$2 \times 2 \times 2 \times \ldots = 2^N$ paths

**Total number of branches grows linearly**

$2 + 2 + 2 + \ldots = 2N$ branches

```
public void AddTargetPoints(IMyList toList, int count)
{
    if (count < 0)
    {
        throw new ArgumentException($"{nameof(count)} < 0");
    }

    for (int i = 0; i < count; i++)
    {
        toList.Append(3 + 2*i);
    }
}
```

1. count = -17 – throws
1a. count = -1 – throws

2. count = 5 – doesn't throw
2a. count = 0 – doesn't throw

3. count = 0 – no elements added

4. count = 17 – 17 elements added
4a. count = 1 – 1 element added

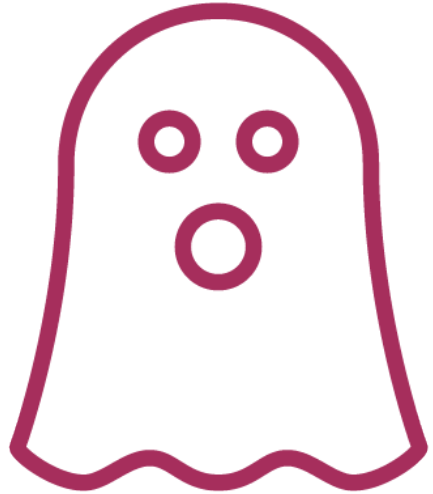
5. count = 216 – list[9] = 21
5a. count = 216 – list[71] = 145
5b. count = 216 – list[0] = 3
5c. count = 216 – list[215] = 433

# The Problem with Isolation Frameworks

They never fail to build

They always try to do something

Test method may pass for no reason

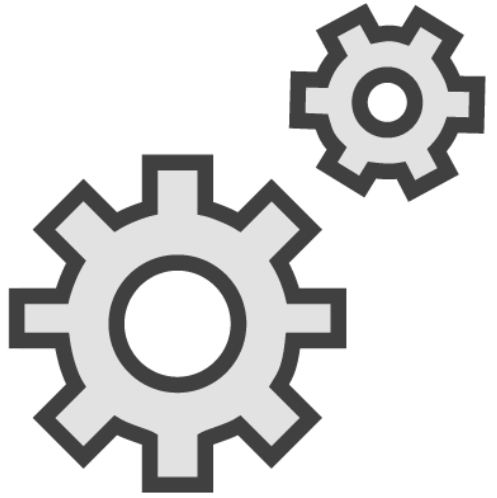


Test which happily passes does not provide sufficient protection

# Testing After Modifying Implementation

Tests affected by the change must raise their hand

Existing tests must not just silently pass



They must really exercise new implementation

Only then can we trust their output

# The Problem with Isolation Frameworks


Automatic mocks will silently incorporate new members

If new member doesn't provide result, the test will pass

That pattern doesn't improve our chances to find bugs

"You may write any random C++ code you like and compiler will produce any random binary it likes."

**Anonymous**

A problem has been detected and Windows has been shut down to prevent damage to your computer.

IRQL_NOT_DISPATCH_LEVEL

If this is the first time you've seen this error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:

*** STOP: 0X00000ed (0X80F128D0, 0xc000009c, 0x00000000, 0x0000000)

Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further assistance.

More Info : https://msdn.microsoft.com/en-us/library/windows/hardware/ff559278(v=vs.85).aspx

For technical support assistance call : 1-855-596-2695 (USA-Canada)

"You may write any random C++ code you like and compiler will produce any random binary it likes."

**Anonymous**

# The Problem with Isolation Frameworks


Test with automatic mock will usually fail after the change

That is false positive

... and it is annoying

The test which just passes may turn into a false negative

# Using Manual Mocks Instead of Automatic


**Build will fail if a new member is missing in the interface**

**But build will succeed if a virtual member was added to base class**

# Using Manual Mocks Instead of Automatic

Try to avoid adding new virtual members

Try to add abstract members instead


That will cause compile errors in all affected places

# Lean on the Compiler

Change code in such way that compilation failures occur precisely in those places that are affected by the change.

**Example:**

Introducing an abstract member causes compile-time failure in all derived classes.

Introducing a virtual member at the same place causes no compile-time errors, but might cause run-time failures.

# Automatic vs. Manual Mocks

**Manual mocks will fail to build when their interface is changed**

Safety net of a compiler over a safety net of automated tests

**Automatic mocks save some work, but make false negatives possible**

Seek balance between added work and added correctness

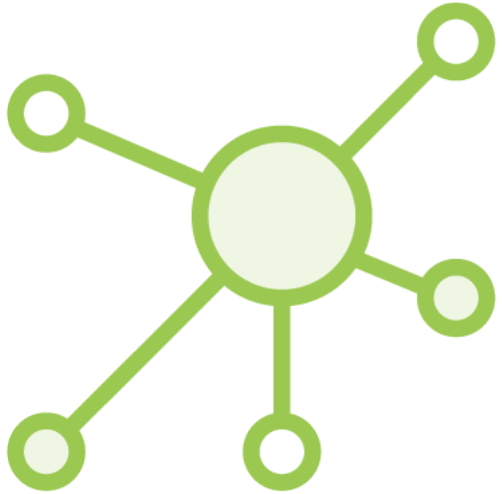
# Interface Segregation Principle Considerations

**Scenario:**

New interface member has no effect on a test

**Observation:**

New interface member is breaking ISP



**Rationale:**

New interface member must be related to interface's role

# Interface Segregation Principle Considerations

**Scenario:**

New interface member affects a test

**Observation:**

New interface member is enforcing ISP

**Consequence:**

Manual mocks will attract our attention

a.k.a. Lean on the Compiler

# Summary

**White-box testing**

- Writing tests for a known segment of code
- Knowing code helps write more complete set of test cases

**Automatic vs. manual mocks**

**State tests were more resilient to change compared to interaction tests**

# Summary

**Lean on the compiler**

- Change the code in such way that related pieces of code fail to build
- Use this technique to locate code correctness of which may be affected

**Practical techniques**

- Favor interfaces over base classes
- Favor abstract methods over virtual

# Summary

**Example:**

- Removing dependency on system time
- Software in which we cannot substitute system time is impossible to test

**Refactoring and tests**

- Use refactoring techniques to make a class testable

# Summary

**More practical techniques**

- Branch coverage technique
  - Write one test case for each execution branch
- Boundary tests
  - Add test cases for boundary conditions

**Next module:**

*Modeling dependencies in tests*