MAINTAINABLE CODE

Write Better Code

Bevan Arps
bevan@nichesoftware.co.nz

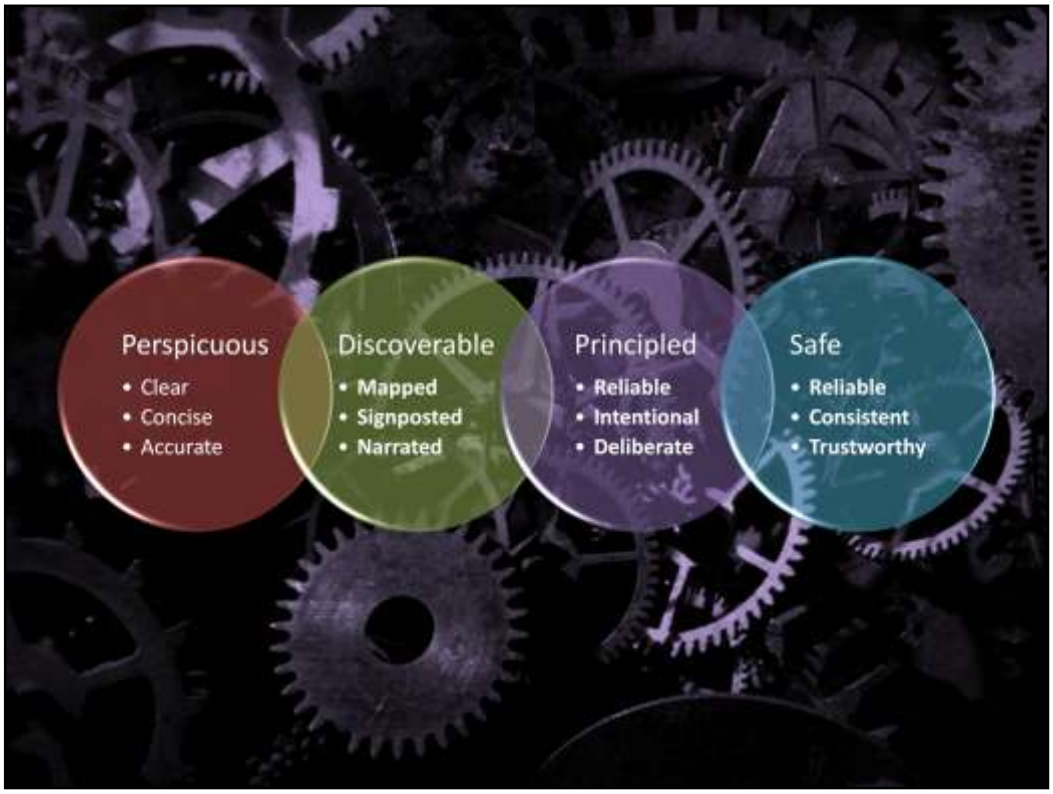
While much of the content is generic
could apply to any platform

I'm assuming a .NET environment,
and where a language is necessary, C#

Focus on Code, not on other things;
not that other things are irrelevant,
but need to focus somewhere.

Apologies in advance to Visual Basic Developers,
though most stuff will still be relevant

License
This presentation licensed under the
Creative Commons Attribution License
http://creativecommons.org/licenses/by/3.0/


Perspicuous
- Clear
- Concise
- Accurate

Discoverable
- Mapped
- Signposted
- Narrated

Principled
- Reliable
- Intentional
- Deliberate

Safe
- Reliable
- Consistent
- Trustworthy

4 Key Terms

Though not exclusive, these are useful.

