Debugging

CMPT 145

Where do software errors come from?

- There are two ultimate sources of software error:
  1. Not writing the code you intended to write.
     - Typographical errors
     - Minor errors in logic.
  2. Not knowing what code to write.
     - Design was bad.
     - No design.
     - No plan.
- Debugging can only help with #1.

Debugging attitude

- To find out what your program is doing wrong, you need to know what your program is doing.
- Do not assume that an error has a quick or easy fix.
- Errors are almost always where you are not looking.
- Challenge your own assumptions.
- Assume everything is broken.

Scientific Debugging

- The output of your test script is data.
- It tells you something went wrong. It does not tell you what the error is.
- Do not make a change to your code too soon

1. Make a hypothesis about the error. Write it down.
2. Create new test cases that should fail if your hypothesis is true.
3. Run the new test cases to verify your hypothesis.
4. Only change the code after you have gained confidence in your hypothesis.
5. If your fix eliminates the errors, you were (probably) right.

Debugging techniques

- For faults discovered by unit testing:
  - Use the debugging tool
  - Careful reading of the function/unit you are testing
- For faults discovered by integration testing:
  - Print statements (Wolf-fencing) to narrow down the location of the error.
  - Set break-points to use the debugging tool on small regions of your code.
- For faults discovered by system testing:
  - Do not debug the application.
  - Identify the conditions that cause the fault.
  - Create an integration test with those conditions, and debug that!

Debugging: Wolf-fencing

- An error in module A might not be cause a fault until module B gets the incorrect data.
- You don't have the time to use the debugger across a whole application.
- Add output to your program, e.g.,
  - `print` statements
  - `assert` statements (Chapter 13).
  - (advanced) `logging output`
- Using a kind of binary search for the location of an error.
- Gather information about what happened.

Debugging non-Python languages

- Many languages (Java, C#, Ruby, etc) have run-time errors like Python.
  - You will at least know where the run-time error occurred.
- Some languages (C, C++) do not.
  - A C/C++ program will halt abnormally without any warning or hint about why.
- Use wolf-fencing to find out where the program halted.
- Almost all languages have interactive debuggers, and they all have the same features. Learn the one you need for your work.

Debugging: Code Walk Throughs

- Your errors may be invisible to you.
- So explain your code to someone else.
- Professional software engineers review each others' code regularly.
- Pair programming is a good practice, when permitted by courses.
- If no one is around, use a puppet. Silly, but it works.

The difference between you and a professional

- Professional developers are learning new technologies all the time.
  - They are not done learning.
- The difference between you, and accomplished software developers is
  - Not knowledge
  - But experience
- Be patient with yourself to gain that experience while you are learning new things.