Clean Code: Writing Code for Humans

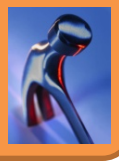
Principles for Cleanliness

Cory House
BitNative.com
Twitter: @housecor

pluralsight
hardcore developer training

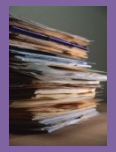
Three Principles for Clean Code

1) Right tool for the job



2) High signal to noise ratio

3) Self-documenting

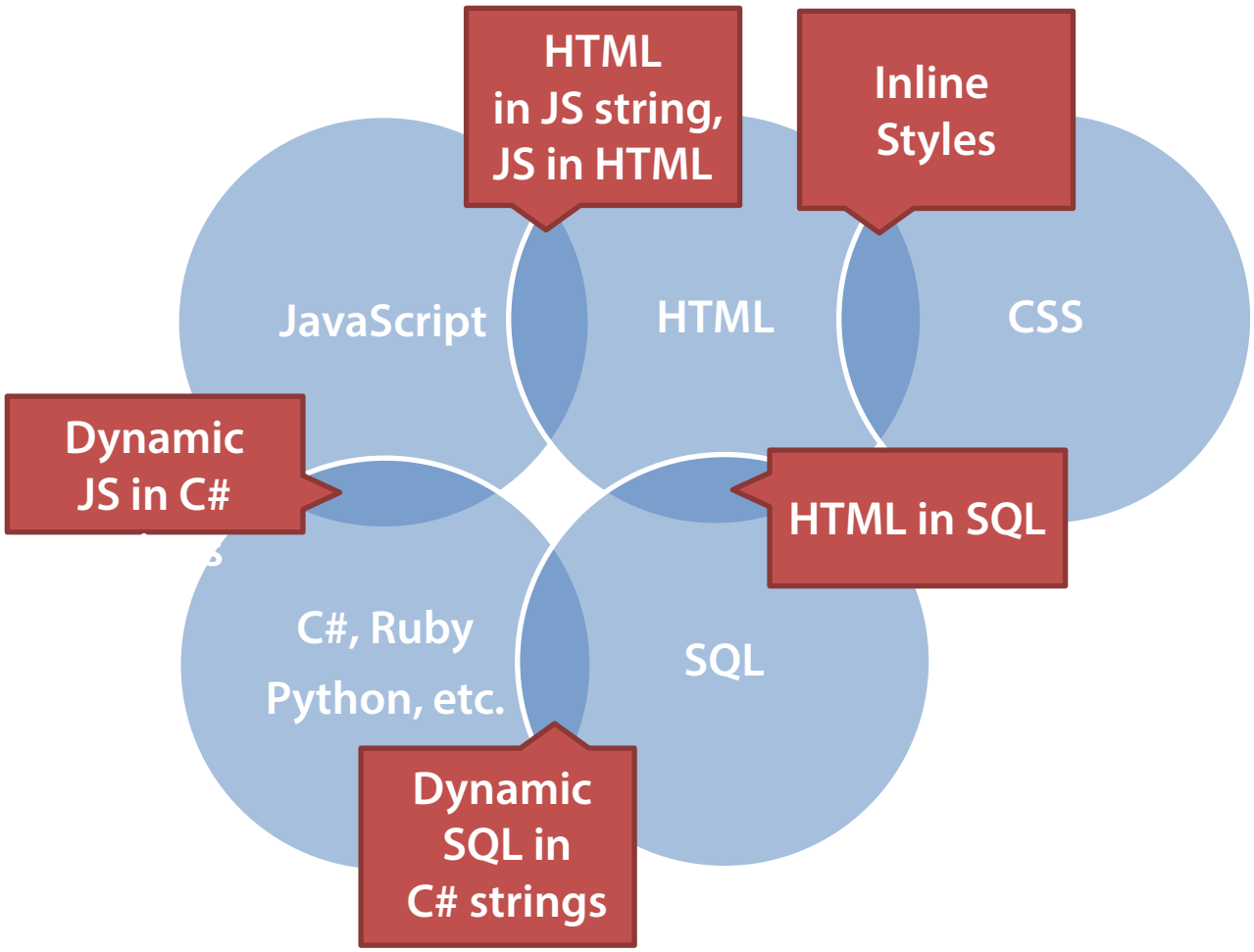


We should use
<lunatic's favorite tool here>
for everything!

1) The Right Tool for the Job

I'm so creative!