Perspicuous
Clarity
Don’t Repeat Yourself

Discoverable
Make it easy to navigate

Principled
Every problem has multiple solutions
All software is opinion

Safe
No booby traps, sinkholes or mazes

Perspicuous Code

Name things well

Naming
of types, of methods, of members, of locals

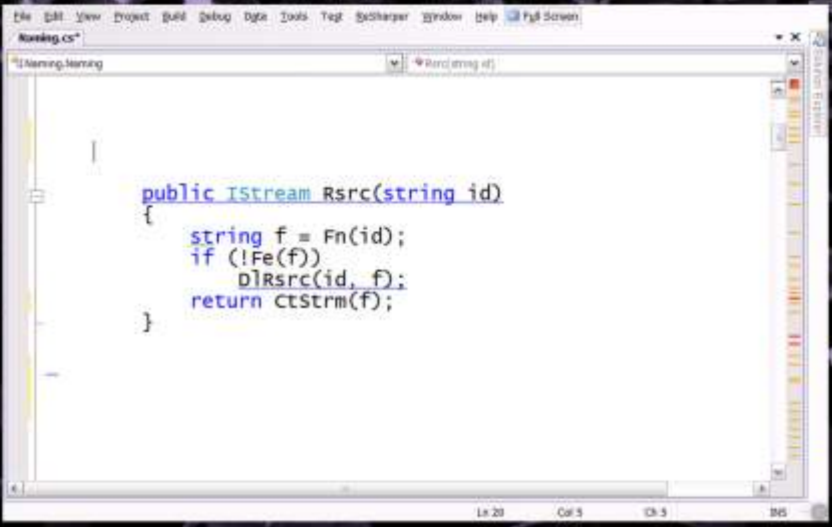
Naming is important.
Clarity, Accuracy, Intelligibility and Reliability

Names are the first aspect encountered by other developers,
may be the most persistent and long lived aspect,
will be present even if everything else is lost

Names need to be clear and accurate (singular vs plural).
Trustworthy
Appropriate to Culture – e.g. “C” prefix in MFC

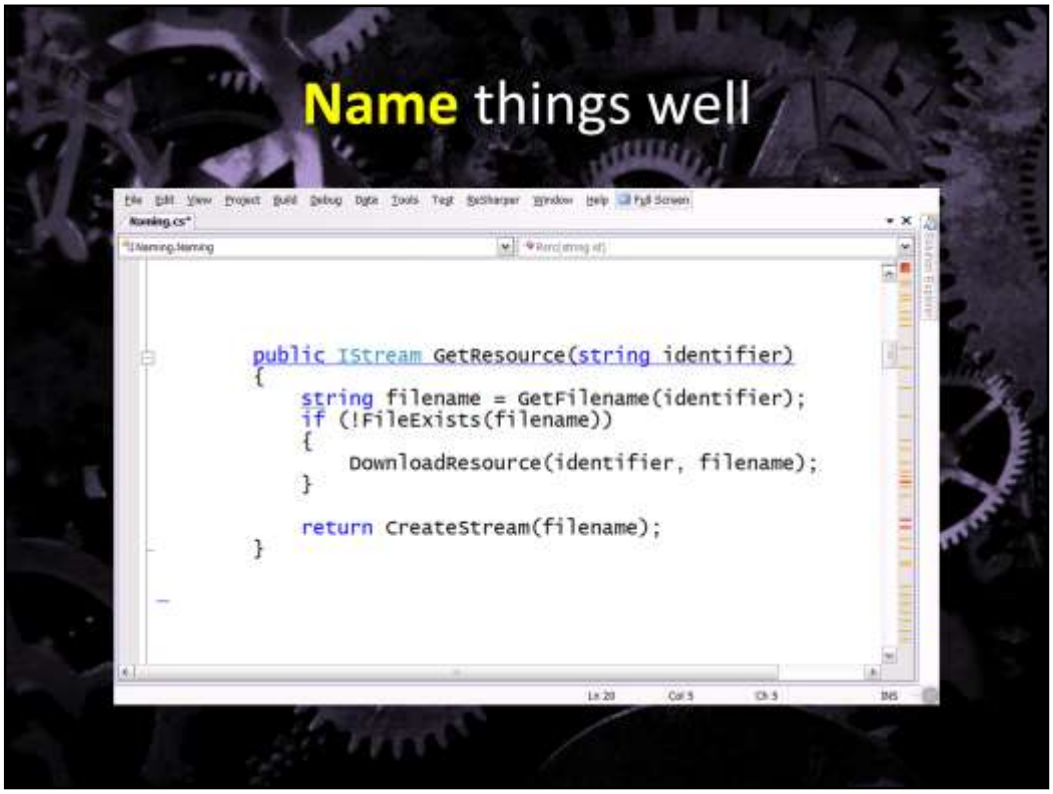
Ubiquitous Vocabulary
Design Pattern Vocabulary
http://www.definr.com

