Testing and Debugging

CSE 373 Winter 2020

Instructor: Hannah C. Tang

Teaching Assistants:

Aaron Johnston
Amanda Park
Anish Velagapudi
Brian Chan
Elena Spasova
Ethan Knutson
Farrell Fileas
Howard Xiao
Jade Watkins
Lea Quan
Nathan Lipiarski
Sam Long
Yifan Bai
Yuma Tou

Announcements

- Extra Drop-in-Times this Saturday; see Piazza for times/locations

- We will cease issuing Gitlab accounts (required to do HW1) on Sunday at 6pm. If you aren't registered for this class but hope to do the homeworks, email cse373-staff@cs *immediately*.

Questions from Reading Quiz

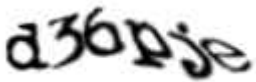
- What does “reproducing the bug” mean?
- How do I write a good unittest?
- How do I write good code-under-test?

Lecture Outline

- **Reproducible bugs and minimum working example**
- Live debugging!

Step 1: Reproduce the Bug

- Story time!



- If you're going to use statistics, then you need to *think about statistics*
  - Eg, What is your underlying population?
  - Eg, Mean vs median?

Step 2: Minimum Working Example

- Reduce your "bug reproduction" so that it's as simple and as fast as is (reasonably) possible
  - Can you make the text file/string shorter?
  - Will 2 iterations show the bug as effectively as 1000 iterations?
  - Does the user need to be logged in?
  - Can the server run without a database?
  - Do you have test data instead of production data?

  - Can you write a short program that demonstrates the bug?

Lecture Outline

- Reproducible bugs and minimum working example
- **Live debugging!**

Example from Reading

- Load a text file from disk
- Divide the text file into "words"
  - For each word, normalize it (remove punctuation, fix capitalization)
  - Insert into a HashSet to deduplicate
- Print the number of unique words in that text file

Tips: Writing Testable Code and Test Code

- Keep your "units" small and as independent as possible
  - A unit is something that should be tested (eg, class, method, etc)
  - `main()` should be where you "glue" your units together
  - If units have dependencies on eachother, pass them in as arguments instead of doing the instantiation internally
    - Eg, `WordCollection` takes a `WordNormalizer` as an argument
      - This technique is known as "dependency injection"
  - Keep the arguments as lightweight as possible
    - `WordCollection` takes the file contents as an argument, not the name of the file, because it doesn't care about the file's properties.
      - This is known as "the law of demeter"
- Write a collection of *test cases* to verify your units
  - Organize your test cases into *test suites*

Tips: Debugging

- Try to find a case that is consistently reproducible
  - Then try to simplify it until it's as simple as possible ("minimal working example")

- Two techniques for using your minimal working example:
  - Form hypothesis; use debugger/print statements to gather information which confirms or denies hypothesis; repeat
  - Identify your invariants; write helper method(s) verifying them; sprinkle broadly in your code until you find where they're broken; repeat