Access Slides

Go to this link: https://swipe.to/6340r

Becoming a Better Programmer

Chaochih Liu and Peter Morrell

July 8th, 2016

Part 1: Write Less Code!

Write Less Code!

Remove unnecessary code!

- Takes up extra storage
- Clutters revision control histories
- Gets in the way of development
- Uses code space

Quality vs. Quantity

Writing more code does not mean more software!

Why does this matter?

- More code to read and understand
- More work to modify
- More places for bugs
- Duplicated code is difficult to troubleshoot

Code Duplication

Strongly advise against: cut-and-paste coding…

…especially if code is copied with slight changes

Example of Copying and Pasting

Example of Avoiding Code Duplication

Discussion Question

How different does a section of code have to be before it is justifiable to not factor into a common function?

Question 3 from end of Chapter 3 (pg. 27)

Dead Code

- Code that never gets run
- Functions that are never called
- Variables that are written but never read
- Parameters passed to an internal method that are never used

Discussion Question

How can you spot and remove dead code?

Question 4 from end of Chapter 3 (pg. 27)

Comments

Carefully choose variable, function, and class names

Good structure

Not necesssary to duplicate information conveyed in variable/function/class names with comments.

Comments

Comments should explain **why**

Code itself explains **what** and **how**

Comments

Do not remove code by commenting it out

Example of Commenting Out Code

Discussion Question

What is a good comment?

When is commenting inappropriate?

Verbosity

Reduce effort required to understand code

Example of Verbosity

Lines 161-166 of Old version of sequence_handling

In Practice

Remove redundancy and duplication as you find it

Separate tidying up and functional changes

Discussion Question

Some coding standards mandate that *every* function is documented with specially formatted code comments. Is this useful? Or is it an unnecessary burden introducing a load of worthless extra comments?

Question 5 from end of Chapter 3 (pg. 27)

Part 2: Wallowing in Filth

Quicksand Code

Code quality benchmarks:

- Well named variable/function/class
- Neat and consistent code layout
- Simple and clear structure of cooperating objects
- Easy to find code that produces a certain effect

Discussion Question

Why does code frequently get so messy?

How can we avoid this from happening in the first place? Can we?

Question 1 and 2 from the end of Chapter 7 (pg. 61)

Tackling Quicksand Code

Questions to help identify the appropriate strategy:

- How long will you be working with this section of code?
- How frequently has the code been modified?

Decide if it is appropriate to tidy up the code.

Always leave the campground cleaner than you found it. - Robert Martin

Making Adjustments

Make changes slowly and carefully!

- Make separate commits for tidying up layout and functional changes
- Ensure tidying preserves existing behavior

Make sure functionality is not removed!

Discussion Question

What are the advantages of making layout changes separately from code changes?

Question 3 from the end of Chapter 7 (pg. 61)

Discussion Question

How many times have you been confronted with distasteful code?

How often was this code really dire, rather than “not to your taste”?

Question 4 from the end of Chapter 7 (pg. 61)