Name things well

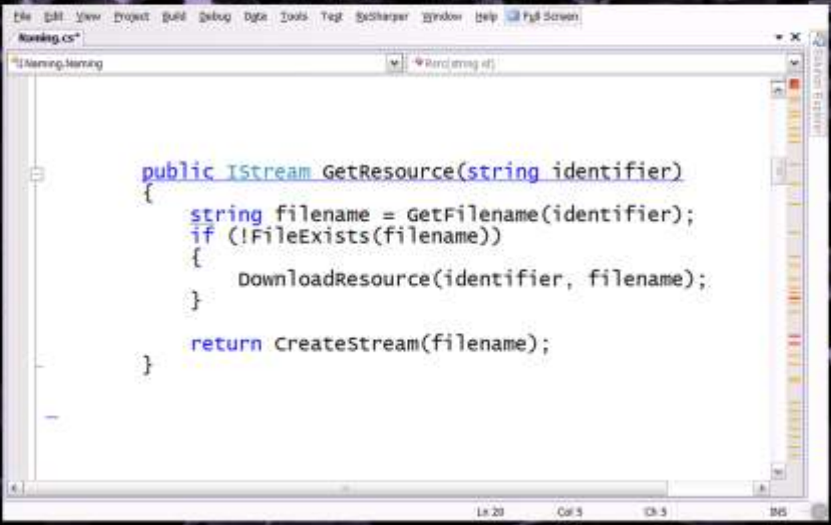

File Edit View Project Build Debug Data Tools Test ReSharper Window Help Full Screen
Naming.cs*
Naming.Naming
Rsrc(string id)
Ln 20 Col 5 Ch 5 INS

```
public IStream Rsrc(string id)
{
    string f = Fn(id);
    if (!Fe(f))
        DlRsrc(id, f);
    return CtStrm(f);
}
```

What does this code do?



Name things well

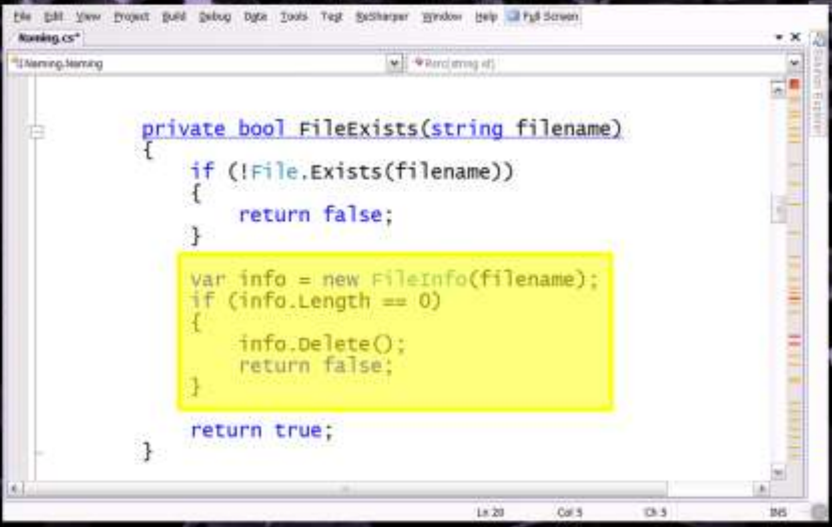


```
public IStream GetResource(string identifier)
{
    string filename = GetFilename(identifier);
    if (!FileExists(filename))
    {
        DownloadResource(identifier, filename);
    }

    return CreateStream(filename);
}
```

This is the same routine as the previous slide.

