

# Com S 417

## Software Testing

Fall 2017 – Week 13, Lecture 22

November 14, 2017

# Announcements

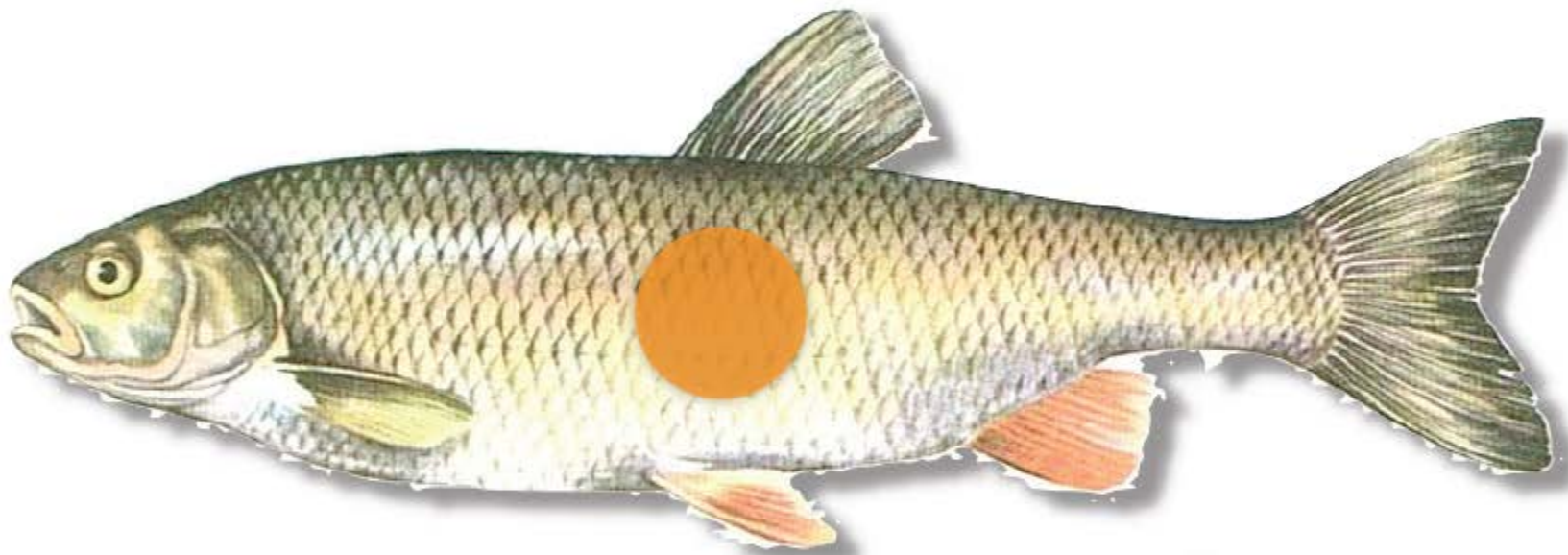
- Pezzè and Young, Chapter 16 (Fault-Based Testing) is available on digital reserve.
- Lab 5 is available.

# Estimating Defects

- How many defects remain in the SUT?
- It's the same problem as estimating how many fish are in a lake after we've caught some.