Boundaries Matter



Stay native

Dirty

```
string script = @"<script type=""text/javascript"" defer=""defer"">
                    //<![CDATA[
                        var _gaq = _gaq || [];
                        _gaq.push(['_setAccount', '" + ws.GoogleAnalyticsID + @"']);
                        _gaq.push(['_trackPageview']);

                        (function() {
                        var ga = document.createElement('script');
                        ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') +
                        '.google-analytics.com/ga.js';
                        ga.setAttribute('async', 'true');
                        document.documentElement.firstChild.appendChild(ga);
                        })();
                    //]]>
                    </script>";
this.Header.Controls.Add(new LiteralControl("\r\n" + script));
```

Clean

```
<!--In document head-->
<script type="text/javascript">
    var WebSiteSetup = { "GoogleAnalyticsKey": "JDSGI832JDUG9831" };
</script>
```

```
//In GoogleAnalytics.js
var _gaq = _gaq || [];
_gaq.push(['_setAccount', WebSiteSetup.GoogleAnalyticsKey]);
_gaq.push(['_trackPageview']);

(function () {
   var ga = document.createElement('script');
   ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') +
   '.google-analytics.com/ga.js';
   ga.setAttribute('async', 'true');
   document.documentElement.firstChild.appendChild(ga);
})();
```

Stay Native - Advantages

- Cached
- Code colored
- Syntax checked
- Separation of concerns
- Reusable
- Avoids string parsing
- Can minify & obfuscate

Stay Native

Avoid using one language to write another language/format via strings. e.g. Using strings in C#, Java, PHP, etc. to create

- JavaScript
- XML
- HTML
- JSON
- CSS

Leverage Libraries

One language per file

Every Tech is Potential Evil

Linq-to-Sql

- Massive Queries
- Outer joins

Flash/Silverlight

- When native is an option

JavaScript

- Sole Validation
- Proprietary Logic

2) Maximize Signal to Noise Ratio

2) Maximize Signal to Noise Ratio

Signal

Logic that follows the TED rule:

Terse

Expressive

Do one thing

Noise

- **High cyclomatic complexity**
- **Excessive indentation**
- **Zombie code**
- **Unnecessary comments**
- **Poorly named structures**
- **Huge classes**
- **Long methods**
- **Repetition**
- **No whitespace**
- **Overly verbose**

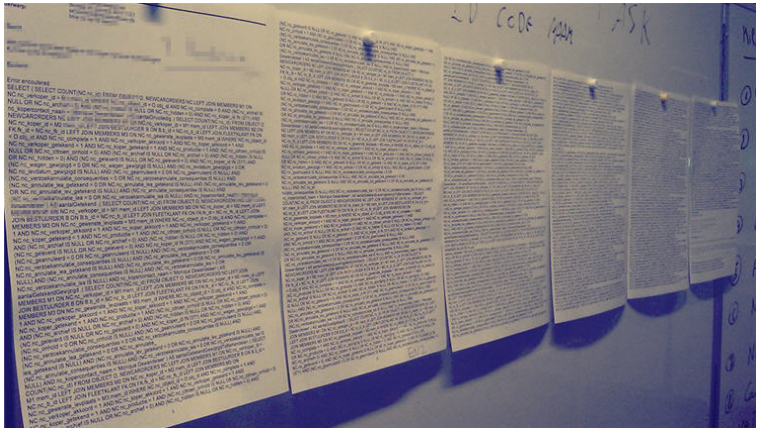
Why is signal to noise ratio so important?