Note how the improved naming makes
the routine easier to understand.

Name things well



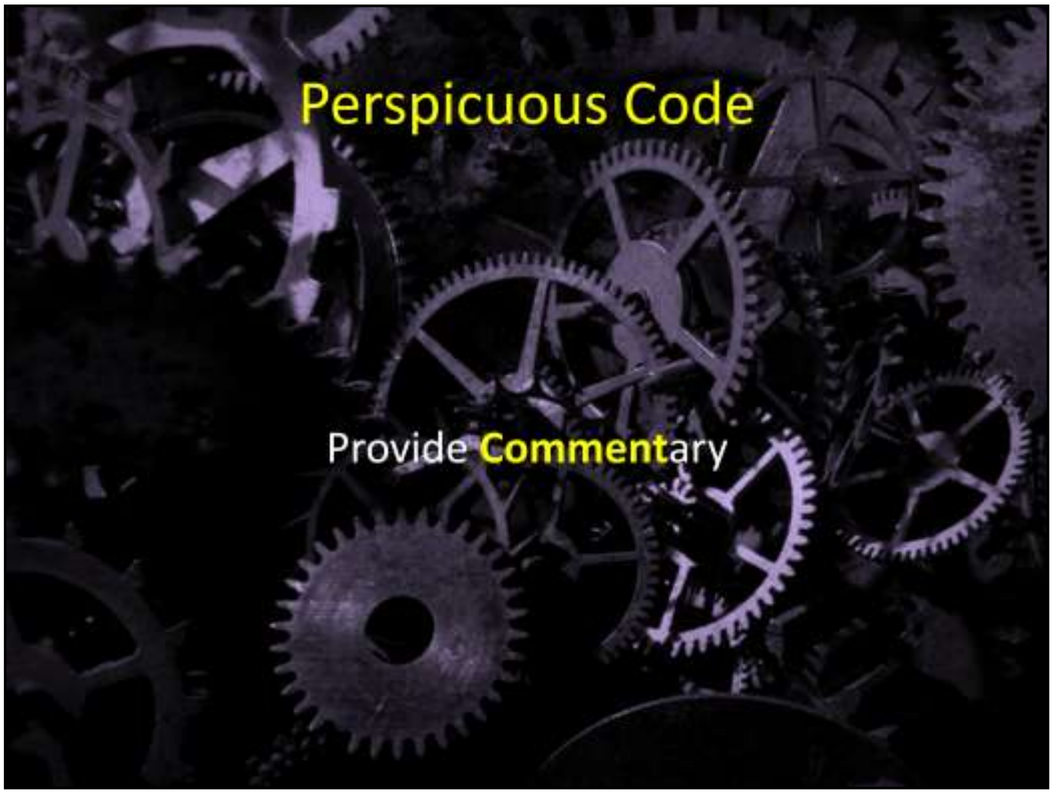
```
private bool FileExists(string filename)
{
    if (!File.Exists(filename))
    {
        return false;
    }

    var info = new FileInfo(filename);
    if (info.Length == 0)
    {
        info.Delete();
        return false;
    }

    return true;
}
```

What behaviour do you expect
from a FileExists() method?

Do you expect the highlighted code?

This one silently deletes zero byte files!?!?

Simulated example, but reproduces a case
discovered in production code.



Perspicuous Code

Provide **Comment**ary

We call talk about commenting our code,
but often the comments are rubbish.

Anyone seen comments that were just plain wrong?

Comments should be intentional, not declarative.

Don’t Repeat Yourself
(the DRY principle)
Comments shouldn’t indicate what is going on,
but why we need to do it.

