Clean Code

Writing Code for Humans

course by Cory House.

Mg. Ing. Efrain R. Bautista Ubillús

ebautistau@unmsm.edu.pe | efrainbautista@gmail.com

Agenda

1. Coding is for humans

2. Principles for Clean Code

3. Clean Code Examples – C#

4. Lab – Java

Coding is for humans

- Programming is the art of telling another human what one wants the computer to do. [Donald Knuth]

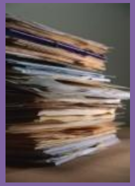
- Any fool can write code that a computer can understand. Good programmers write code that humans can understand. [Martin Fowler]

Three Principles for Clean Code

1) Right tool for the job

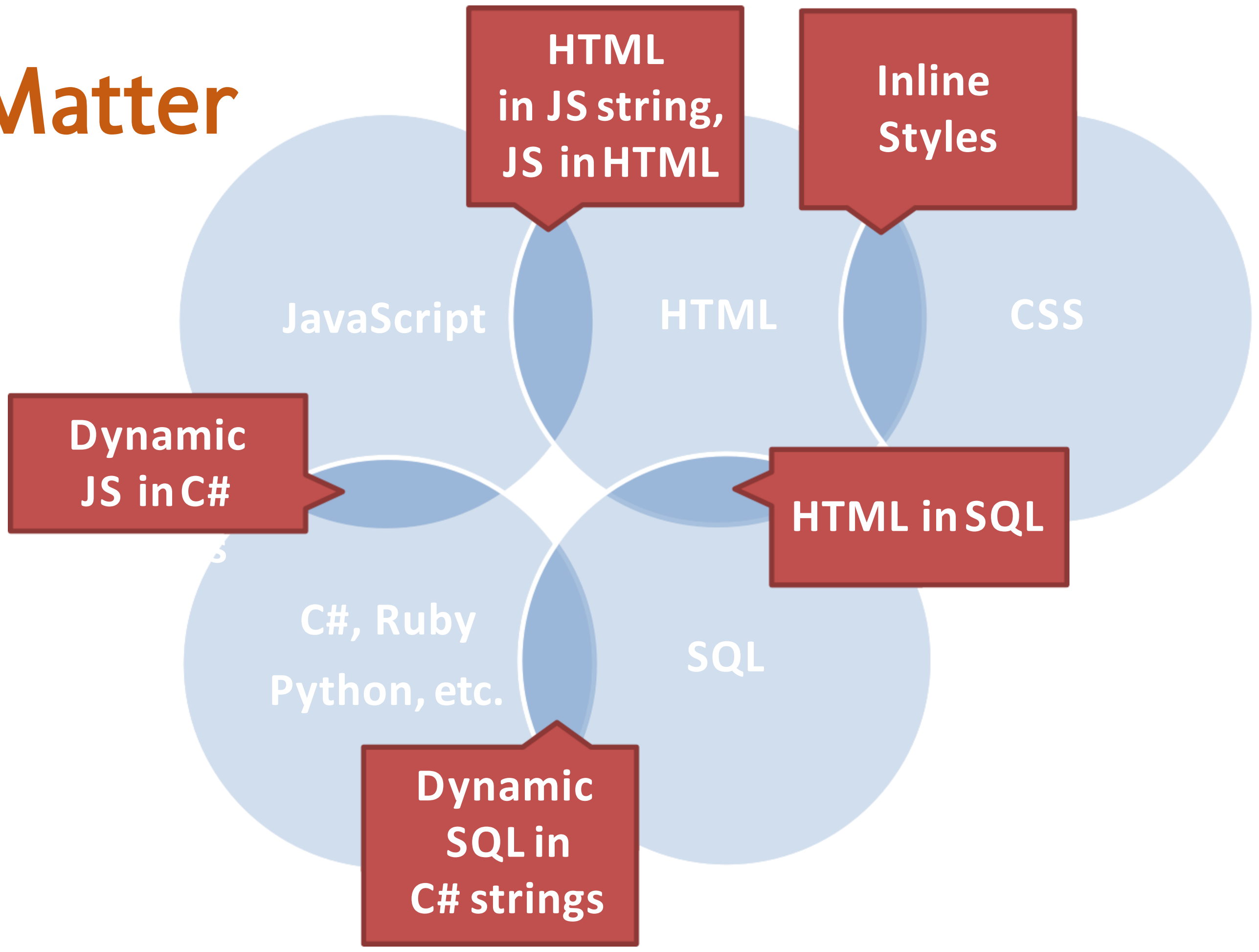
2) High signal to noise ratio

3) Self-documenting



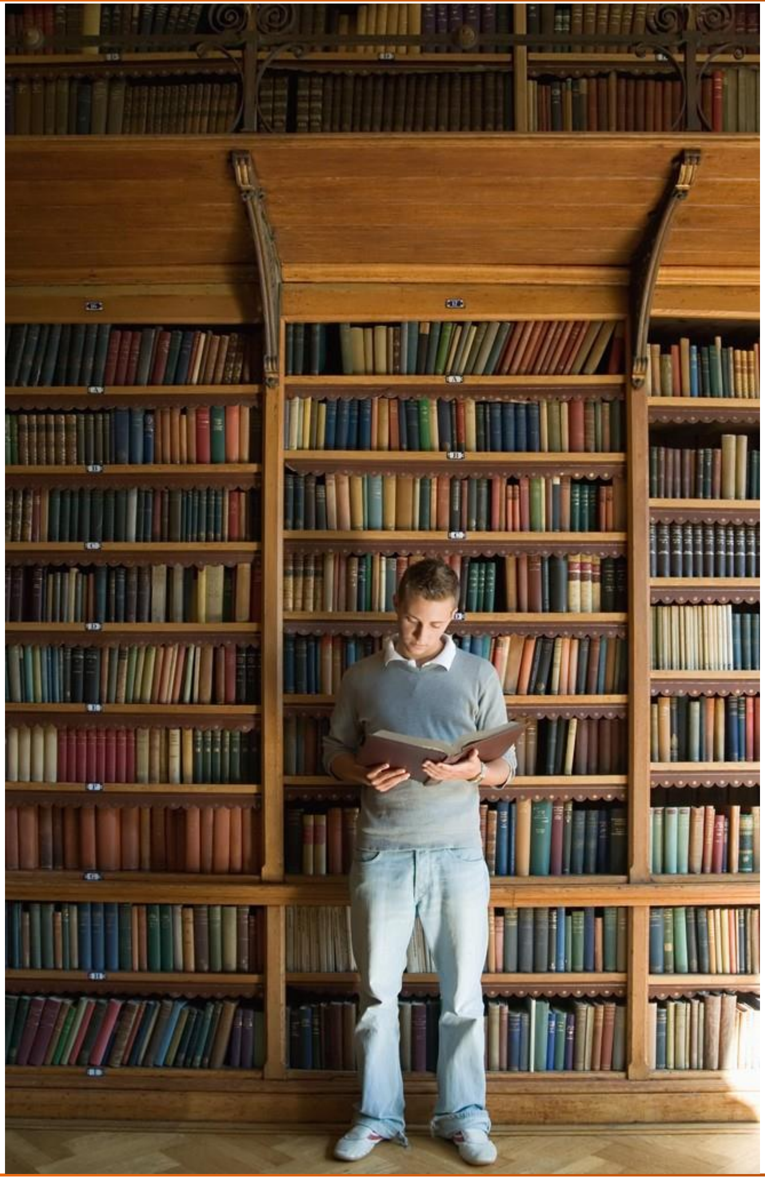
1) The Right Tool for the Job

Boundaries Matter



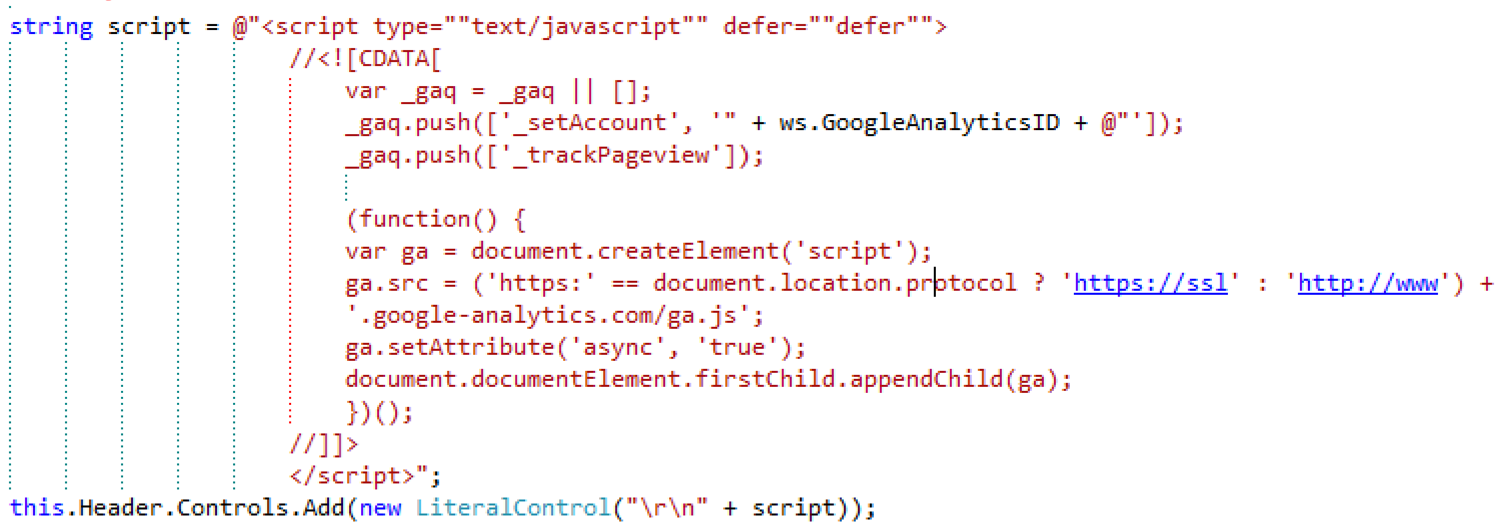
Stay Native

- Avoid using one language to write another language/format via strings.
- Using strings in C#, Java, PHP, etc. to create
  - JavaScript
  - XML
  - HTML
  - JSON
  - CSS
- Leverage Libraries
- One language per file



Stay Native

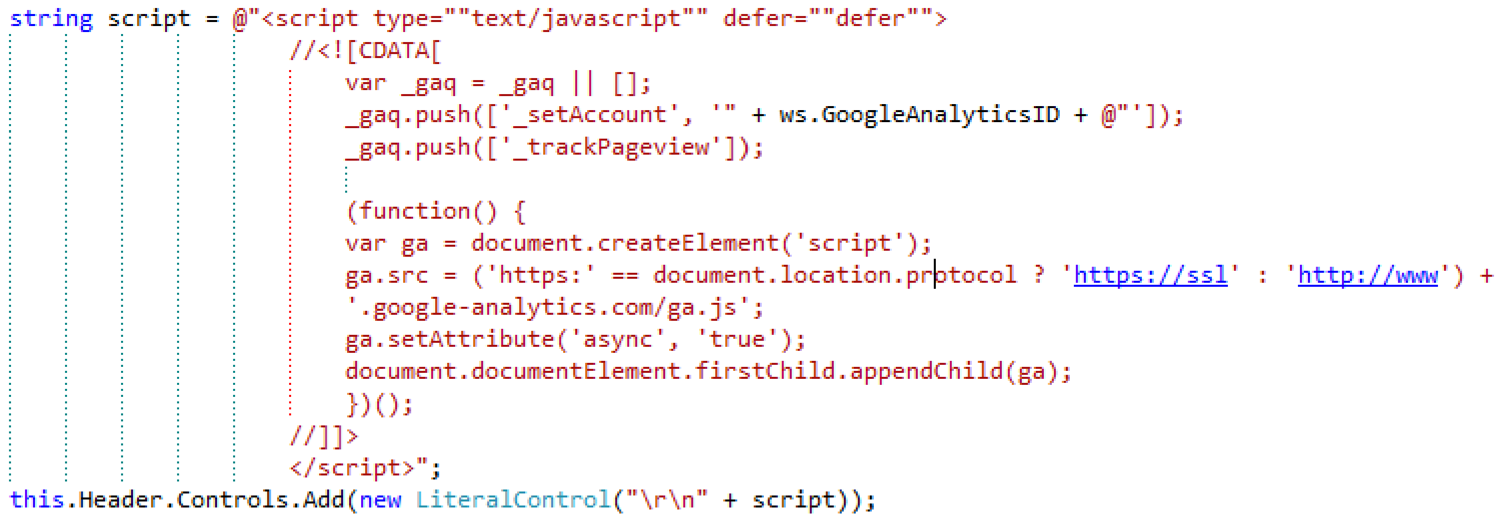
Dirty



```
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));
```

