Debugging

Shin-Jie Lee (李信杰)
Associate Professor
Computer and Network Center
Department of CSIE
National Cheng Kung University

What is BUG?

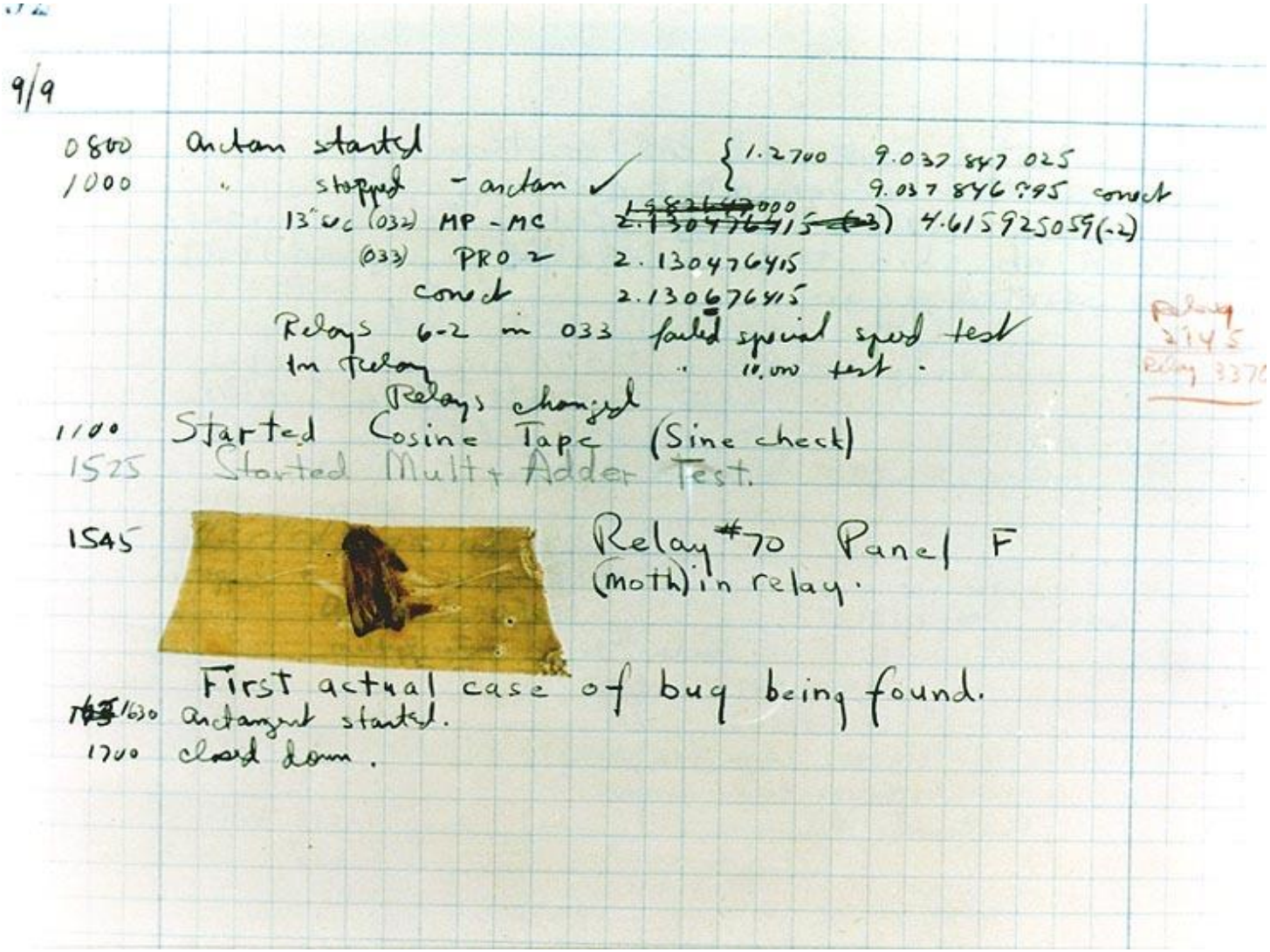
- Things the software does that it is not supposed to do, [or] something the software doesn't do that it is supposed to. [Telles and Hsieh]

- A **software bug** is an error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways. [From Wikipedia]

- 1. Synonym of *defect*. 2. Synonym of *failure*. 3. Synonym of *problem*. 4. Synonym of *infection*. [Andreas Zeller]

[Telles and Hsieh] Telles, Matt and Yuan Hsieh. *The Science of Debugging*. Scottsdale: Coriolis, 2001.
[Andreas Zeller] Andreas Zeller. Why Programs Fail, Second Edition: A Guide to Systematic Debugging, 2009

The First "Computer Bug" 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 1947.