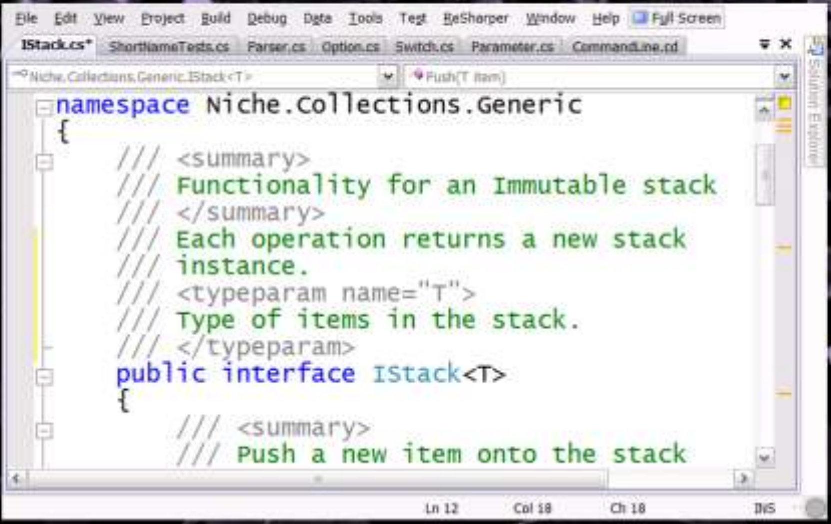
Don’t just write comments that can be worked out by reading the code –
give more information, things that otherwise would be guessed at.

Include references – to documents, websites –
so that others can learn what you know.

If something might need to be improved, leave reminders to later self
If something didn’t work, leave notes so the next person doesn’t repeat the mistake

Not everything needs commenting – good naming helps a lot.

Provide Commentary

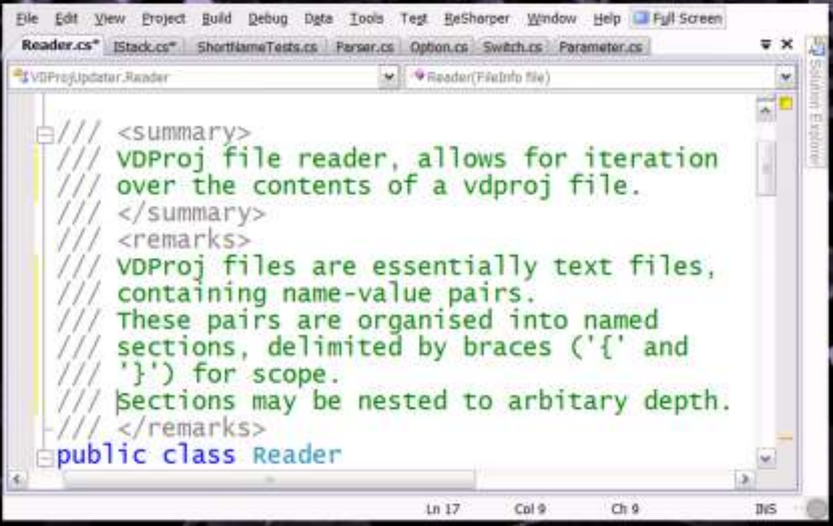

File Edit View Project Build Debug Data Tools Test ReSharper Window Help Full Screen
IStack.cs* ShortNameTests.cs Parser.cs Option.cs Switch.cs Parameter.cs CommandLine.cd
Niche.Collections.Generic.IStack<T>
Push(T item)
Ln 12 Col 18 Ch 18 INS

```
namespace Niche.Collections.Generic
{
    /// <summary>
    /// Functionality for an Immutable stack
    /// </summary>
    /// Each operation returns a new stack
    /// instance.
    /// <typeparam name="T">
    /// Type of items in the stack.
    /// </typeparam>
    public interface IStack<T>
    {
        /// <summary>
        /// Push a new item onto the stack
```

Here's a sample object with some commentary.