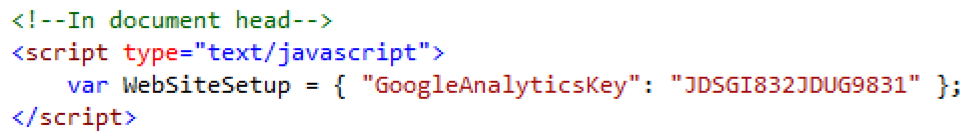
Stay Native

Dirty

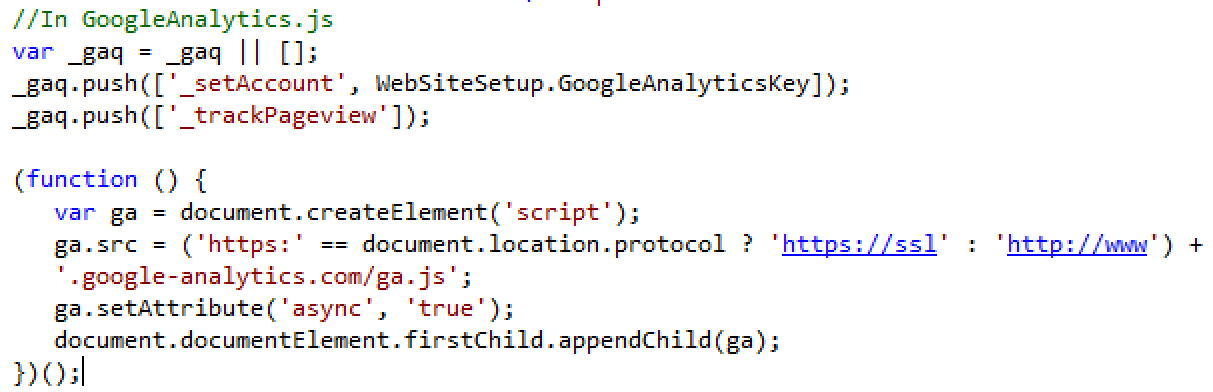


```
string script = @"<script type=""text/javascript"" defer=""defer"">
                    //<![CDATA[
                        var _gaq = _gaq || [];
                        _gaq.push(['_setAccount', '" + ws.GoogleAnalyticsID + @"']);
                        _gaq.push(['_trackPageview']);

                        (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

2) Maximize Signal to Noise Ratio

Signal

Logic that follows the TED rule:

Terse (Breve)

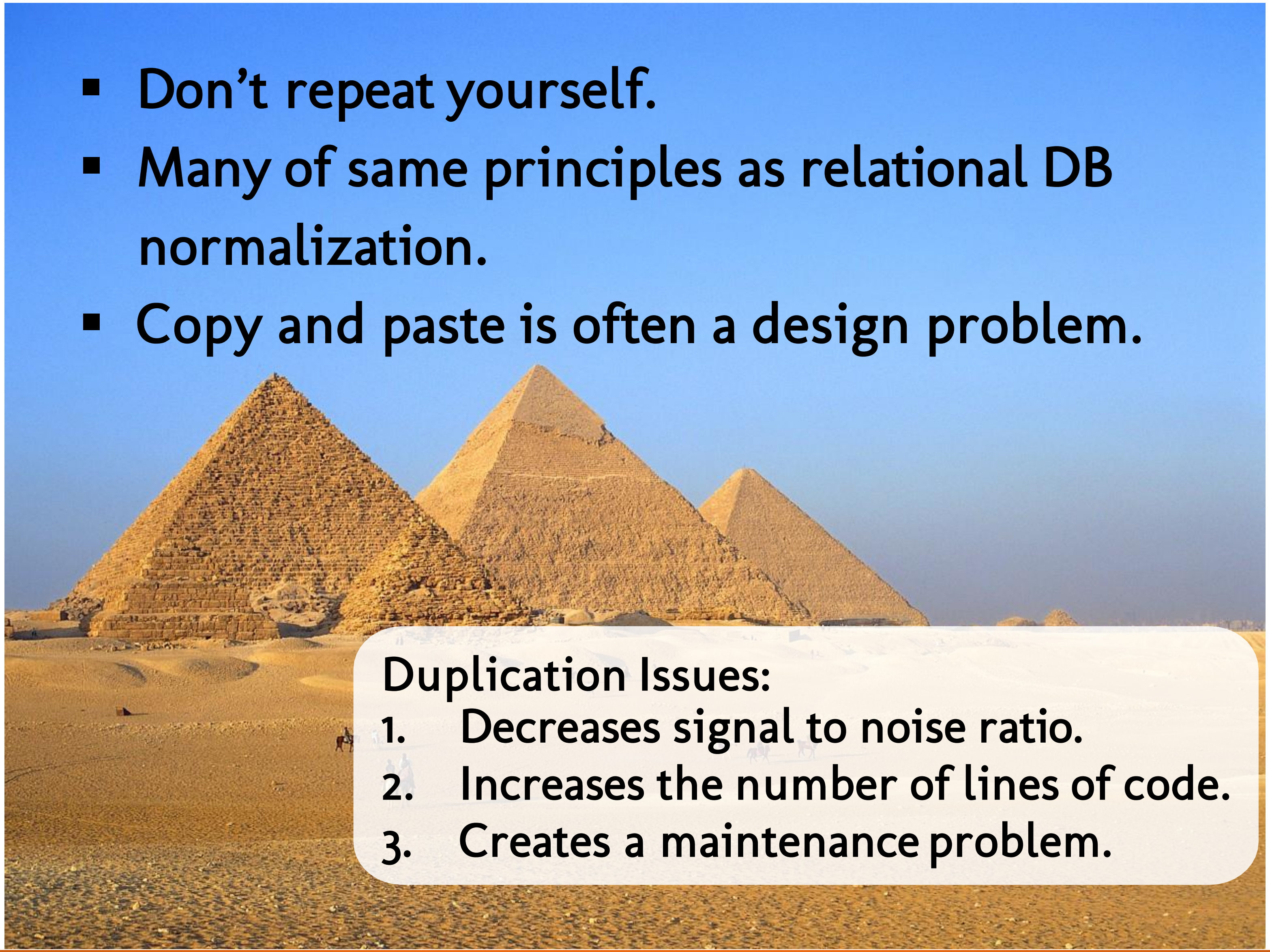
Expressive

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

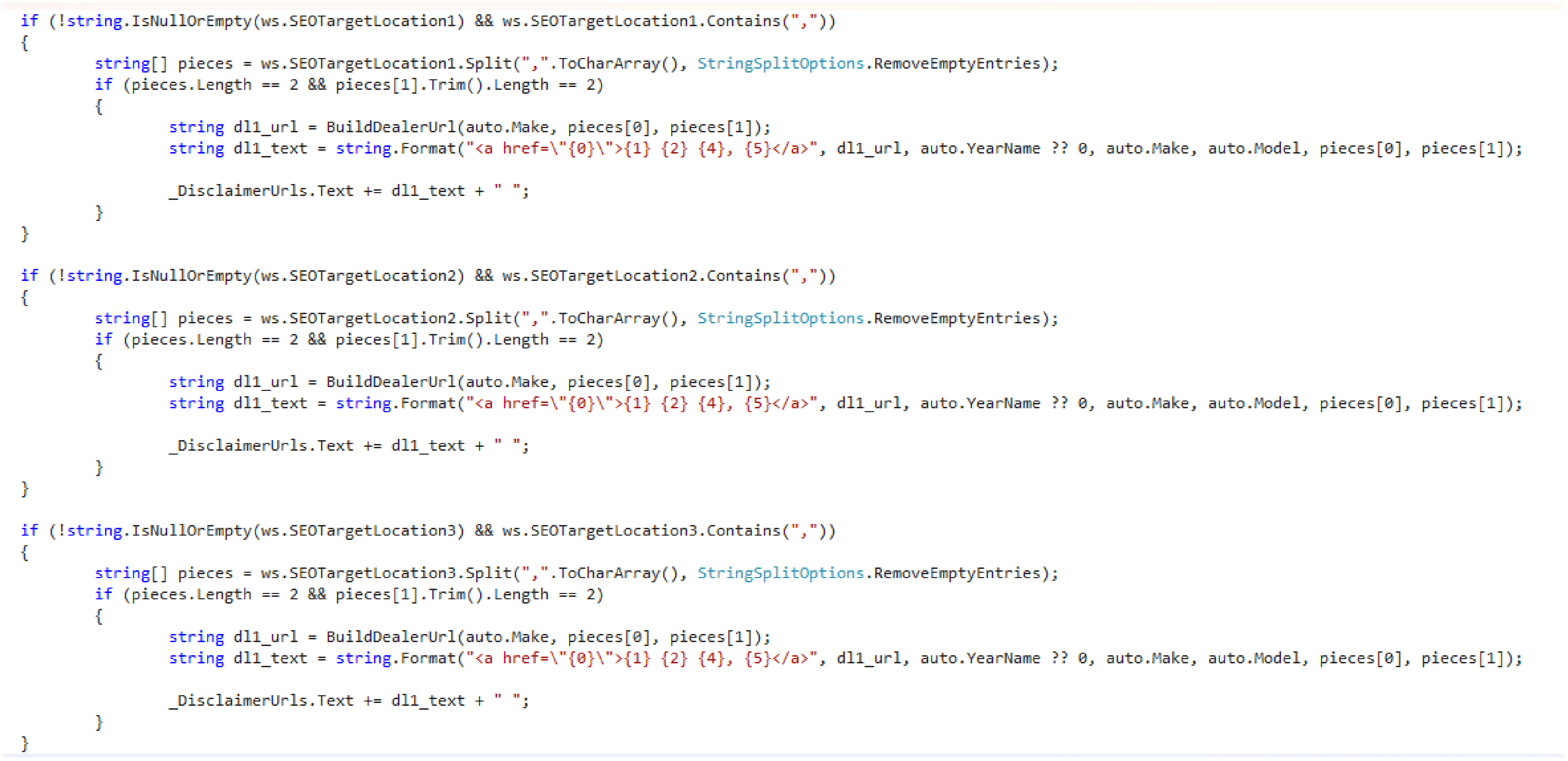
DRY Principle



- 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

Namimg

Dirty

```
List<decimal> p = new List<decimal>() { 5.50m, 10.48m, 12.69m };
decimal t = 0;
foreach (var i in p)
{
    t += i;
}

return t;
```

Could you read this book?

P was very angry with G for insulting her M. G kicked P in the A. He slept on the C.

Namimg

Dirty

```
List<decimal> p = new List<decimal>() { 5.50m, 10.48m, 12.69m };
decimal t = 0;
foreach (var i in p)
{
    t += i;
}

return t;
```

Could you read this book?

P was very angry with G for insulting her M. G kicked P in the A. He slept on the C.

Clean

