Writing Code that is Easy to Change

Jesse Bunch

Hello.

Jesse Bunch

Engineering Director

paramore.is

Co-founder
designers.mx

@bunchjesse

#cfneasychange

Writing Code that is Easy to Change

Why?

Excellence.

Steve Jobs

Peter Hallam

former

Microsoft engineer

78% Understanding

22% Coding

Why? To spend less time understanding and more time building.

Why? To spend less time debugging and refactoring.

Why? You never know when you'll need to present your code.

You should write your code as if it were going on display at any time.

Fact: Making your code easily readable/changeable is your first responsibility --- even before making it work.

That's why.

Elements of Easy to Change Code:

Easily read

Easily understood

Can be changed without fear

Has minimal side-effects

The Law of Demeter

Method *M* of Object *O*

- O's methods
- M's parameters
- Objects instantiated by M
- Direct component objects of O
- Global variables accessible by O, and in the scope of M.

Two Areas

Code Quality

Testing (Unit)

Code Quality

Over-Arching Theme:

Shorter/less is better.

Naming

Naming

- Your primary tool for communicating.
- Should reveal your intent.
- Shouldn’t need to read the code.
- Should be pronounceable and use proper parts of speech.
- Should be descriptive, but not too long.

Functions

Functions

- Must be short.
- Must have only one responsibility.
- Must be named well.
- Few arguments. Shoot for <= 3.
- No boolean arguments.
- No by-ref arguments.

Comments

Comments

- Are usually crap.
- Are a sign of failure to be expressive.
- Don’t usually respect DRY.
- Can become lies quickly.
- Don’t comment out code. Remove it.
- Should be rare.

Formatting

Formatting

- First impression of your code.
- Spaces or Tabs?
- Brace position?
- Use of whitespace? Logical.
- Use shorthand. Don’t abuse it though.
- Consistency is key.

Conventions

Conventions

- Are important.
- Are your common tongue amongst peers.
- Syntactical, Framework, Application, Team
- Should be followed.

Show and Tell

Design

Design

- Think through the design first.
- Think about use-cases.
- Sketch it out.
- Implement/respect boundaries. Partition well.
- Use a framework.

About Code Standards

Testing (Unit Tests)

How?

How?

- Tests MUST be written FIRST.
- Start with a failing test case.
- Write code to make it pass.
- Refactor.
- Rinse and repeat.

Benefits?

Allow fearless code change.

Provide low-level documentation.

Less bugs. Less debugging.

Test promote refactoring.

Discover simpler algorithms.

Bowling Game.

Conclusion

- Name things well.
- Write small, single-responsibility functions.
- Format your code consistently.
- Follow convention.
- Design you app well with good partitioning.
- Use test-driven development. Test First.

Questions?

Thank you.

- @bunchjesse
- http://getbunch.com/
- http://joind.in/6727