Includes key information: **Immutable**

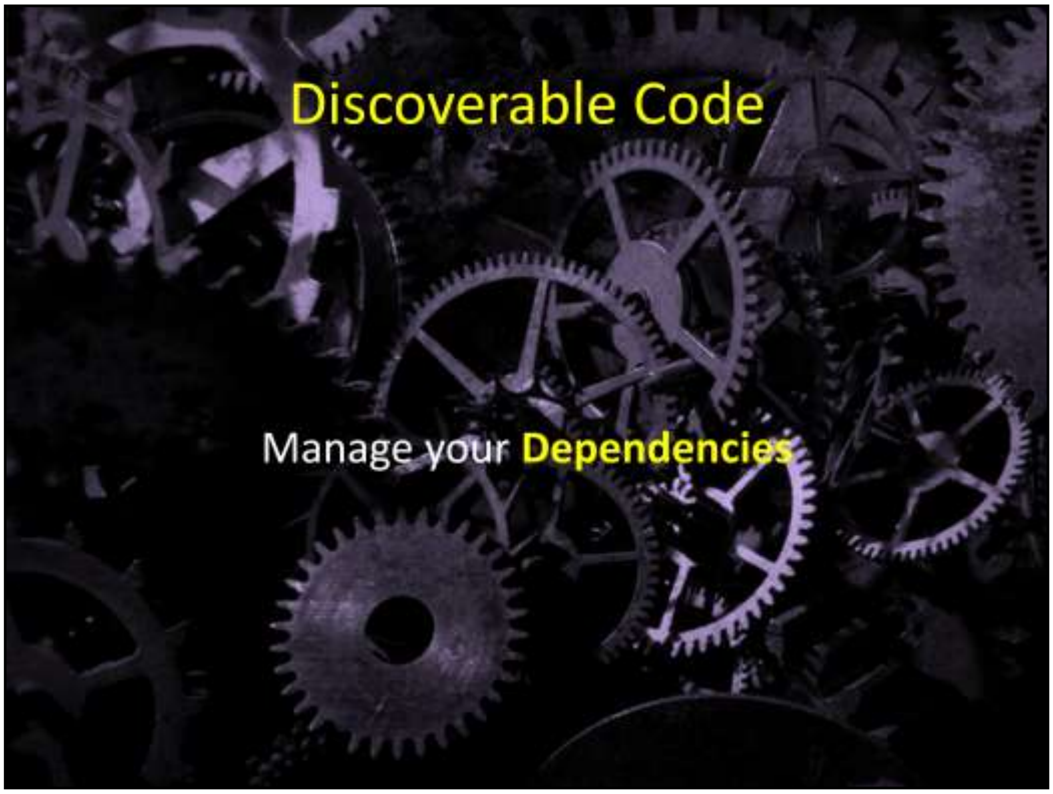
Provide Commentary


File Edit View Project Build Debug Data Tools Test ReSharper Window Help Full Screen
Reader.cs* IStack.cs* ShortNameTests.cs Parser.cs Option.cs Switch.cs Parameter.cs
VDProjUpdater.Reader
Reader(FileInfo file)
Ln 17 Col 9 Ch 9 INS

```
/// <summary>
/// VDProj file reader, allows for iteration
/// over the contents of a vdproj file.
/// </summary>
/// <remarks>
/// VDProj files are essentially text files,
/// containing name-value pairs.
/// These pairs are organised into named
/// sections, delimited by braces ('{' and
/// '}') for scope.
/// Sections may be nested to arbitary depth.
/// </remarks>
public class Reader
```

