Naming & Code Reviews

Slides adapted from Craig Zilles

Style & Design

- A key focus of this class is developing your sense of style and design with respect to code.

- Why do we care?

What metric should we use?

- **Book proposes:**
  - Code should be written to minimize the time it would take for someone else to understand it.

Names (variables, functions, tests, etc.)

- **Why are names important?**

Which is NOT a book recommendation

A) it is better to be clear/precise than cute

B) prefer concrete names over abstract names

C) for measurements, encode units into variable's name

D) you should never use i, j, or k for loop variable names

E) throw out unneeded words

Boolean Variables

- **Which name is most problematic for a Boolean variable**

A) done

B) notFound

C) processingComplete

D) isValidFormat

E) hasChildren

What is the best name length?

- Software engineering researchers did a study and found that the effort to debug a program was minimized when variables had names that averaged in a given size range. Can you guess what the range was?

A) 1-8 characters

B) 5-12 characters

C) 10-16 characters

D) 15-22 characters

E) 18-30 characters

Should scope affect variable name length?

A) Yes. Variables with larger scope should have longer variable names.

B) No. Variables should be named independent of their scope.

C) Yes. Variables with larger scope should have shorter variable names.

Magic Numbers

- Generally, you shouldn't have hard-coded numbers in your code (what are called "magic numbers") other than 0 and 1.

```
// check if password is too short
if (password.length() < 5) { …
```

CS 126 Style Guide (for Java)

- **Google Style-guide:**
  - Real
  - Good style
  - Sets clear expectations to students and moderators
  - One instance of good style, not the only one

- **https://google.github.io/styleguide/javaguide.html**

More about tests and re-factoring

A few words about Code Reviews

Code Reviews start this week!

- **Check your SVN for a file: moderator_assignment**
- **Bring a laptop.**
  - And display adapters (to VGA or HDMI) as necessary
- **Be ready to present your code.**

Your job when presenting

- **Think about your presentation before your code review**
  - What are the most important things to show?
  - What is the logical order to show things?
  - How much time should be spent on each thing?

- **Be aware of your audience (moderator AND fellow students)**
  - Speaking loud enough and clearly
  - Appropriate pacing
  - Observe audience and adjust as necessary

Your job when presenting (2)

- **Manage time:** ***be prepared!***
  - 2 hours / 6 students = 20 minutes
  - Need time to present and for feedback / questions
- **Focus on what is important:**
  - What is on the rubric? Be sure to show those things.
  - Focus on what is new in each assignment.
  - What did you need to improve from last time? Show fixed.
- **View this as a learning opportunity, not an evaluation:**
  - Try not to be defensive; take suggestions to heart.
    - But don't overweight one-off comments. 'Grain of salt'
  - I know this is hard.

Your job when others are presenting

- **Pay attention. Participate! Respectfully.**
- **What can you learn from their implementation?**
  - What idea is in their code that you could later in life?
  - It is okay to ask questions.
    - Why did you choose to use recursion over iteration in function X?
- **What feedback can you give to your fellow students?**
  - Make sure that your intention is to help them.
  - Be modest. Not everything you think is true may be true
    - Learning can happen when others disagree with your suggestions
- **Be sure to respect their time.**

How to give constructive criticism?

- **Intermix criticism within praise:**
  - "I think your function names are really good, but your global variable names seem too brief to provide enough context."
  - "I like your structure, but the 2nd loop seems overly complex."

- **Focus on the code not the person:**
  - **NO** "You always write loops that are too large"
  - **YES** "I feel that this loop is larger than it has to be"

- **Use first person statements:** *state as opinions not facts.*

How to give constructive criticism? (cont.)

- **Be specific with your feedback:**
  - **NO** “I think the names of your variables are bad”
  - **YES** “The variable name ‘temp’ doesn’t show its intent”

- **Give actionable feedback:**
  - “I would suggest simplifying the code by pulling out lines 42-57 into its own function that is called from the loop.”

Everyone in this class was admitted to CS

- **Your (and others's) admission was not a mistake**
- **You all are expected to graduate and succeed**
  - This will take work on your part, but you can do this.

Computing Needs Diversity

- **Computing is going to be deployed in society in every aspect of industrial and personal needs.**
  - We need people from every part of society to make sure it is done in a way that meets all of society's needs

- **Diverse groups are more innovative.**
  - https://www.scientificamerican.com/article/how-diversity-makes-us-smarter/
  - "This is not only because people with different backgrounds bring new information. Simply interacting with individuals who are different forces group members to prepare better, to anticipate alternative viewpoints and to expect that reaching consensus will take effort."

Expectations in CS

- **Don’t make discriminatory remarks.**
  - Discrimination is about putting people down and keeping them down so we can more safely exploit them in future. Or, so they will not compete with us. Or, simply to feel superior.
  - There is no excuse for it.
  - Not even in “private”; don’t normalize this behavior.

- **Call out other people when they make discriminatory remarks**
  - Make it clear to others that such behavior is unacceptable.
  - Especially if you are not part of the targeted group.
  - Be a positive force.

Link to CS196 Slides

- https://docs.google.com/presentation/d/1f2s1mysMlrpdEMVNDF_Y6NjqdWZs-H4r5PQ3uBPG1oM/edit#slide=id.g165324e2d7_2_102