# Fish Tag

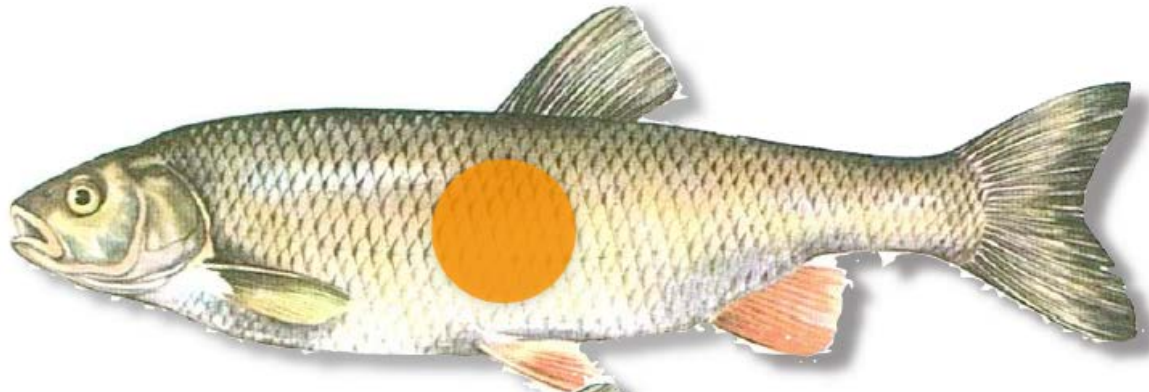


- We catch 1,000 fish and tag them

Over the next week we ask fisherman to track how many fish they catch and whether they are tagged.

# Counting Tags

50



300



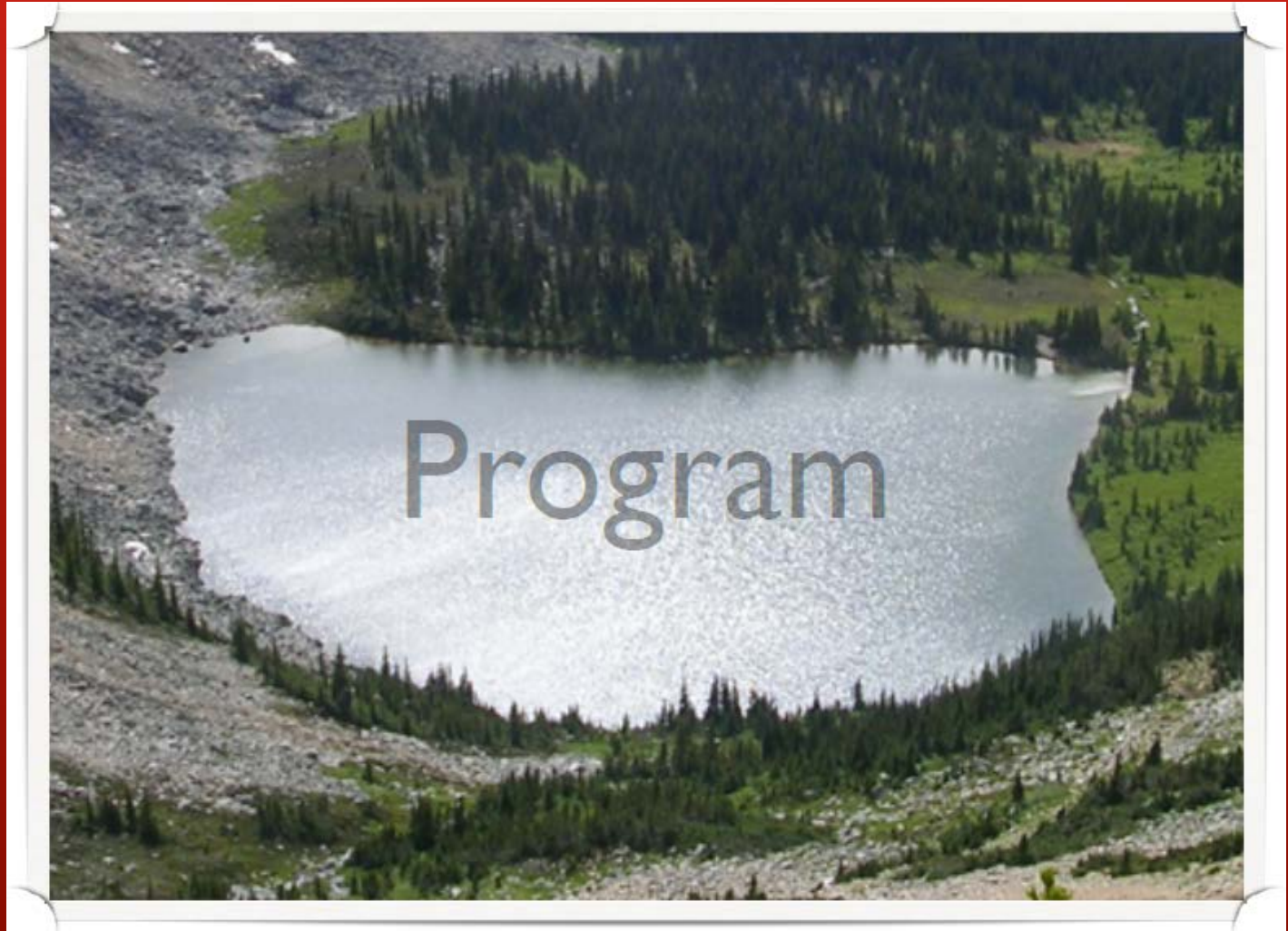
From the relative rates of capture and the knowledge of how many tagged fish are in the lake, we can estimate how many untagged fish were there at the beginning.