Another class header

Describes nature of files processed by the reader



Discoverable Code

Manage your **Dependencies**

Every dependency has a weight

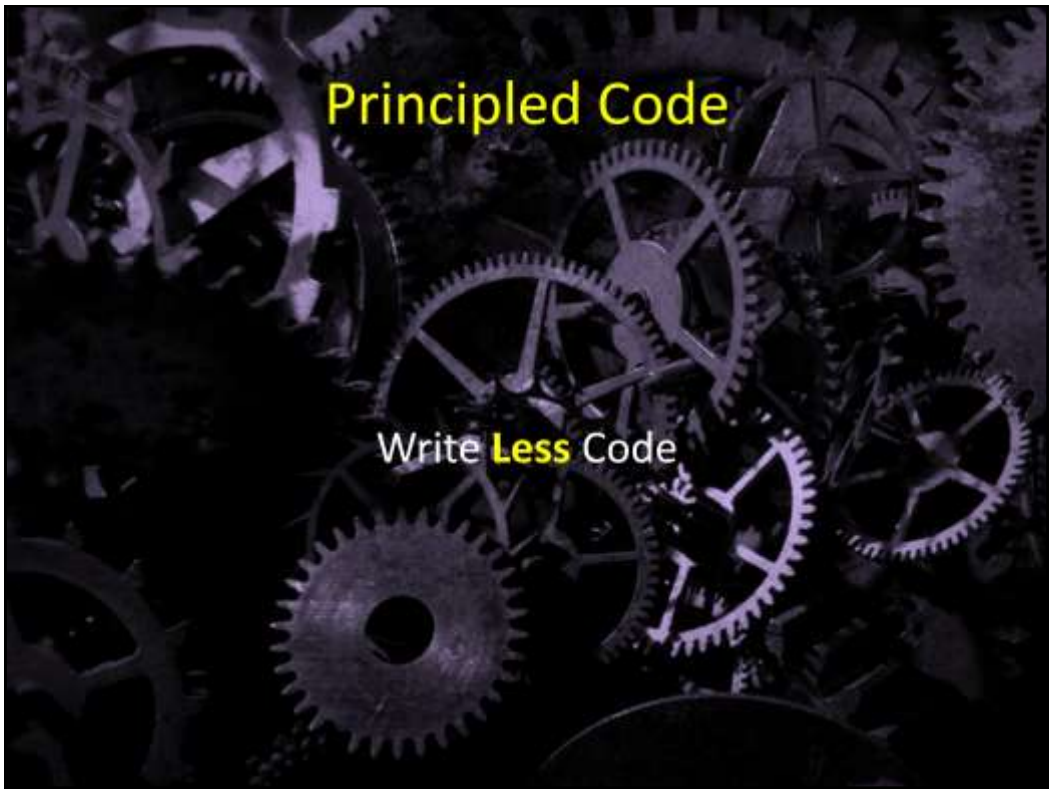
Too many dependencies
Or the wrong dependency
Can weigh down your system

Don’t take on a dependency unless there is real benefit

Important to balance cost verses benefit

Don’t do it for a single method!
(Unless that single method is uniquely valuable and hard to reproduce)

Microsoft Excel Team
Focussed on eliminating dependencies
Reportedly had their own C Compiler!



Principled Code

Write **Less** Code

Don't write any more code than you need to

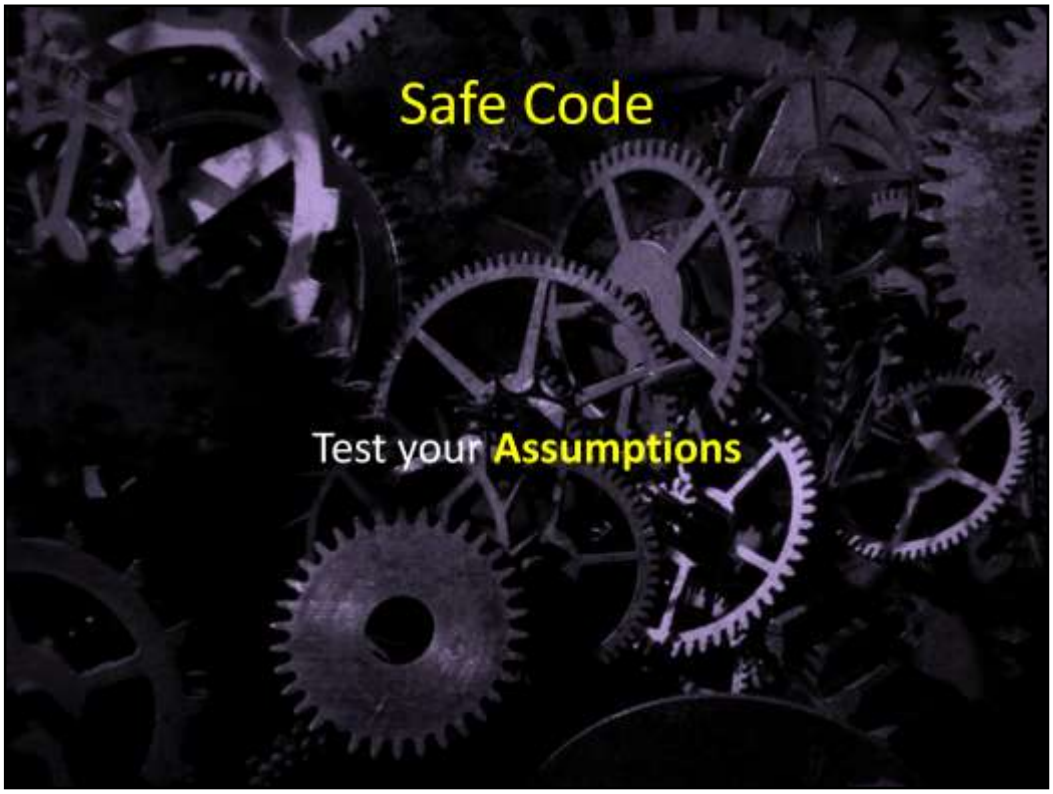
Code that doesn't exist can't have bugs!

