Writing Good Code

Raymond Xu
raymondxu.io

Lucas Schuermann
lvs.io

Code Quality

Just because your code works doesn't make it good

High quality code is:

- correct
- efficient
- idiomatic
- readable
- maintainable

Code Style

Design Patterns

Testing

Code Style

Design Patterns

Testing

What is Code Style?

Code Style primarily discusses:

- Structure
- Naming
- Idioms

These don't affect the behavior of your program, but rather the static organization

Why?

- Consistency
- Elegance
- Maintainability

Imagine you have just joined a new team and want to familiarize yourself with their codebase

Style Guides

A set of language-specific conventions that should be followed when writing code

Somewhat arbitrary

Consistency within a project is key

Google Java Style Guide

5.2.4 Constant names use CONSTANT_CASE: all uppercase letters, with words separated by underscores.

```
// Constants
static final int NUMBER = 5;

static final ImmutableList<String> NAMES =
ImmutableList.of("Ed", "Ann");

static final Joiner COMMA_JOINER = Joiner.on(',');
// because Joiner is immutable static final

SomeMutableType[] EMPTY_ARRAY = {};

enum SomeEnum { ENUM_CONSTANT }
```

Airbnb JavaScript Style Guide

7.1 Use named function expressions instead of function declarations.

```
// bad
const foo = function () { };

// bad
function foo() { }

// good
const foo = function bar() { };
```

Pep8 Python Style Guide

Write docstrings for all public modules, functions, classes, and methods. Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the def line.

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.
"""
```

Comments

Don’t explain obvious things

Explain why, not what

Always document

Linters

Linters are tools that reinforce style guides by pointing out all violations in your code

Code -> Test -> Lint -> Code Review -> Merge

Beyond Style

Many linters can also be used to check for common coding errors, especially in interpreted languages

This is called **static analysis**

Static Analysis

Particularly useful in dynamically typed interpreted languages, such as Python

Linters can help catch static errors like

- Invalid typing, naming, importing
- Use of deprecated features
- Other general coding problems

Usually done by compilers (Java, C, C++)

Linting Example

What errors might be returned by linting this file?

```
import json

def      f(x):
       y = x + 1

def g(y):
 return f(x)
```

Linting Example

Let's test with pylint...

```
(...)
W:  4, 0: Bad indentation. Found 7 spaces, expected 4
W:  7, 0: Bad indentation. Found 1 spaces, expected 4
C:  1, 0: Missing module docstring (missing-docstring)
C:  3, 0: Invalid function name "f" (invalid-name)
C:  3, 0: Invalid argument name "x" (invalid-name)
C:  3, 0: Missing function docstring (missing-docstring)
C:  4, 7: Invalid variable name "y" (invalid-name)
W:  4, 7: Unused variable 'y' (unused-variable)
C:  6, 0: Invalid function name "g" (invalid-name)
C:  6, 0: Invalid argument name "y" (invalid-name)
C:  6, 0: Missing function docstring (missing-docstring)
E:  7,10: Undefined variable 'x' (undefined-variable)
W:  6, 6: Unused argument 'y' (unused-argument)
W:  1, 0: Unused import json (unused-import)
```

Linting Example

And with pep8...

```
dyn-160-39-171-190:swe-talk-materials Lucas$ pep8 lint.py
lint.py:3:1: E302 expected 2 blank lines, found 1
lint.py:3:4: E271 multiple spaces after keyword
lint.py:4:8: E111 indentation is not a multiple of four
lint.py:6:1: E302 expected 2 blank lines, found 1
lint.py:7:2: E111 indentation is not a multiple of four
```

Linter Tools Comparison

Pylint gives verbose output, includes style warnings and coding errors

Pep8 gives (sometimes) shorter output, includes just style problems

Is there a way to combine and give only warnings/errors we care about?

Prospector

Prospector is a python linter that combines and truncates the output of a number of tools (pylint, pep8, etc.)