```
List<decimal> prices = new List<decimal>() { 5.50m, 10.48m, 12.69m };
decimal total = 0;
foreach (var price in prices)
{
    total += price;
}

return total;
```

Namimg Classes

Dirty

- WebsiteBO
- Utility
- Common
- MyFunctions
- JimmysObjects
- *Manager /*Processor/*Info

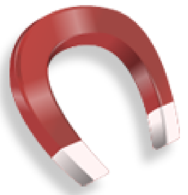
Guidelines:

1. Noun
2. Be specific
3. Single Responsibility
4. Avoid generic suffixes

Namimg Classes

Dirty

- WebsiteBO
- Utility
- Common
- MyFunctions
- JimmysObjects
- *Manager /*Processor/*Info



Guidelines:

1. Noun
2. Be specific
3. Single Responsibility
4. Avoid generic suffixes

Clean

- User
- Account
- QueryBuilder
- ProductRepository

Specific names lead to smaller more cohesive classes

The Method Name Should Say It All

Say what?

- **Get**
- **Process**
- **Pending**
- **Start**

The Method Name Should Say It All

Say what?

- Get
- Process
- Pending
- Start

Right on.

- GetRegisteredUsers
- IsValidSubmission
- ImportDocument
- SendEmail

Watch for Side Effects

- CheckPassword shouldn’t log users out.
- ValidateSubmission shouldn’t save.
- GetUser shouldn’t create their session.
- ChargeCreditCard shouldn’t send emails.

Solution?

Refactor until the method name completely describes what it does.

Avd Abbr

- It’s not the 80’s
- No standard
- We talk about code

RegUsr

RegistUser

RegisUser

RegisterUsr

Naming variables: Booleans

- Boolean names should sound like true/false questions

Dirty

- **open**
- **start**
- **status**
- **login**

```
if (login)
{

}
```

Naming variables: Booleans

- Boolean names should sound like true/false questions

Dirty

- open
- start
- status
- login

```
if (login)
{

}
```

Clean

- isOpen
- done
- isActive
- loggedIn

```
if (loggedIn)
{

}
```

Naming variables: Booleans

- When dealing with states that toggle, consistently use matching pairs

Dirty

- **on/disable**
- **quick/slow**
- **lock/open**
- **slow/max**

Naming variables: Booleans

- When dealing with states that toggle, consistently use matching pairs

Dirty	Clean
on/disable	on/off
quick/slow	fast/slow
lock/open	lock/unlock
slow/max	min/max

Compare Booleans Implicitly

Dirty

```
if (loggedIn == true)
{
    //do something nice.
}
```

Compare Booleans Implicitly

Dirty

```
if (loggedIn == true)
{
    //do something nice.
}
```

Clean

```
if (loggedIn)
{
    //do something nice.
}
```

Assign Booleans Implicitly

Dirty

```
bool goingToChipotleForLunch;

if (cashInWallet > 6.00)
{
    goingToChipotleForLunch = true;
} else {
    goingToChipotleForLunch = false;
}
```

Assign Booleans Implicitly

Dirty

```
bool goingToChipotleForLunch;

if (cashInWallet > 6.00)
{
    goingToChipotleForLunch = true;
} else {
    goingToChipotleForLunch = false;
}
```

Clean

```
bool goingToChipotleForLunch = cashInWallet > 6.00;
```

1. Fewer lines
2. No separate initialization
3. No repetition
4. Reads like speech

Don’t Be Anti-negative

In other words, use positive conditionals!
when it’s possible

Dirty

```
if (!isNotLoggedIn)
```

Don't Be Anti-negative

In other words, use positive conditionals!
when it's possible

Dirty

```
if (!isNotLoggedIn)
```

Clean

```
if (loggedIn)
```

Avoid being “Stringly” Typed

Dirty

```
if (employeeType == "manager")
```

Avoid being “Stringly” Typed

Dirty

```
if (employeeType == "manager")
```

Clean

```
if (employee.Type == EmployeeType.Manager)
```

1. Strongly typed
2. Intellisense support
3. Documents states
4. Searchable

Magic Numbers

Dirty

```
if (age > 21)
{
    //body here
}
```

Dirty

```
if (status == 2)
{
    //body here
}
```

Magic Numbers

Dirty

```
if (age > 21)
{
    //body here
}
```

Clean

```
const int legalDrinkingAge = 21;
if (age > legalDrinkingAge)
{
    //body here
}
```

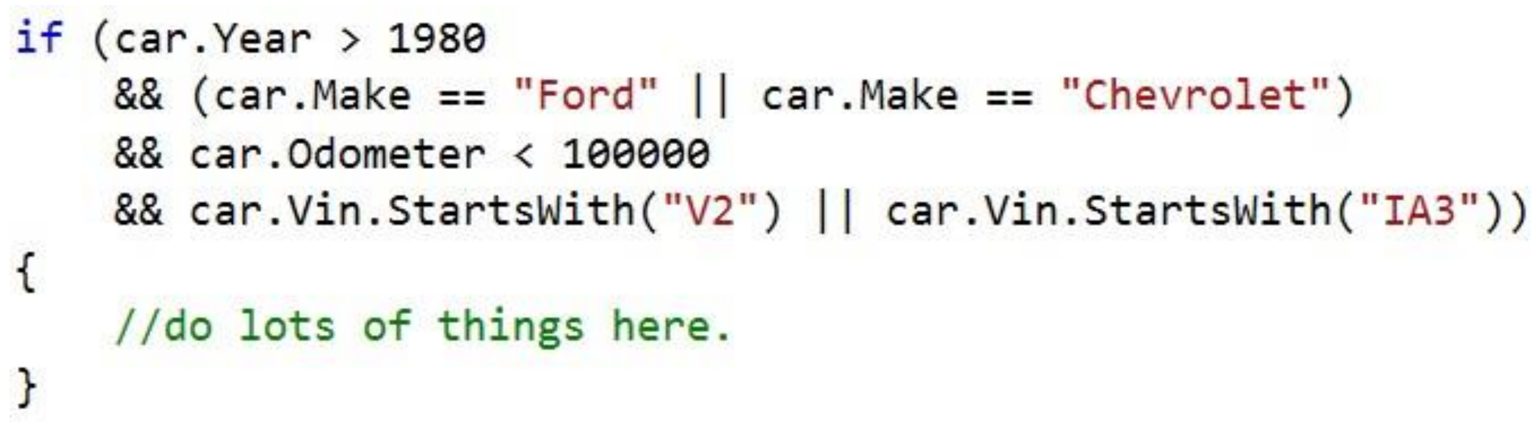
Dirty

```
if (status == 2)
{
    //body here
}
```

Clean

```
if (status == Status.Active)
{
    //body here
}
```

Complex Conditionals



```
if (car.Year > 1980
    && (car.Make == "Ford" || car.Make == "Chevrolet")
    && car.Odometer < 100000
    && car.Vin.StartsWith("V2") || car.Vin.StartsWith("IA3"))
{
    //do lots of things here.
}
```

1. Intermediate variables
2. Encapsulate via function

Intermediate Variables

Dirty

```
if (employee.Age > 55
    && employee.YearsEmployed > 10
    && employee.IsRetired == true)
{
    //logic here
}
```

$\leftarrow$ What question is this trying to answer?

Intermediate Variables

Dirty

```
if (employee.Age > 55
    && employee.YearsEmployed > 10
    && employee.IsRetired == true)
{
    //logic here
}
```

What question is this trying to answer?

Clean

```
bool eligibleForPension = employee.Age > MinRetirementAge
    && employee.YearsEmployed > MinPensionEmploymentYears
    && employee.IsRetired;
```

Encapsulate Complex Conditionals

Dirty

```
//Check for valid file extensions. Confirm admin or active
if (fileExtension == "mp4" ||
    fileExtension == "mpg" ||
    fileExtension == "avi")
    && (isAdmin || isActiveFile);
```

Principle: Favor expressive code over comments

Encapsulate Complex Conditionals

Dirty

```
//Check for valid file extensions. Confirm admin or active
if (fileExtension == "mp4" ||
    fileExtension == "mpg" ||
    fileExtension == "avi")
    && (isAdmin || isActiveFile);
```

Principle: Favor expressive code over comments

Clean

```
if (ValidFileRequest(fileExtension, active, isAdmin))

private bool ValidFileRequest(string fileExtension, bool isActiveFile, bool isAdmin)
{
    var validFileExtensions = new List<string>() { "mp4", "mpg", "avi" };

    bool validFileType = validFileExtensions.Contains(fileExtension);
    bool userIsAllowedToViewFile = isActiveFile || isAdmin;

    return validFileType && userIsAllowedToViewFile;
}
```

Favor Polymorphism over Enums for Behavior

Dirty

