CSC290 Communication Skills for Computer Scientists

Lisa Zhang

Lecture 6; October 21, 2019

Announcements

Design Review Presentation is this week!

- Slides are due tonight 11pm.
- Presentation during your tutorials.
- Attendance will be taken; you must attend all presentations in your tutorial.

Critical Review

Grading is slower than expected due to benchmarking.

- Critical Review Edits currently due Nov 7th
- If grading is too slow, we will push the deadline back so you have at least 7 days to complete the edits

Suggested Plan

- work on the coe commit after this class
- work on the documentation in the meantime, as you explain your work to your team members
- revise your documentation after next class

Mid-term Survey

I would like to get a sense of how you think the course is going, and what (if anything) you would like me to do differently.

Please spend 5-10 minutes to complete the mid-term survey. The survey is completely anonymous.

https://forms.gle/boWzg7CWyX7xLzvf7

Today

Everything you need to know for the **code commit** portion of your project.

- Writing clean code
- Writing commit messages

Clean Code

Writing clean code requires a shift in mentality

"Programs must be written for people to read, and only incidentally for machines to execute"

– Harold Abelson, Structure and Interpretation of Computer Programs

Writing clean code requires a shift in mentality

"Programs must be written for people to read, and only incidentally for machines to execute"

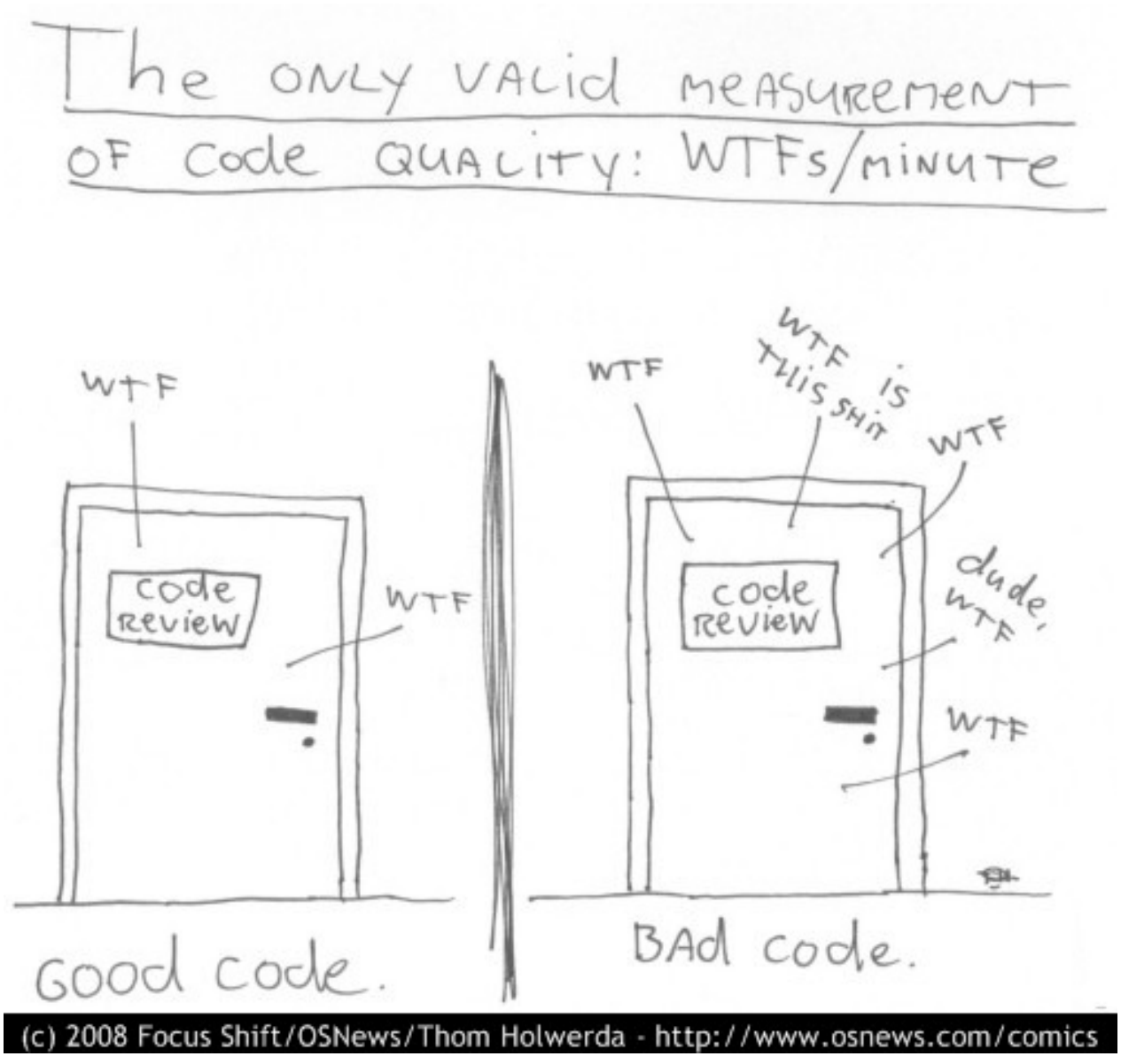
– Harold Abelson, Structure and Interpretation of Computer Programs