Bug-Free Software?

Ineffective Approaches to Debugging$_1$

- **Find the defect by guessing**
  - Scatter print statements randomly throughout a program
  - If you can't find the defect with print statements, try changing things in the program until something seems to work
  - Don't back up the original version
  - Programming is more exciting when you're not quite sure what the program is doing

Ineffective Approaches to Debugging$_2$

- **Don't waste time trying to understand the problem**
  - It's likely that the problem is trivial, and you don't need to understand it completely to fix it
  - Simply finding it is enough

Ineffective Approaches to Debugging$_3$

- **Fix the error with the most obvious fix**
  - Fix the specific problem you see, rather than wasting a lot of time making some big, ambitious correction that's going to affect the while program.
  - An example:

```
x = compute(y)
If(y==17)
    x=25.15   -- compute() doesn't work for y=17, so fix
it
```

Ineffective Approaches to Debugging$_4$

- **Debugging by Superstition (The attitude in debugging)**
  - If you have a problem with a program you've written, it's your fault. It's not computer's fault, and it's not the compiler's fault.
  - Even if an error at first appears not to be your fault, it's strongly in your interest to assume that it is
  - **It's hard enough to find a defect when you assume your code is error-free**

The Scientific Method of Debugging

1. Stabilize the error (Refine the test cases that produce the error)
2. Locate the source of the error
   a. Gather the data that produces the defect
   b. Analyze the data that has been gathered, and form a hypothesis about the defect
   c. Determine how to prove or disprove the hypothesis, either by testing the program or by examining the code
   d. Prove or disprove the hypothesis by using the procedure identified in 2(c)
3. Fix the defect
4. Test the fix
5. Look for similar errors

Stabilize the Error

- The defect is easier to diagnose if you can stabilize it – that is, make it occur reliably

- To find test cases that produces the error

Tips for Finding Defects$_1$

- **Use all the data available to make your hypothesis**
  - When creating a hypothesis about the source of a defect, account for as much of the data as you can in your hypothesis
- **Refine the test cases that produce the error**
  - You might be able to vary one parameter more than you had assumed, and focusing on one of the parameters might provide the crucial breakthrough