```
public void LoginUser(User user)
{
    switch (user.Status)
    {
        case Status.Active:
            //logic for active users
            break;
        case Status.Inactive:
            //logic for inactive users
            break;
        case Status.Locked:
            //logic for locked users
            break;
    }
}
```

Favor Polymorphism over Enums for Behavior

Dirty

```
public void LoginUser(User user)
{
    switch (user.Status)
    {
        case Status.Active:
            //logic for active users
            break;
        case Status.Inactive:
            //logic for inactive users
            break;
        case Status.Locked:
            //logic for locked users
            break;
    }
}
```

Clean

```
public void LoginUser(User user)
{
    user.Login();
}
```

Favor Polymorphism over Enums for Behavior

```
public abstract class User
{
    public string FirstName;
    public string LastName;
    public Status Status;
    public int AccountBalance;

    public abstract void Login();
}
```

```
public class ActiveUser : User
{
    public override void Login()
    {
        //Active user logic here
    }
}

public class InactiveUser : User
{
    public override void Login()
    {
        //Inactive user logic here
    }
}

public class LockedUser : User
{
    public override void Login()
    {
        //Locked user logic here
    }
}
```

Be declarative if possible

Dirty

```
List<User> matchingUsers = new List<User>();

foreach (var user in users)
{
    if (user.AccountBalance < minimumAccountBalance
        && user.Status == Status.Active)
    {
        matchingUsers.Add(user);
    }
}

return matchingUsers;
```

Be declarative if possible

Dirty

```
List<User> matchingUsers = new List<User>();

foreach (var user in users)
{
    if (user.AccountBalance < minimumAccountBalance
        && user.Status == Status.Active)
    {
        matchingUsers.Add(user);
    }
}

return matchingUsers;
```

Clean

```
return users
    .Where(u => u.AccountBalance < minimumAccountBalance)
    .Where(u => u.Status == Status.Active);
```

C#: LINQ to objects
Java: Lambda

Table Driven Methods

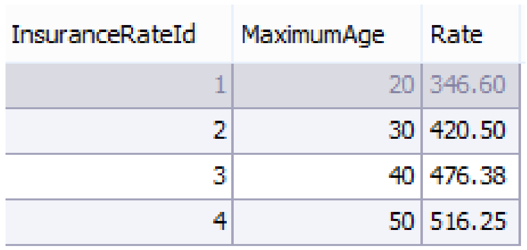
Dirty

```
if (age < 20)
{
    return 345.60m;
}
else if (age < 30)
{
    return 419.50m;
}
else if (age < 40)
{
    return 476.38m;
}
else if (age < 50)
{
    return 516.25m;
}
```

Clean

```
return Repository.GetInsuranceRate(age);
```

InsuranceRate table



InsuranceRateId	MaximumAge	Rate
1	20	346.60
2	30	420.50
3	40	476.38
4	50	516.25

Examples
- **Insurance rates**
- **Pricing structures**
- **Complex and dynamic business**

- Great for dynamic logic
- Avoids hard coding
- Write less code – Avoids complex data structures
- Easily changeable without a code change/app deployment

When to create a method / function

Duplication

Indentation

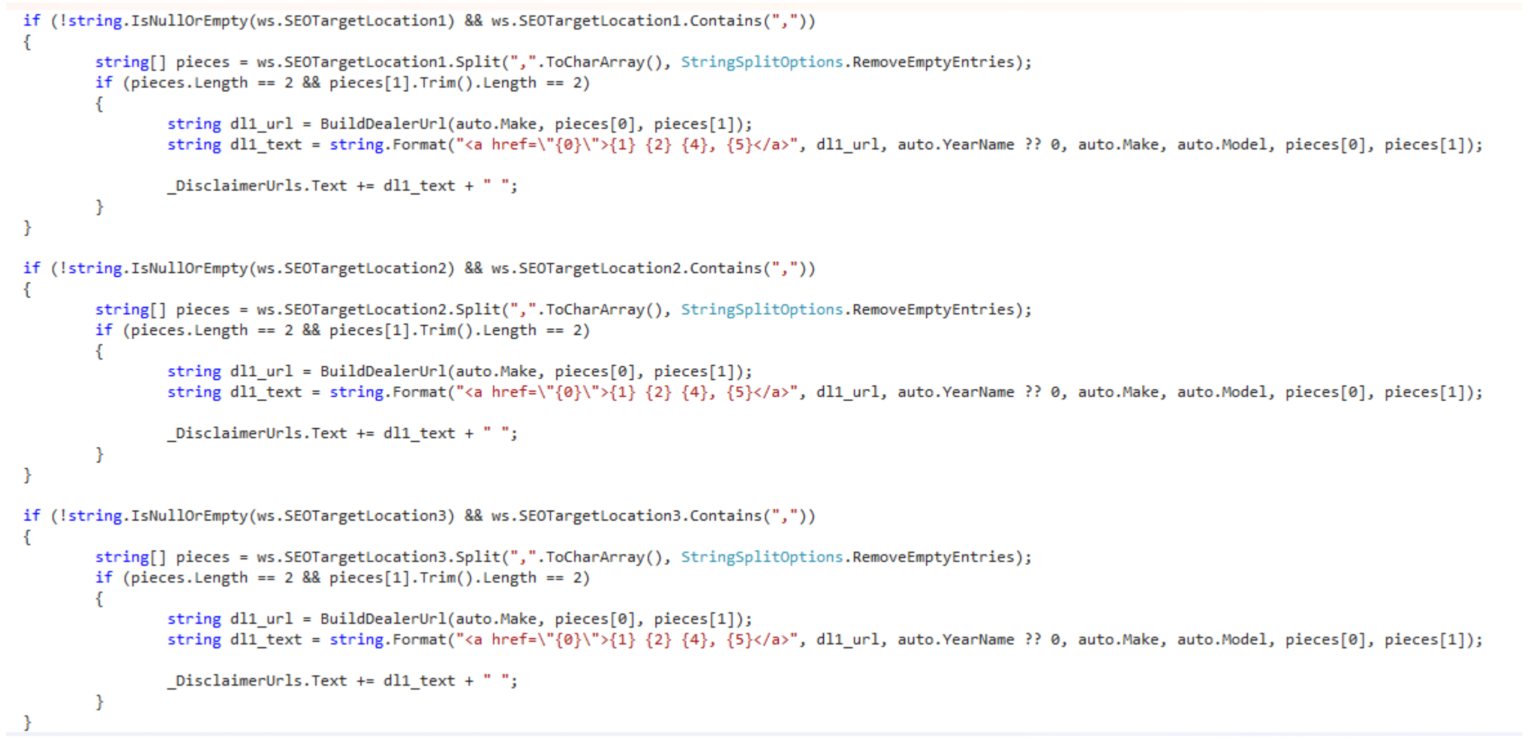
Unclear intent

> 1task

1) Duplication

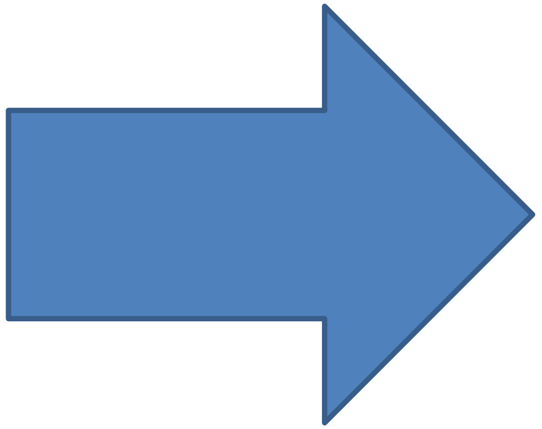
Key: Don’t repeat yourself.
Less is more.

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 + " ";
        }
}
```

2) Excessive Indentation: Arrow Code



```
if
    if
        if
            if
                do stuff
            endif
        endif
    endif
endif
```

Comprehension decreases beyond three levels of nested 'if' blocks.