"Always code as if the [person] who ends up maintaining your code will be a violent psychopath who knows where you live. Code for readability."

– John Woods

What does clean code mean to you?

It's hard to define "clean code"



Code Samples

- There are four versions of a function.
- Which version has the "cleanest" code?
- Each version has some issues – what are they?
- Brainstorm: what does **clean code** mean to you?

Activity

Clean code should not stray from a reader's expectations. For example, clean code should

- follow the coding conventions for the project and language,
- use meaningful names,
- avoid duplications of code,
- and be self-explanatory with good documentation.

What do each of these mean? Give as many specific examples as you can of where the code samples follow or break these guidelines.

Following appropriate conventions

Different organizations will have different conventions. Different projects may have different conventions.

- https://google.github.io/styleguide/javaguide.html
- https://github.com/google/styleguide/blob/gh-pages/pyguide.md

Larger organizations will have more formal conventions.

- Use tools to automatically check whether your code follow conventions

For your project, use the standard conventions for your language (why?)

Python Conventions

- use `pothole_case` for variables, `CamelCase` for class names
- use four space tabs

Naming

Are the names used in verisons A-D good?

Function Names

- `f`: bad name, does not say anything
- `translate_to_piglatin`: good name, starts with a verb
- `piglatin`: okay name
- `english_to_piglatin`: descriptive name

Helper Function Names

- `index_of_first_vowel`: descriptive, but a bit long
- `first_vowel_index`: just as descriptive, and shorter
- `hasVowel`: good name (other than breaking convention)

Functions that Return a Boolean

- Functions that return a boolean often starts with “has” or “is”
- These are verbs used to ask a yes/no question

Variable Names: Version A

- `w`: bad name, hard to search
- `i`: borderline, still hard to search

Naming Consideration

- Does the name fully and accurately describe what the variable represents?
- Name should have the right level of specificity
  - The larger the scope, the more specific the name
  - Reserve single characters names for short loops only
  - Use `i`, `j`, `k` for integer loop indicies (why not `l`?)
- Name should be easy to **search** (e.g. global search and replace)

Variable Names: Version B

- `word`: good name
- `i`: borderline, hard to search

Variable Names: Version C

- `vowels`: good name
- `i` in `index_of_first_vowel`: okay
- `i` in `piglatin`: not okay!

Variable Names: Version D

- `VOWELS`: reasonable name for a constant
- `vowel`: good name
- `str`: very bad name, because `str` means something in Python!

Avoid common, meaningless names

- flag
- status
- data
- variable
- tmp
- foo, bar, etc.

Duplication

```
# Version A has a lot of duplication:

if (w[0] == "a" or w[0] == "e" or w[0] == "i" or
                   w[0] == "o" or w[0] == "u"):

# Copy & paste introduces error

while i < len(w) and (w[i] != "a" and w[i] != "e" and
                      w[i] != "i" and
                      w[i] != "o" and w[i] == "u"):
```

Did you notice it?

