Lecture 7: Testing and Debugging

Armando Solar-Lezama

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Testing and debugging

How do you know your code is correct?

- Black box testing
- White box testing or Glass box testing

So it's not correct; how do you fix it?

- Debugging with asserts
- Debugging with the scientific method
- Print

Problem: Break a word into syllables

Simple algorithm

- not entirely accurate, but simple and effective

Specification

- Introduce a break halfway between any two vowels that have one or more consonants between them
  - VCCV -> VC-CV
  - No break required if there is no consonant between vowels
- If there is an odd number of consonants, break before the middle consonant
  - VCV -> V-CV
  - VCCCV -> VC-CCV
- if the algorithm breaks ‘sh’ or ‘th’ keep the ‘h’ next to the ‘s’ or ‘t’

Black box testing

Create test cases without looking at the code

- focus on corner cases in the specification

Some corner cases

- Cases the specification says should be handled differently
  - Word with multiple vowels together
  - Word with even and odd number of consonants
  - Word with ‘sh’ or ‘th’
- Corner cases in terms of size of the input
  - Single letter words
  - Empty words
  - Word without consonants
  - Word with one vowel
  - Word with no vowels

Some testing terms

Test first programming

- Create the test before writing the code
- Helps you think about corner cases in the specification

Regression testing

- Have an automated test suite that you can run as you develop
- When you find a bug, add a test that exposes it to the test suite
- Helps you find if you are reintroducing errors you have fixed before.

Some test cases for syllable()

Code can be found here:

```
def tester(fun):
    out = fun('cooperate')
    print out
    if(out != ['coo', 'pe', 'ra', 'te']):
        print "ERROR!!"
        return False
    return True
```

Take function as a parameter

Automatically check for error; return false

If no error return true

Syllable Breaking Algorithm

Two phases

- buildSyllable
  - keep adding letters to the current syllable
  - transition to next phase when you find a vowel

- findNextVowel
  - find next vowel so you can find the midpoint between last and next vowel
  - if next vowel is right next to the last vowel, just make that the last vowel
  - if the midpoint breaks a ‘th’ or ‘sh’ increment it by one.
  - once you find the next vowel,
    - push letters between vowel and midpoint into the current syllable
    - add it to the list
    - start a new word with the remaining letters

Syllable Breaking Algorithm

State

- buildSyllable(word, result, csyll, cpos, size)
  - keep adding letters to the current syllable
  - transition to next phase when you find a vowel

- findNextVowel(word, result, csyll, lastVowel, cpos, size)
  - find next vowel so you can find the midpoint between last and next vowel
  - if next vowel is right next to the last vowel, just make that the last vowel
  - if the midpoint breaks a 'th' or 'sh' increment it by one.
  - once you find the next vowel,
    - push letters between vowel and midpoint into the current syllable
    - add it to the list
    - start a new word with the remaining letters

Recursive Structure

Find the correct algorithm at:

- http://bit.ly/WtSrO0

To run the tester, download the tester here:

- http://bit.ly/WtStWn
- Place the tester and algorithm in the same folder
- Run the tester

Debugging with the scientific method

Make a hypothesis

Design an experiment to try to invalidate hypothesis

- if you invalidate it, make a new hypothesis and repeat
- if you don't invalidate it, try another experiment

When you are confident you have found the bug fix it

- but not before that!

This is just a fancy way to say

"be systematic and don't go around making arbitrary changes to your code until you know what you are doing"

Debugging with asserts

assert check

assert check, message

Example: syllableBuggy1.py

Find it at http://bit.ly/XFfj0m

Program outputs syllable with two non-consecutive vowels

- buildSyllable could do this if called incorrectly

Hypothesis:

- buildSyllable is getting called incorrectly
- caller is breaking assumption about 'no vowel in csyll'

Experiment:

- Use assertion to test hypothesis
  - assert checkNoVowel(csyll), "Bad csyll " + csyll
- Experiment confirms hypothesis!

```
def checkNoVowel(word):
    for x in word:
        if(isVow(x)):
            return False
    return True
```

Have we found the bug?