2) Excessive Indentation: Solutions

Extract Method

Fail Fast

Return Early

2) Excessive Indentation: Extract Method

Before

```
if
    if
        while
            do
            some
            complicated
            thing
        end while
    end if
end if
```

2) Excessive Indentation: Extract Method

Before

```
if
    if
        while
            do
            some
            complicated
            thing
        end while
    end if
end if
```

After

```
if
    if
        doComplicatedThing()
    end if
end if
```

```
doComplicatedThing()
{
    while
        do some complicated thing
    end while
}
```

2) Excessive Indentation: Return Early

Use a return when it enhances readability...
In certain routines, once you know the answer…
Not returning immediately means that you have to write more code.

Steve McConnell, “Code Complete”

2) Excessive Indentation: Return Early

Dirty

```
private bool ValidUsername(string username)
{
    bool isValid = false;

    const int MinUsernameLength = 6;
    if (username.Length >= MinUsernameLength)
    {
        const int MaxUsernameLength = 25;
        if (username.Length <= MaxUsernameLength)
        {
            bool isAlphaNumeric = username.All(Char.IsLetterOrDigit);
            if (isAlphaNumeric)
            {
                if (!ContainsCurseWords(username))
                {
                    isValid = IsUniqueUsername(username);
                }
            }
        }
    }
    return isValid;
}
```

2) Excessive Indentation: Return Early

Clean

```
private bool ValidUsername(string username)
{
    const int MinUsernameLength = 6;
    if (username.Length < MinUsernameLength) return false;

    const int MaxUsernameLength = 25;
    if (username.Length > MaxUsernameLength) return false;

    bool isAlphaNumeric = username.All(Char.IsLetterOrDigit);
    if (!isAlphaNumeric) return false;

    if (ContainsCurseWords(username)) return false;

    return IsUniqueUsername(username);
}
```

2) Excessive Indentation: Fail Fast

Dirty

```
public void RegisterUser(string username, string password)
{
    if (!string.IsNullOrWhiteSpace(username))
    {
        if (!string.IsNullOrWhiteSpace(password))
        {
            //register user here.
        }
        else
        {
            throw new ArgumentException("Username is required.");
        }
    }
    else
    {
        throw new ArgumentException("Password is required");
    }
}
```

2) Excessive Indentation: Fail Fast

Dirty

```
public void RegisterUser(string username, string password)
{
    if (!string.IsNullOrWhiteSpace(username))
    {
        if (!string.IsNullOrWhiteSpace(password))
        {
            //register user here.
        }
        else
        {
            throw new ArgumentException("Username is required.");
        }
    }
    else
    {
        throw new ArgumentException("Password is required");
    }
}
```

Clean

```
public void RegisterUser(string username, string password)
{
    if (string.IsNullOrWhiteSpace(username)) throw new ArgumentException("Username is required.");
    if (string.IsNullOrWhiteSpace(password)) throw new ArgumentException("Password is required");

    //register user here.
}
```

3) Unclear Intent

Dirty

```
//Check for valid file extensions. Confirm admin or active
if (fileExtension == "mp4" ||
    fileExtension == "mpg" ||
    fileExtension == "avi")
    && (isAdmin || isActiveFile);
```

Clean

```
if (ValidFileRequest(fileExtension, active, isAdmin))

private bool ValidFileRequest(string fileExtension, bool isActiveFile, bool isAdmin)
{
    var validFileExtensions = new List<string>() { "mp4", "mpg", "avi" };

    bool validFileType = validFileExtensions.Contains(fileExtension);
    bool userIsAllowedToViewFile = isActiveFile || isAdmin;

    return validFileType && userIsAllowedToViewFile;
}
```

4) Do one thing

Aids the reader

Promotes reuse

Eases naming and testing

Avoids side-effects

Could you read a book with no paragraphs?

How many parameters?

- Strive for 0 – 3 parameters
- Easier to understand
- Easier to test
- Helps assure function does one thing

Dirty

```
public void SaveUser(User user, bool sendEmail, int emailFormat,
    bool printReport, bool sendBill)
```

How many parameters?

- Strive for 0 - 3 parameters
- Easier to understand
- Easier to test
- Helps assure function does one thing

Dirty

```
public void SaveUser(User user, bool sendEmail, int emailFormat,
    bool printReport, bool sendBill)
```

Clean

```
private void SaveUser(User user)
```

Watch for Flag Arguments

- A sign the function is doing two things.

Dirty

```
private void SaveUser(User user, bool emailUser)
{
    //save user

    if (emailUser)
    {
        //email user
    }
}
```

Watch for Flag Arguments

- A sign the function is doing two things.

Dirty

```
private void SaveUser(User user, bool emailUser)
{
    //save user

    if (emailUser)
    {
        //email user
    }
}
```

Clean

```
private void SaveUser(User user)
{
    //save user
}

private void EmailUser(User user)
{
    //email user
}
```

Signs it's too long?

- Whitespace & Comments
- Scrolling required
- Naming issues
- Multiple Conditionals
- Hard to digest

Rarely be over 20 lines
Hardly ever over 100 lines
No more than 3 parameters
Robert C. Martin , "Clean Code"

Signs it’s too long?

The maximum length…is inversely proportional to the complexity and indentation level of that function. So, if you have a conceptually simple function that is just one long (but simple) case statement…it's OK to have a longer function…if you have a complex function…adhere to limits all the more closely.

Linux style guide

Simple functions can be longer. Complex functions should be short.

Try/Catch/Log = Fail Slow

Dirty

```
try
{
    RegisterSpeaker();
}
catch(Exception e)
{
    LogError(e);
}

EmailSpeaker();
```

Try/Catch/Log = Fail Slow

Dirty

```
try
{
    RegisterSpeaker();
}
catch(Exception e)
{
    LogError(e);
}

EmailSpeaker();
```

Clean

```
RegisterSpeaker();
EmailSpeaker();
```

Try/Catch Body Standalone

Dirty

```
try
{
    //many
    //lines
    //of
    //complicated
    //and
    //verbose
    //logic
    //here
}
catch (ArgumentOutOfRangeException)
{
    //do something here
}
```

Try/Catch Body Standalone

Dirty

```
try
{
    //many
    //lines
    //of
    //complicated
    //and
    //verbose
    //logic
    //here
}
catch (ArgumentOutOfRangeException)
{
    //do something here
}
```

Clean

```
try
{
    SaveThePlanet();
}
catch (ArgumentOutOfRangeException)
{
    //do something here
}

private void SaveThePlanet()
{
        //many
        //lines
        //of
        //complicated
        //and
        //verbose
        //logic
        //here
}
```

High Cohesion

Low

- **Vehicle**
  - Edit vehicle options
  - Update pricing
  - Schedule maintenance
  - Send maintenance reminder
  - Select financing
  - Calculate monthly payment

High Cohesion

Low

- **Vehicle**
  - Edit vehicle options
  - Update pricing
  - Schedule maintenance
  - Send maintenance reminder
  - Select financing
  - Calculate monthly payment

High

- **Vehicle**
  - Edit vehicle options
  - Update pricing
- **VehicleMaintenance**
  - Schedule maintenance
  - Send maintenance reminder
- **VehicleFinance**
  - Select financing
  - Calculate monthly payment

Primitive Obsession

Dirty

```
private void SaveUser(string firstName, string lastName, string state, string zip,
    string eyeColor, string phone, string fax, string maidenName)
```

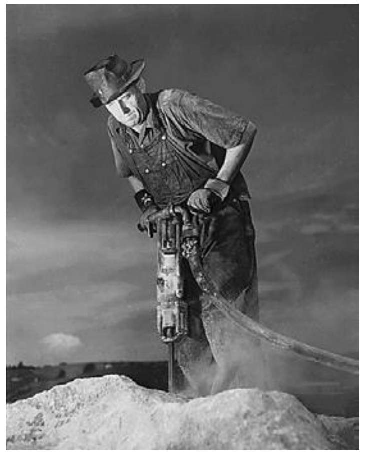
Primitive Obsession

Dirty

```
private void SaveUser(string firstName, string lastName, string state, string zip,
    string eyeColor, string phone, string fax, string maidenName)
```

