IS THIS YOUR CODING PHILOSOPHY?

“I DON’T KEEP MY CODE CLEAN, I JUST TURN THE LIGHTS OFF!”

Agile 2014 Conference
Orlando, FL USA

Scott Dillman
Sr. Director Analysis & Design
Travelers Insurance
August 1, 2014
Agenda

What is clean code?

Why clean code?

How do we write clean code?

Live Refactoring
What is clean code?

clean code reads like a book
readable by others
expressive
does one thing
no surprises
elegant
minimal
looks taken care of
efficient
has tests
small abstractions
no duplication
has tests
read like a book
minimal
does one thing
expressive
elegant
no surprises
Why clean code?

Messy Code:

• Slows us down
• Makes us more likely to make mistakes
• Is very hard for a new person to learn
• Makes it difficult to add new business functionality
Why isn’t your code clean?

Have you ever heard answers like this?

- Our managers didn’t let us do it right!
- Too much schedule pressure!
- Requirements changed and screwed up our design!
- Somebody else must have messed it up!
- I’ll clean it up later!
We know messy code slows us down

yet…

We give in to pressure to make messes in order to meet deadlines
So...why isn’t your code clean?

Now are the answers you hear more like this?

I want to write clean code but I don’t know how.

I don’t feel like taking the time.

What’s the difference anyway?

Is that really part of my job?

Is it Friday yet?
The Solution

- Software Craftsmanship
- Professionalism
- Pride
- Discipline
- Courage
How do we write clean code?

- Documented coding standards
- Code reviews
- Tool support
- Automation
- Clean-code categories
  - General
  - Naming
  - Comments
  - Variables
  - Functions
  - Classes
  - Tests
General

• No duplication
• No dead code
  – No commented-out code
  – No unused code
Naming

- Follow naming standards for projects, namespaces, and test methods
- No abbreviations (unless common business terms)
- Use intention-revealing names
  - NO: `int d; // elapsed time in days`
  - YES: `int elapsedTimeInDays;`
- Avoid disinformation
  - NO: `Hashtable accountList;`
  - YES: `Hashtable accounts;`
- Avoid “noise” words
  - NO: `PolicyInfo, PolicyData`
  - YES: `Policy`
- Use searchable names
  - NO: `int i = 0;`
  - "The length of a name should correspond to the size of its scope."

True story…

What should we name your new baby brother?

Batman Baby!
Comments

• Include good comments
  – Comment at top of every class/interface documenting the responsibility
  – Comments that focus on the “why”, not the “what”. Good naming should take care of the “what”

• Do not include bad comments
  – Redundant or “noise” comments
    // Processor delay for this component
    protected int processorDelay;

  – Misleading comments
    // Return when this.closed is true
    public void waitForClose(long timeout) {
      if (closed) return;
      wait(timeout);
      if (!closed) throw new Exception();
    }

  – Closing-brace comments
  – Commented-out code
Variables

• Instance variables grouped together at top of class

• Local variables appear as close to where they are used as possible

• String and number constants are defined as constant instance variables

• What about a class dedicated to constants? Dependency Magnet!
Functions

- Small – really small!
- Do only one thing (try/catch counts as one thing)
- Have descriptive names
  
  `assertEquals()` vs. `assertExpectedEqualsActual()`

`Verb(Noun): Render(page), Save(policy)`

- Don’t return null
- Have no side effects

```csharp
public bool CheckPassword(string password) {
    if (ValidPassword(password)) {
        InitializeSession();
        return true;
    }

    return false;
}
```
public void BuildHouse() {
    BuildFoundation(); // Level 1
    BuildWallFrames(); // Level 2
    CoverWallFramesInDrywall(); // Level 2
    PaintWalls(); // Level 3
    HangPicturesOnWalls(); // Level 3
    BuildRoof(); // Level 1
}

public void BuildHouse() {
    BuildFoundation();
    BuildWalls();
    BuildRoof();
}

public BuildWalls() {
    BuildWallFrames();
    CoverWallFramesInDrywall();
    DecorateWalls();
}
Functions - Arguments

• Have few arguments (3 or less)
  – Can force you to pay attention to details at a different abstraction level from the function
  – Increase the number of test cases needed to fully test the function
• Have no flag arguments – they indicate a function does more than one thing
• Don’t pass null arguments
• Avoid output arguments

    AppendFooter(string report)

    report.AppendFooter();
Functions – Command Query Separation

- Functions should either do something or answer something, not both

```cpp
public bool set (string attribute, string value); // Returns success or fail
```

```cpp
If (set (“username”, “scott”))
```

Is this asking if the username is currently set to “scott”?

Is this asking if the username was successfully set to “scott”?

```cpp
If (attributeExists (“username”))
    setAttribute (“username”, “scott”);
```
Classes

• Small
• Do only one thing
• Code written to be read top to bottom
Tests

• All ‘Function’ rules apply to test functions as well
• One concept per test
• Cover all paths through code

Find $x$

3 cm

4 cm

Here it is!
LIVE REFACTORIZING
Questions

“Quality is the result of a million selfless acts of care…”  
James O. Coplien

Session Feedback
You can do so in 3 ways:
1. Visit this session on the Mobile App. Click Session Feedback.
2. Scan the unique QR Code for this session located at the front and back of the room.
3. Visit the unique URL for this session located at the front and back of the room.

Thank you for providing your feedback 😊

References