Abstraction is Better

Instead of:

```
while i < len(w) and (w[i] != "a" and w[i] != "e" and
                      w[i] != "i" and w[i] != "o" and
                      w[i] == "u"):
```

Write:

```
while i < len(w) and all(w[i] != v for v in "aeiou"):
```

Or write a helper function as in Version C & D.

Reduce code repetition

```
def make_egg():
    egg = take_out("egg")
    cooked_egg = cook(egg)
    plated_egg = plate(cooked_egg)
    return plated_egg

def make_ham():
    ham = take_out("ham")
    cooked_ham = cook(ham)
    plated_ham = plate(cooked_ham)
    return plated_ham
```

Don't rewrite the builtins

```
def round(num):
    frac = num % 1
    if frac >= 0.5:
        return (num - frac + 1)
    return (num - frac)
```

Don't re-write code that other people in your project have already written.

Reduce nesting (exit early)

This code is **worse** than the one in Version C:

```
def piglatin(word):
    i = index_of_first_vowel(word)
    if i != len(word): # has vowel
        if i == 0:
            return word + "way"
        else:
            return word[i:] + word[:i] + "ay"
    else:
        return word
```

If possible, edit early and get rid of the easy cases

Writing Testable Code

- **Unit test** verifies the behaviour of a small part of your code
  - Easy to write and run
- **Integration test** verifies that components interacts well with each other
  - Difficult to write and run

So what makes code easier to test?

Writing Testable Code

- **Unit test** verifies the behaviour of a small part of your code
  - Easy to write and run
- **Integration test** verifies that components interacts well with each other
  - Difficult to write and run

So what makes code easier to test?

- Each function should do one thing only.
- Isolate functions that interact with external systems (file system, database)
- Prefer **pure** functions
  - Function whose output is deterministic given its arguments

Code that is difficult to unit-test

```
def read_file_and_compute_total(file):
    total = 0
    for line in open(file):
        item, price = line.split(",")
        price = float(price)
        if item not in FOOD_LIST:
            total += price * 1.13
        else:
            total += price
    return total
```

Code that is easier to unit-test

```
def read_product_price(file):
    products = []
    for line in open(file):
        item, price = line.split(",")
        products.append(item, float(price))
    return products

def compute_total(item, price):
    if item in FOOD_LIST:
        return price
    return price * 1.13

def read_file_and_compute_total(file):
    return sum([compute_total(item, price)
                for (item, price)
                in read_product_price(file)])
```

Comments

- Comments should **explain why** the code is what it is.
- Comments should never repeat the code.
- Ideally, the code will make sense without any comments.

Comments that repeat the code are bad!

```
def translate_to_piglatin(word):
    if word[0] in "aeiou":  # first character is a vowel
        return word + "way" # return the word + "way"
```

These are useless comments!

Comments that explain the code

```
def piglatin(word):
    i = index_of_first_vowel(word)
    if i == 0: # begins with vowel
        return word + "way"
    if i == len(word): # no vowel
        return word
    return word[i:] + word[:i] + "ay"
```

These are better comments.

Comments that mark the code

```
VOWELS = "aeiou" #TODO: include y?
```

. . . but clean these up, ideally before committing.

Other comments:

- Block comments to lay out code
- Comments that describe the code's intent
- Comments that summarizes a chunk of code
- Information like copyright notices, references, etc.

Grammar: Conciseness

What is wrong with the following sentence?

The authors come to the conclusion that technical skills on their own are not enough to be hired by businesses these days.

What is wrong with the following sentence?

The authors come to the conclusion that technical skills on their own are not enough to be hired by businesses these days.

Wordy.

What is wrong with the following sentence?

The authors come to the conclusion that technical skills on their own are not enough to be hired by businesses these days.

Wordy.

- The authors **conclude** that technical skills alone is not enough to be hirable ~~by businesses these days~~.

Shorten the following sentence:

On the basis of the data gathered, the article supports the reasoning that having non-technical skills will help an individual obtain IT related jobs.