Clean

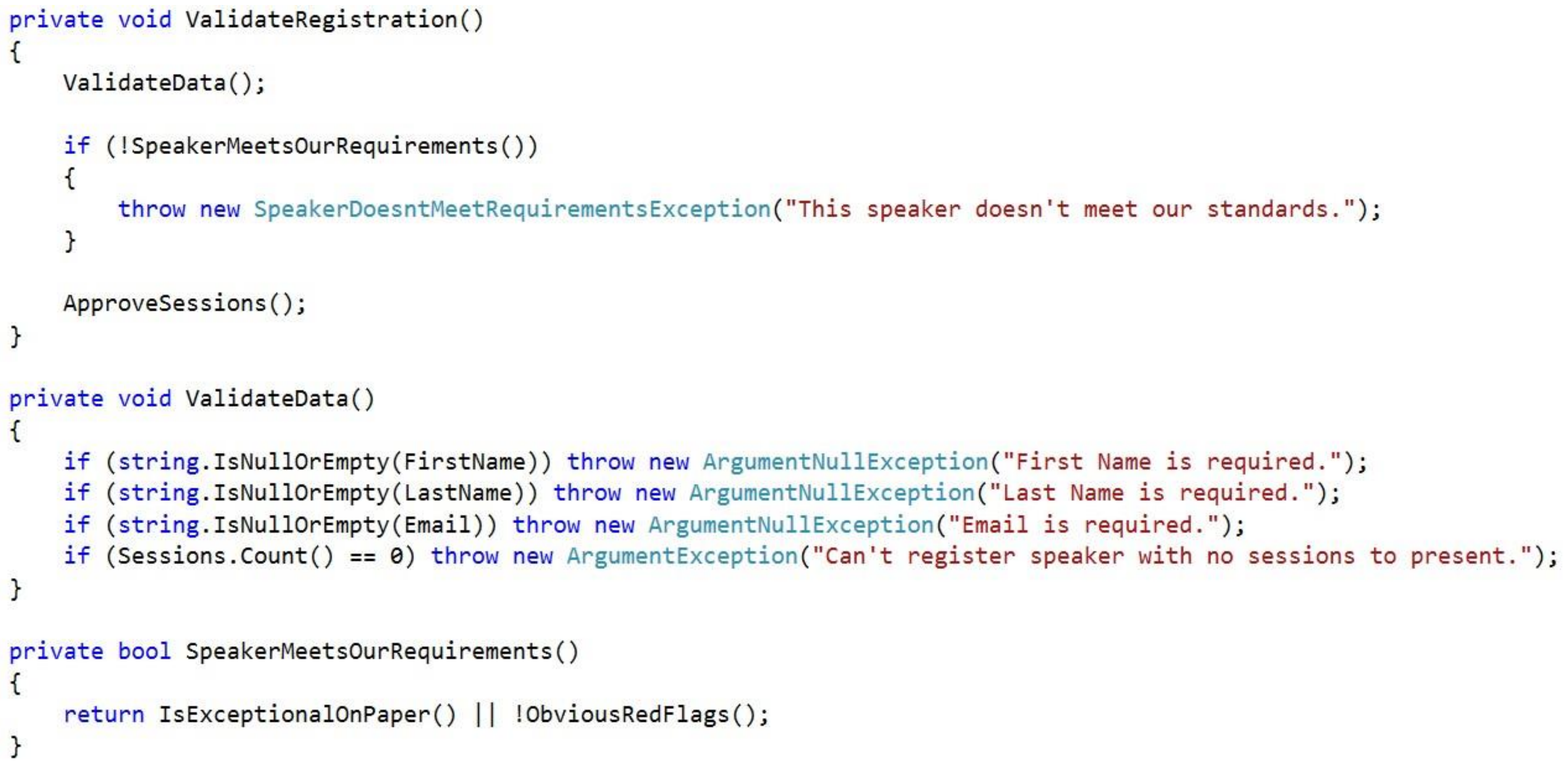
```
private void SaveUser(User user)
```



1. Helps reader conceptualize
2. Implicit -> Explicit
3. Encapsulation
4. Aids maintenance
5. Easy to find references

Principle of Proximity

- Strive to make code read top to bottom when possible
- Keep related actions together



```
private void ValidateRegistration()
{
    ValidateData();

    if (!SpeakerMeetsOurRequirements())
    {
        throw new SpeakerDoesntMeetRequirementsException("This speaker doesn't meet our standards.");
    }

    ApproveSessions();
}

private void ValidateData()
{
    if (string.IsNullOrEmpty(FirstName)) throw new ArgumentNullException("First Name is required.");
    if (string.IsNullOrEmpty(LastName)) throw new ArgumentNullException("Last Name is required.");
    if (string.IsNullOrEmpty(Email)) throw new ArgumentNullException("Email is required.");
    if (Sessions.Count() == 0) throw new ArgumentException("Can't register speaker with no sessions to present.");
}

private bool SpeakerMeetsOurRequirements()
{
    return IsExceptionalOnPaper() || !ObviousRedFlags();
}
```

Comments – Typical interview

How do you write code that's readable and maintainable by others?
Uh…I use comments?

Comments

General Rules:

1. Prefer expressive code over comments.
2. Use comments when code alone can't be sufficient.

Redundant Comments

```
int i = 1; // Set i = 1


var cory = new User(); //Instantiate a new user


/// <summary>
/// Default Constructor
/// </summary>
public User()
{
}


/// <summary>
/// Calcuates Total Charges
/// </summary>
private void CalculateTotalCharges()
{
    //Total charges calculated here
}
```

- Assume your reader can read.
- Don’t repeat yourself.

Intent Comments

Dirty

```
// Assure user’s account is deactivated.
if (user.Status == 2)
```

Intent Comments

Dirty

```
// Assure user's account is deactivated.
if (user.Status == 2)
```

Clean

```
if (user.Status == Status.Inactive)
{

}
```

Instead, clarify intent in code:

- Improved function naming
- Intermediate variable
- Constant or enum
- Extract conditional to function

Apology Comments

Dirty

```
// Sorry, this crashes a lot so I'm just swallowing the exception.

// I was too tired to refactor this pile
// of spaghetti code when I was done...
```

- Don't apologize.
  - Fix it before commit/merge.
  - Add a TODO marker comment if you must

Warning Comments

Dirty

```
// Here be dragons - See Bob

// Great sins against code
// begin here...
```

- To avoid warning, refactor.

Kill Zombie Code

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        Page.ClientScript.RegisterStartupScript(this.GetType(), "maps", "initialize();", true);
        address1.Value = Request.QueryString["z"];
        txtEstDistance.Visible = true;
    }
    if (!Page.IsPostBack)
    {
        imgbtnBinManagerGreen.Visible = false;
        imgbtnBinCheckGreen.Visible = false;
        imgbtnBinManagerBasicGreen.Visible = false;
        SetNewCustomerID();
    }


    //HttpWebRequest request = WebRequest.Create("http://api.hostip.info/get_json.php") as HttpWebRequest;
    //WebResponse response = request.GetResponse();
    //DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(ZipCode));
    //ZipCode zip = serializer.ReadObject(response.GetResponseStream()) as ZipCode;

   // address1.Value = "64064";
    //address1.Value = zip.country_name;

    //Label1.Text = ipaddress;
}

/// <summary>
/// If an existing customer is selected on the previous step, then NewCustomerID = 0.
/// It needs to have a value since it's referenced when creating the quote. So set the NewCustomerID
/// to the UserID sent in the querystring
/// </summary>
private void SetNewCustomerID()
{
    SessionHelper.NewCustomerID = Convert.ToInt32(Request.QueryString["uid"]);
}

//protected void LinkButton1_Click(object sender, EventArgs e)
//{


//        Page.ClientScript.RegisterStartupScript(this.GetType(), "maps", "initialize();", true);
//        txtBoxEnterZip.Visible = false;
//        txtEstDistance.Visible = true;
//        lnkbtnGetZip.Visible = false;
//        address1.Value = txtBoxEnterZip.Text;