$$\frac{1,000}{\text{untagged fish population}} = \frac{50}{300}$$

Initial untagged Fish Population: 6000



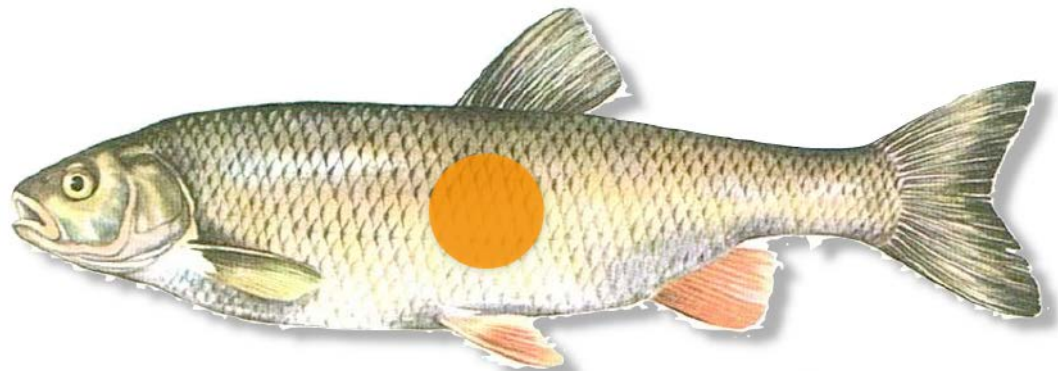
A program is like the lake ...



# We 'seed' some number of faults

Instead of tagging existing bugs, we create some new ones, and "throw them in" with the others.

## A Mutant



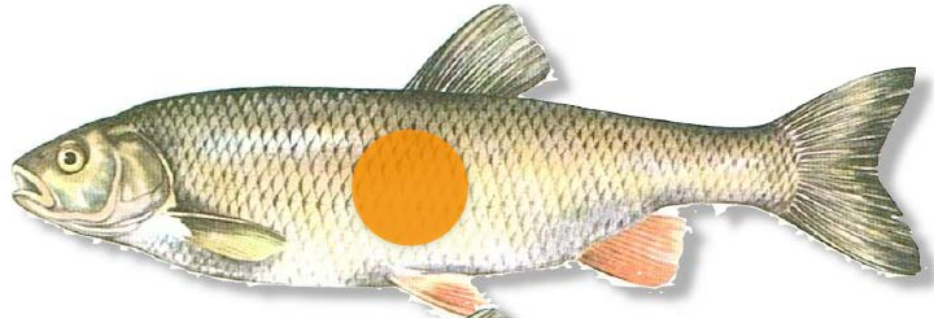
- We seed 1,000 mutations into the program



# We go fishing and keep track ...

Our test suite  
"catches" 50  
mutants and  
300 natural  
faults.