```
dyn-160-39-171-190:swe-talk-materials Lucas$ prospector
Messages
========

lint.py
  Line: 1
    pyflakes: F401 / 'json' imported but unused (col 1)
  Line: 4
    pyflakes: F841 / local variable 'y' is assigned to
but never used (col 8)
  Line: 7
    pyflakes: F821 / undefined name 'x' (col 11)
```

Takeaways

Read and follow style guides for your favorite languages

Incorporate a linter into your development workflow, especially when static analysis is necessary

Seek mentorship and code review to improve your intuition for clean code

Code Style

Design Patterns

Testing

What are Design Patterns?

General solutions to common software design problems

May vary slightly from language to language

Avoid anti-patterns!

Abstraction

Extracting physical implementation from method signatures

Key to extensible software design

Builder Pattern

Used to separate the creation of a complex object from its implementation

Provides more control over the steps of an object's construction, such as ease of parameterization

Strategy Pattern

Used to create algorithms with interchangeable components with behavior defined at runtime

- Define a family of algorithms
- Encapsulate each algorithm
- Interchange algorithms within the family

Algorithm varies with clients that use it

Composite Pattern

Goal is to allow groups of objects to be treated the same as individual objects

In general, algorithms operate on composite objects (groups) that contain multiple child objects

We can loop through all children and perform a similar action

Model-View-Controller

More complex pattern used in user interface design

Model: application logic and state
View: displayed to user
Controller: user interaction->updates model

Useful in organizing complex interactions between front/backend

Sample Code

https://github.com/cerrno/swe-talk-materials

Design Documents

Before you begin writing any code, write a design doc

Design docs are usually peer reviewed before development begins

Identify major problems at the design phase not the implementation phase

Takeaways

Design patterns are solutions to common problems

Familiarize yourself with as many as you can

Using a pattern is usually cleaner than a custom solution, and more readable for those working on your code in the future

Code Style

Design Patterns

Testing

But I know my code works...

But I know my code works...

- How do you know it works?
- What do you mean by “it works”?
- Does it really work?
- If you change something does it still work?
- How does everyone else know it works?

Types of Tests

- Unit
- Integration
- End-to-end
- Regression
- Smoke
- Sanity
- Load

Unit Tests

A unit test is a function that you write to test one small part of your project

Unit tests typically test a single case of a single function

Use assertions to define whether a test passes or not

Code Coverage

How much of your code is run by your tests?

More code coverage yields more confidence

How do you measure this?

Code Coverage

Function coverage

- Does the function get called?

Statement coverage

- Does every statement in the function get executed?

Branch coverage

- Does every branch get executed?

Condition coverage

- Does every boolean subexpression evaluate to both true and false?

Test Doubles

Test doubles enable isolation of behavior and decoupled tests

What if the function you want to test has many dependencies and calls other functions?

Stubs

Stubs simply return a hardcoded value

```
void testTransferMoneyBadPass() {
    Authenticator a = new Authenticator();
    TransferUtil t = new TransferUtil(a);
    String pass = "foo bar";
    when(a.validate(pass)).thenReturn(false);
    assertFalse(t.transferMoney(from, to, pass));
}

// in TransferUtil's transferMoney() function
if (!a.validate(pass)) {
    return false;
} else {
    // logic
}
```

Mocks

Mocks track and verify interactions

```
Authenticator mock; // created by mocking framework

void testTransferMoneyChecksPass() {
    TransferUtil t = new TransferUtil(mock);
    String pass = “foo bar”;
    t.transferMoney(from, to, pass));
    verify(mock.validate(pass), 1);
}

// in TransferUtil’s transferMoney() function
if (!a.validate(pass)) {
    return false;
} else {
    // logic
}
```

Fakes

Lightweight implementations for testing only

e.g. using an in-memory database

Testing Frameworks

Testing frameworks provide an organized and automated way to write and run tests

Examples:

- Java
  - Junit
  - Mockito
- Python
  - unittest
  - nosetest

A Simple Test

Our function

```
def mean(xs):
    return sum(xs) / len(xs) - 1
```

Our test(s)

```
def test_mean():
    assert mean([1, 2, 3]) == 2.0
    assert mean([-1.5, 0, 1.5]) == 0
```

Automated Testing

Use nosetest to run all “test-like” scripts

```
dyn-160-39-171-190:swe-talk-materials Lucas$ nosetests
.F
======================================================================
FAIL: unit_test.test_mean
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Library/Python/2.7/site-packages/nose/case.py",
line 197, in runTest
    (...)
    assert mean([1, 2, 3]) == 2.0
AssertionError

----------------------------------------------------------------------
Ran 2 tests in 0.005s

FAILED (failures=1)
```

Continuous Integration

Developers working each on their own copy of the codebase can quickly create a number of merge conflicts

Continuous integration (CI) is a system where code is integrated (merged) into the master **early and often**

CI relies on unit tests which verify code before it is integrated to ensure quality

Takeaways

Integrate automated testing frameworks into your projects

Practice thinking up test cases for your code and implementing them

Continuous integration is an example of a workflow that requires frequent testing-- a number of tools are popular on github etc.

Code Style

Design Patterns

Testing

Putting It All Together

Create a design doc before writing code, use design patterns as possible and conforming to style standards

Lint and run well-designed, high-coverage unit tests before submitting pull requests for review

ALWAYS require independent review before code goes live/into master

Resources

Clean Code

Effective Java

Pylint/Prospector Documentation

Nosetest Documentation

Travis CI Github Tutorials

Writing Good Code

Raymond Xu
raymondxu.io

Lucas Schuermann
lvs.io