Don't stop when it seems to work

Is any of the code extraneous?

Can the code be simplified and still work properly?

Is there an API that delivers much (all?) of what's required?
(Tension with Managing Dependencies)



Safe Code

Test your **Assumptions**

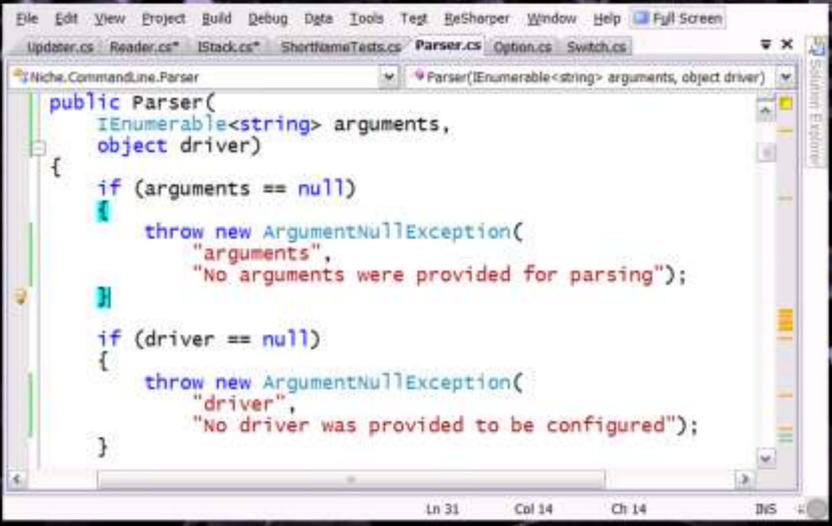
Code doesn’t exist in isolation

There are always assumptions

Make the assumptions explicit on your code

Where possible, write tests and throw exceptions

Test your Assumptions


File Edit View Project Build Debug Data Tools Test ReSharper Window Help Full Screen
Updater.cs Reader.cs* IStack.cs* ShortNameTests.cs Parser.cs Option.cs Switch.cs
Niche.CommandLine.Parser
Parser(IEnumerable<string> arguments, object driver)
Ln 31 Col 14 Ch 14 INS

```
public Parser(
    IEnumerable<string> arguments,
    object driver)
{
    if (arguments == null)
    {
        throw new ArgumentNullException(
            "arguments",
            "No arguments were provided for parsing");
    }

    if (driver == null)
    {
        throw new ArgumentNullException(
            "driver",
            "No driver was provided to be configured");
    }
```

Example of tests to ensure constructor prerequisites are satisfied.

Could also use Code contracts.

Maintainable Code

Any **Questions?**

Bevan Arps

bevan@nichesoftware.co.nz
http://www.nichesoftware.co.nz