Shorten the following sentence:

On the basis of the data gathered, the article supports the reasoning that having non-technical skills will help an individual obtain IT related jobs.

- The data supports the authors' hypothesis that non-technical skills help individuals obtain IT related jobs.

Conciseness

- Verbs are more concise than nouns
  - “come to the conclusion” vs “conclude”
- Active voice is more concise than passive
  - “the conclusion is supported by the data” vs “the data supports the conclusion”
- Positive statements are more concise than negative
- Many expressions (e.g. “the fact that”) are unnecessarily long

Make these sentences more concisce

- In spite of the fact that most people think they write well, study shows that they do not.
- The most current trends in the industry can have a really significant impact on programmers' approaches to coding.
- Outsourcing is not always the best strategy for your business, especially if the business being run is considered small.
- The provided PyTA script can also be used for type checking purposes.

Commits and Commit Messages

What is a “commit”?

- Small set of modifications to a code base
  - Each commit should contain one (atomic) change
  - Commits should be standalone (independent of other commits)

Open Source Examples

- Chromium
  - https://github.com/chromium/chromium/commits/master
- NumPy
  - https://github.com/numpy/numpy/commits/master
- Evennia (python text-based game library)
  - https://github.com/evennia/evennia/commits/master

Commits

- Do not fold small changes (e.g. typo fixes) into another commit
- When commits are atomic and standlone, they can be applied and reverted independently
- The **commit message** summarizes the code changes
- Makes the version control utilities much easier to use

Commit Messages

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Guidelines

- Commit message guidelines differ by company.
- Bigger organizations programmatically read/modify commit messages.
  - Chromium's commit messages have a lot of boilerplating.

Common guidelines

From https://chris.beams.io/posts/git-commit/

- Separate subject from body with a blank line.
- Limit the subject line to 50 characters.
- Capitalize the subject line.
- Do not end the subject line with a period.
- Use the imperative mood in the subject line.
- Wrap the body at 72 characters.
- Use the body to explain what and why vs. how.

Example:

You add the following lines to a file called “tictactoe.py”

```
+ # Python TicTacToe game on the command line
+ # Author: Mr. Pirate <mr@pirate.com>
```

What should your commit message be?

Example:

What is wrong with the following commit messages?

```
Added comments to tictactoe
author and file description in tictactoe.py
comment description and author and email to the first few l
Add description, author, and email.
```

Your Project Commits

- Should be non-trivial (can't just be fixing a typo)
- Should be adding a major functionality
- Should have a commit message that follows the "Common guidelines"
- Should have code that follows the coding guidelines discussed today

Reviewing Code

Code Review

Most companies use "code review" to ensure high code quality.

1. Code writer submits code for review.
2. One or more reviewers (peers) read the code.
3. If reviewers notice areas of improvement, reviewers will request for changes.
4. Code writer works with the reviewer to address any raised issue (back to step 2)
5. When all reviewer concerns are addressed, the code is accepted (pushed).

Why code review?

- Encourage committers to write clean code.
- Share knowledge across team members.
- Encourages consistency in the code base.
- Help prevent bugs and other issues.

In most large organizations, **all** code, no matter who wrote it or how large/small it is, need to be reviewed.

What does the reviewer do?

- Does the code accomplish the author's purpose?
- What is the author's approach? Would you have solved the problem differently?
- Do you see potential for useful abstractions?
- Do you spot any bugs or issues?
- Does the change follow standard patterns?
- Is the code easy to read?
- Is this code documented and tested?

Example:

- https://github.com/evennia/evennia/pull/1666
- https://github.com/numpy/numpy/pull/11721
- https://github.com/numpy/numpy/pull/10931
- https://github.com/numpy/numpy/pull/10771

How to Review Code

- Critique the code, not the author.
  - "*your* code has a bug" vs "the code has a bug".
- Ask questions (perception checking!).
- Reviews should be concise and actionable:
  - Make it clear what you are asking for
- Don't be mean.

Responding to Code Review

- Be civil, and be open minded.
- First time you submit code, you will have *many* comments, don't feel daunted.