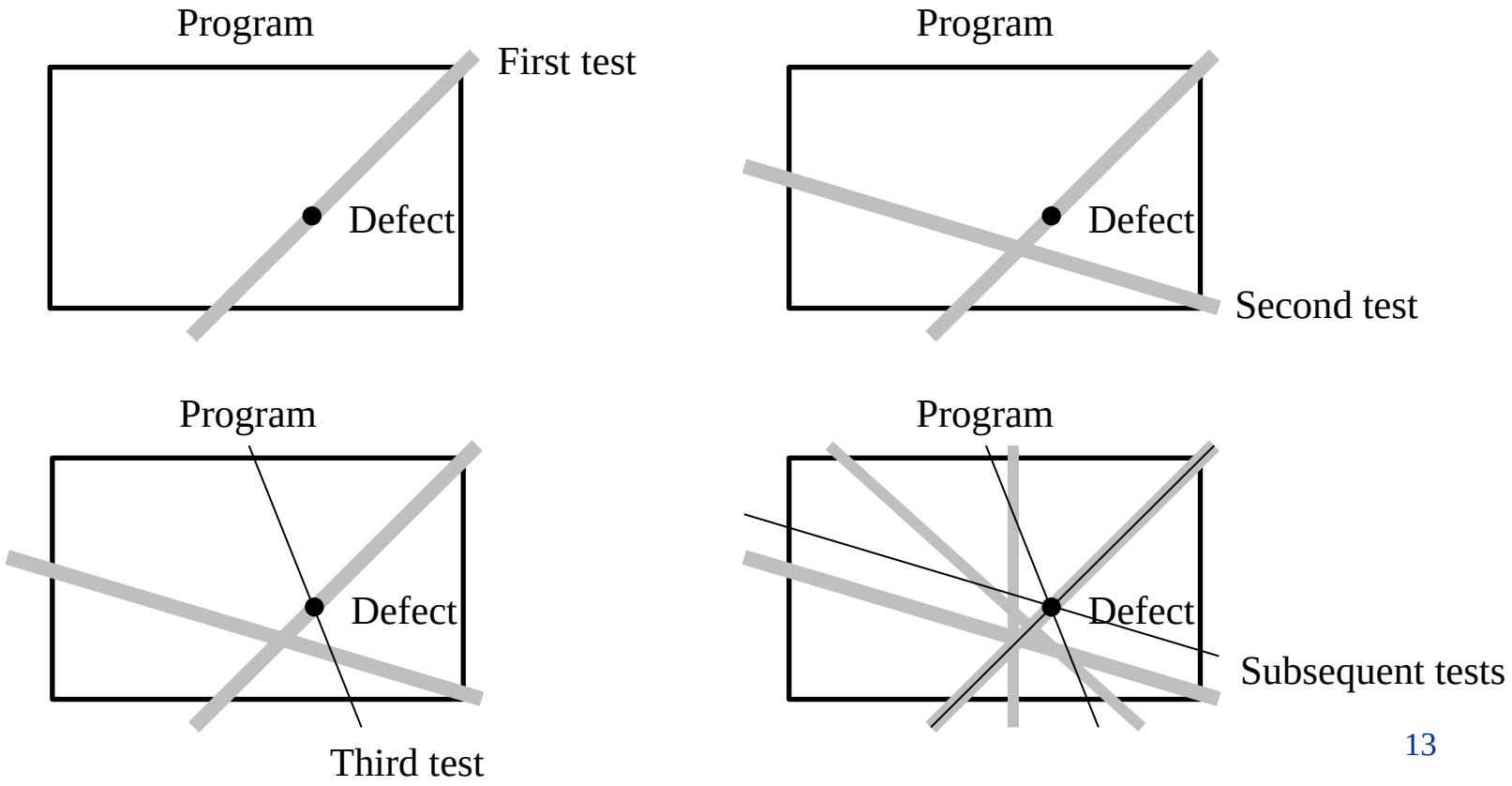
Tips for Finding Defects$_2$

- **Exercise (test) the code in your unit test suite**
  - Defects tend to be easier to find in small fragments of code than in large integrated programs. Use your unit tests to test the code in isolation.

- **Use available tools**
  - With one tough-to-find error, for example, one part of the program was overwriting another part's memory.
    - This error was difficult to diagnose using conventional debugging practices.
  - To use tool, for example, Eclipse Java Debugger.

Tips for Finding Defects$_3$

- **Reproduce the error several different ways**
  - If you can get a fix on it from one point and a fix on it from another, you can better determine exactly where it is.



Tips for Finding Defects$_4$

- **Generate more data to generate more hypotheses**
  - Choose test cases that are different from the test cases you already know to be erroneous or correct.
  - Run them to generate more data, and use the new data to add to your list of possible hypotheses.

- **Use the results of negative tests**
  - Suppose that a test case disproves your hypothesis, so you still don't know the source of error.
  - However, you do know that the defect is not in the area you thought it was. That narrows your search field and the set of remaining possible hypotheses.

Tips for Finding Defects$_5$

- **Brainstorm for possible hypotheses**
  - Rather than limiting yourself to the first hypothesis you think of, try to come up with several

- **Keep a notepad by your desk, and make a list of things to try**
  - One reason programmers get stuck during debugging sessions is that they go too far down dead-end paths.
  - Make a list of things to try, and if one approach isn't working, move on to the next approach

Tips for Finding Defects$_6$

- **Narrow the suspicious region of the code**
  - Rather than removing regions haphazardly, divide and conquer
  - Use a binary search algorithm to focus your search

- **Be suspicious of classes and routines that have had defects before**
  - Classes that have had defects before are likely to continue to have defects

Tips for Finding Defects 7

- **Check code that's changed recently**
  - If you can't find a defect, run an old version of the program to see whether the error occurs
  - Check the version control log to see what code has changed recently

- **Expand the suspicious region of the code**
  - If you don't find the defect in a focused small section of code, consider the possibility that the defect isn't in the section

Tips for Finding Defects$_8$

- **Integrate incrementally**
  - If you add a piece to a system and encounter a new error, remove the piece and test it separately

Tips for Finding Defects$_{10}$

- **Talk to someone else about the problem (confessional debugging)**
  - You often discover your own defect in the act of explaining it to another person
- **Take a break from the problem**
  - Sometimes you concentrate so hard you can't think
  - The auxiliary benefit of giving up temporarily is that it reduces the anxiety associated with debugging