- Not yet, but we know where to start searching.
- Result suggests that we need to look at calls to buildSyllable
- Adding checks to each call points to call inside findNextVowel as the culprit

Print Statements

Often a good way to test hypothesis

Some good things to print

- When you enter a function
- Parameters to that function
- Results produced by a function

Beware

- Make sure you know what you are looking for
  - Make sure you have a hypothesis in mind
- Otherwise you can stare mindlessly at print output without making any progress

White box testing

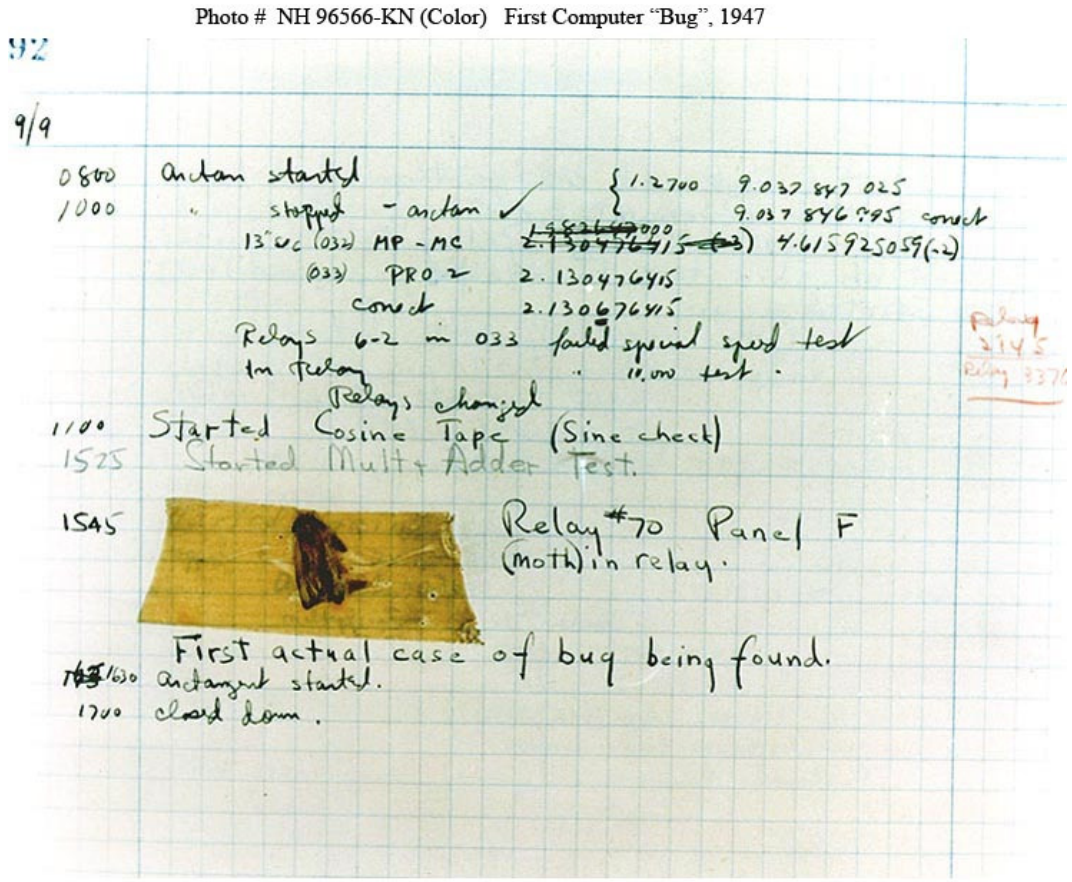
Design tests based on what you see in the code

- Look for unvisited branches
  - Is the programmer making special cases for things that shouldn't be special cases?
- Is there an input that you could produce that would exercise a path?

Look for suspicious code patterns

- If you see recursive calls, make sure to use test cases that could lead to infinite recursion
- If you see arithmetic used to compute a list index, make sure to have test cases that could trigger reads or writes outside the bounds
- Look for tests that would indicate off-by-one errors

The first computer bug



From DEPARTMENT OF THE NAVY -- NAVAL HISTORICAL CENTER:
"Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1945. The operators affixed the moth to the computer log, with the entry: "First actual case of bug being found". They put out the word that they had "debugged" the machine, ...
In 1988, the log, with the moth still taped by the entry, was in the Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia.

Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988.