1. Our brain is the compiler

=

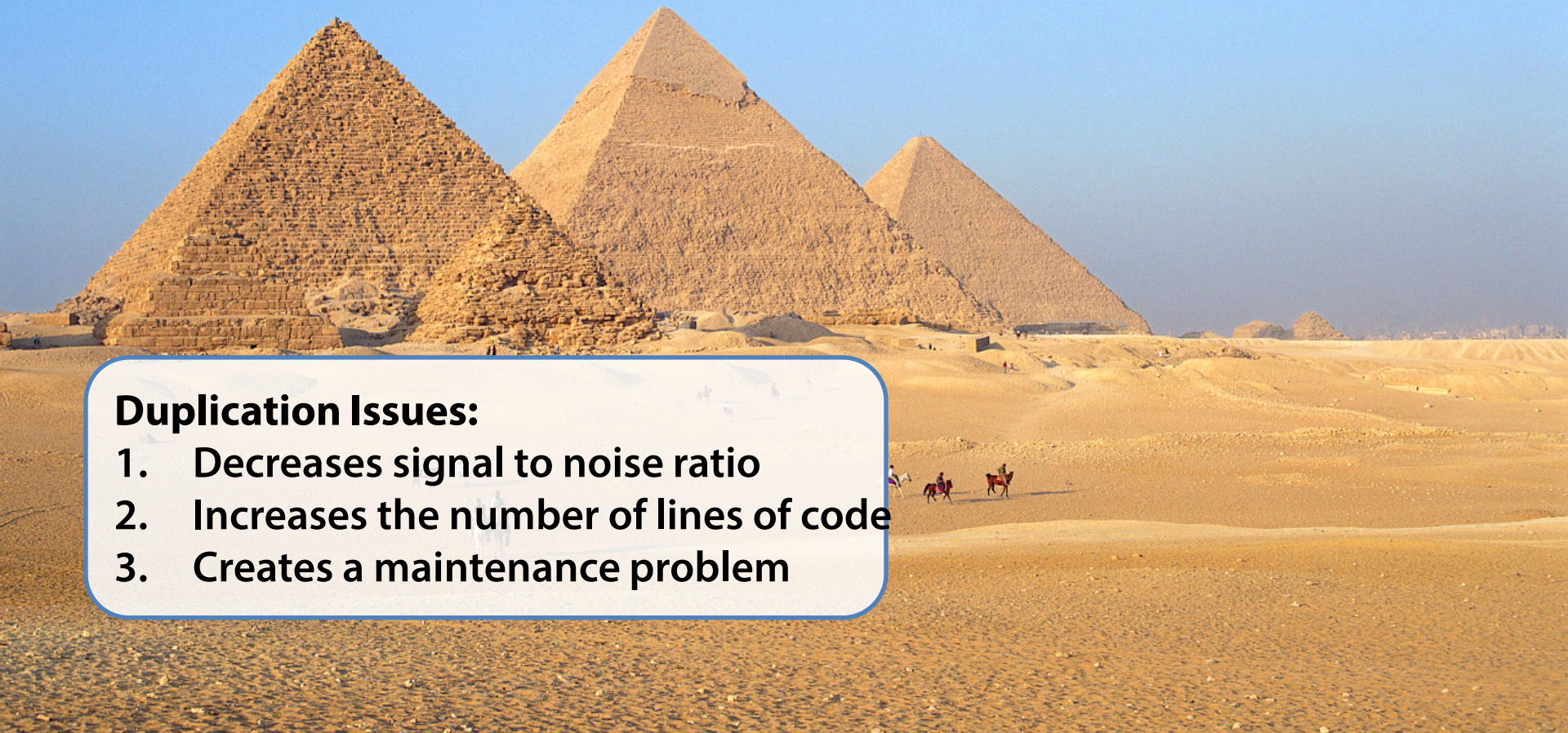
2. The mess builds quietly

=



DRY Principle

- Don't repeat yourself
- Many of same principles as relational DB normalization
- Copy and paste is often a design problem



Duplication Issues:

1. Decreases signal to noise ratio
2. Increases the number of lines of code
3. Creates a maintenance problem

Look for Patterns

```
if (!string.IsNullOrEmpty(ws.SEOTargetLocation1) && ws.SEOTargetLocation1.Contains(","))
{
        string[] pieces = ws.SEOTargetLocation1.Split(",".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
        if (pieces.Length == 2 && pieces[1].Trim().Length == 2)
        {
                string dl1_url = BuildDealerUrl(auto.Make, pieces[0], pieces[1]);
                string dl1_text = string.Format("<a href=\"{0}\">{1} {2} {4}, {5}</a>", dl1_url, auto.YearName ?? 0, auto.Make, auto.Model, pieces[0], pieces[1]);

                _DisclaimerUrls.Text += dl1_text + " ";
        }
}

if (!string.IsNullOrEmpty(ws.SEOTargetLocation2) && ws.SEOTargetLocation2.Contains(","))
{
        string[] pieces = ws.SEOTargetLocation2.Split(",".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
        if (pieces.Length == 2 && pieces[1].Trim().Length == 2)
        {
                string dl1_url = BuildDealerUrl(auto.Make, pieces[0], pieces[1]);
                string dl1_text = string.Format("<a href=\"{0}\">{1} {2} {4}, {5}</a>", dl1_url, auto.YearName ?? 0, auto.Make, auto.Model, pieces[0], pieces[1]);

                _DisclaimerUrls.Text += dl1_text + " ";
        }
}

if (!string.IsNullOrEmpty(ws.SEOTargetLocation3) && ws.SEOTargetLocation3.Contains(","))
{
        string[] pieces = ws.SEOTargetLocation3.Split(",".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
        if (pieces.Length == 2 && pieces[1].Trim().Length == 2)
        {
                string dl1_url = BuildDealerUrl(auto.Make, pieces[0], pieces[1]);
                string dl1_text = string.Format("<a href=\"{0}\">{1} {2} {4}, {5}</a>", dl1_url, auto.YearName ?? 0, auto.Make, auto.Model, pieces[0], pieces[1]);

                _DisclaimerUrls.Text += dl1_text + " ";
        }
}
```

3) Self-documenting Code

Understanding the original programmer's intent is the most difficult problem.
Fjelstad & Hamlen 1979

Well written code is self-documenting.

- **Clear intent**
- **Layers of abstractions**
- **Format for readability**
- **Favor code over comments**

Summary

1. **Clean code begins with picking the right tool**
   - Watch for “boundaries” between technologies
2. **Strive to maximize signal to noise**
   - TED principle: Terse, expressive, do one thing.
   - Don’t repeat yourself
3. **Goal: Self-documenting code**
   - Clear intent
   - Layers of abstractions