//}
```

Kill Zombie Code

- Reduces readability
- Creates ambiguity
- Hinders refactoring
- Add noise to searches
- Code isn't "lost" anyway

Kill Zombie Code – A mental checklist

About to comment out code? Ask yourself:

- When, if ever, would this be uncommented?
- Can I just get it from source control later?
- Is this incomplete work that should be worked via a branch?
- Is this a feature that should be enabled/disabled via configuration?
- Did I refactor out the need for this code?

Divider Comments

Dirty

```
private void MyLongFunction()
{
    lots
    of
    code

    //Start search for available concert tickets

    lots
    of
    concert
    search
    code

    //End of concert ticket search

    lots
    more
    code
}
```

Need comments to divide function sections?
Refactor.

Brace Tracker Comments

Dirty

```
private void AuthenticateUsers()
{
    bool validLogin = false;
    //deeply
        //nested
            //code

            if (validLogin)
            {
                //Lots
                //of
                //code
                //to
                //log
                //user
                //in

            } //end user login

        //even
    //more code
}
```

Brace Tracker Comments

Dirty

```
private void AuthenticateUsers()
{
    bool validLogin = false;
    //deeply
        //nested
            //code

            if (validLogin)
            {
                //Lots
                //of
                //code
                //to
                //log
                //user
                //in

            } //end user login

        //even
    //more code
}
```

Clean

```
private void AuthenticateUsers()
{
    bool validLogin = false;
    //deeply
        //nested
            //code

            if (validLogin)
            {
                LoginUser();
            }

        //even
    //more code
}
```

Bloated Header

Dirty

```
//*****************************************************
// Filename: Monolith.cs                              *
//                                                    *
// Author: Cory House                                 *
// Created: 12/20/2012                                *
// Weather that day: Patchy fog, then snow            *
//                                                    *
// Summary                                            *
// This class does a great many things. To make it   *
// extra useful I placed pretty much all the app      *
// logic here. You wish your class was this           *
// powerful. Bwahhahha!                               *
//*****************************************************
```

- Avoid line endings
- Don’t repeat yourself
- Follow Conventions

Defect Log

Dirty

```
// DEFECT #5274 DA 12/10/2010
// We weren't checking for null here.
if (FirstName != null)
{
    //code continues...
```

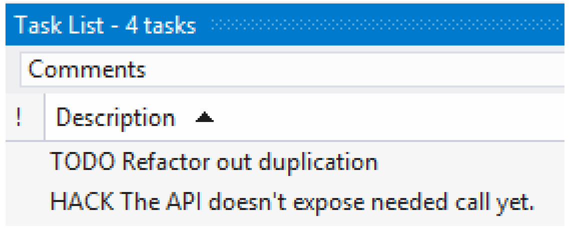
- Change metadata belongs in source control
- A well written book doesn't need covered in author notes

Clean Comments

- To Do
- Summary
- Documentation

To Do Comments

```
// TODO Refactor out duplication

// HACK The API doesn't expose needed call yet.
```



Task List - 4 tasks

Comments

!	Description ▲
	TODO Refactor out duplication
	HACK The API doesn't expose needed call yet.

- **Standardize**
- **Watch out:**
  - May be an apology or warning comment in disguise
  - Often ignored

Summary Comments

Clean

```
//Encapsulates logic for calculating retiree benefits


//Generates custom newsletter emails
```

- Describes intent at general level higher than the code
- Often useful to provide high level overview of classes
- Risk: Don’t use to simply augment poor naming/code level intent

Documentation

Clean

```
// See www.facebook.com/api for documentation
```

- Only when it can’t be expressed in code.

About to write a comment?

For clean coders, comments are useful, but generally a last resort. Ask yourself:

1. **Could I express what I'm about to type in *code*?**

   Intermediate variable, eliminate magic number, utilize enum?

   Refactor to a well-named method.

   - ◻ Separate scope
   - ◻ More likely to stay updated
   - ◻ Better testability
2. **Am I explaining bad code I've just written instead of refactoring?**
3. **Should this simply be a message in a source control commit?**

The Outline Rule

Typical Class

- **Class**
  - Method 1
    - Method 1a
      - Method 1ai
      - Method 1aii
      - Method 1aiii
    - Method 1b
    - Method 1c

The Outline Rule

Typical Class

- **Class**
  - Method 1
    - Method 1a
      - Method 1ai
      - Method 1aii
      - Method 1aiii
    - Method 1b
    - Method 1c

Strive for this

- **Class**
  - Method 1
    - Method 1a
      - Method 1ai
      - Method 1bii
    - Method 1b
    - Method 1c
  - Method 2
    - Method 2a
    - Method 2b
  - Method 3
    - Method 3a
    - Method 3b

The Outline Rule

- **Speaker**
  - Register

The Outline Rule

- Speaker
  - Register

- Speaker
  - Register
    - Validate Registration
      - Validate Data
      - Check if speaker appears qualified
        - Appears Exceptional
        - Obvious Red Flags
      - Approve Sessions
        - Session is about old tech
    - Save Speaker

The Boy Scoute Rule

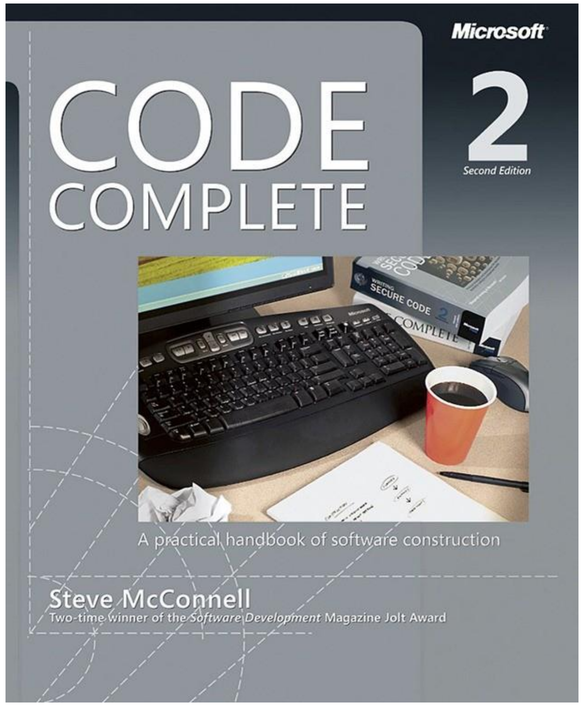
Always leave the code you’re editing a little better than you found it.

[Robert C Martin]

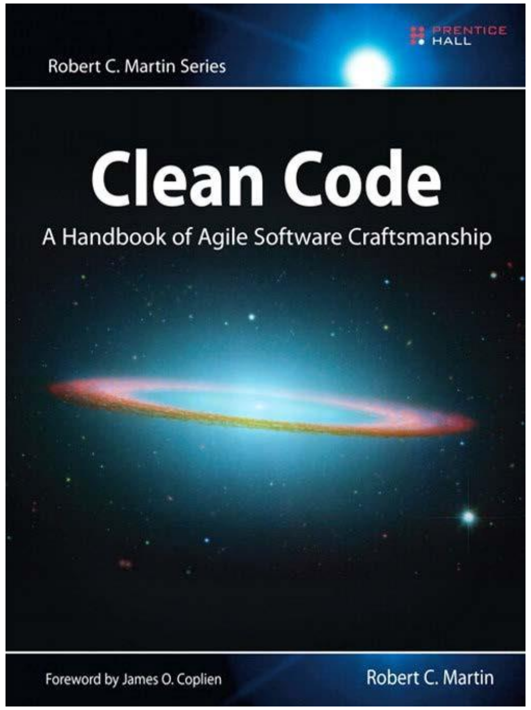
References

Clean code course by Cory House.

Steve McConnell
stevemcconnell.com


Microsoft
CODE COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell
Two-time winner of the Software Development Magazine Jolt Award

Robert C. Martin
objectmentor.com


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Andrew Hunt, David Thomas
pragprog.com