50



300



# Estimate

$$\frac{1,000}{\text{remaining defects}} = \frac{50}{300}$$

We conclude that there are 6000 remaining defects (including the 300 we just found.)

# Lot's of caveats:

- Mutants and natural faults are found at similar rates.
  - For example, we haven't already found and corrected all the easy bugs in the native population.
- Finding a mutant doesn't obscure finding a related natural fault.

# Why Study Debugging



- Debugging is hard
- Debugging is a big time sink
- Debugging is part of programming
- Debugging is a different subdiscipline than programming
- Debugging effectiveness can be improved by study and practice.

# Debugging is Hard



*“Debugging is an important skill that continues to be both difficult for novice programmers to learn and challenging for computer science educators to teach. These challenges persist despite a wealth of important research ... dating back as far as the mid 1970s.” [7]*

*“Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?”*

*Brian Kernighan and P.J. Plauger [2]*

# Debugging is Hard



Bugs can be an extreme example of a problem where effect is remote from cause. [4]

Research in System Dynamics has demonstrated that humans are intrinsically inept at solving such problems. [13]




# Debugging is a Time Sink



- 30% to 50% of developer's time is spent on debugging [9]

# Debugging success supports Programming success

- 
- in a pair programming experiment, pairs were more successful finding bugs and more likely to continue with programming studies. [25]
  - debugging challenges incorrect mental models (when properly mentored)
  - debugging skills place an upper bound on learning rate and change rate.

# Why study it separately



A discipline is determined by

- shared models
- shared commitment to certain values and practices
- shared library of exemplars

And usually captured (even codified) by textbooks.

Kuhn [8]


# Debugging != Testing

	Testing	Debugging
Core Models	Sampling	Experimental method
Goals	Measurement Risk & Cost	Localization Repair
Concerns	Quaility Risk Cost	Correctness
Textbooks	Many: leading mentions debugging on one page.	None

# Debugging != Programming

	Programming	Debugging
Core Models	Abstraction Programming paradigms Language Theory	Experimental method
Goals	Reliability, Security, Maintainability Usability	Localization Repair
Concerns	Algorithms Complexity Data Structures	Correctness
Textbooks	Many, many	None

# Programming → Debugging




*“While tools, languages, and environments have reduced the time spent on individual debugging tasks, they have not significantly reduced the total time spent debugging, nor the cost of doing so.”*

*“Therefore, a hyperfocus on elimination of bugs during development is counterproductive; programmers should instead embrace debugging as an exercise in problem solving.”*

[9]



# The debugging proces

- 
- Code understanding
  - Bug characterization & stabilization
  - Hypothesis formulation
    - especially with regards to the active dimension(s) of fault propagation
  - Fault localization
    - Usually the hardest part.

# Role of Code Understanding



- Code reading skills
- Experience recognizing idioms of design and implementation.
- Experts typically have better code comprehension skills and spend more time than novices on understanding code. [7]

# Nature and role of Bug characterization



- Stability
- Clues to propagation mechanism
- Data and resource sensitivity

# Fault Localization



- Initial infection is always immediately adjacent to fault.
- Localization operates in 3 dimensions
  - time, lexical distance, referential distance [4]
- Propagation can switch dimensions
- No guaranteed technique for concurrency issues.

Localization is hardest step.



# **EXPERTS DEBUG DIFFERENTLY THAN NOVICES**

# Experts Study Actively




- Like good students, generally they are active readers.
  - select what they do and do not read. “Don’t read source code until you have to.”
  - attend to their assumptions and insights,
  - ask *why* it was designed to work the way it does.

[Riedl et. al 1991]



# Experts read differently

- 
- Experts do “loose” reading and “close” reading.
    - loose is a form of scanning used to generate a rough sense of implementation and to gather cues about quality and

# Experts model the code




- Success in problem solving is largely a function of how one frames or represents the problem
  - Experts choose among a variety of representations during debugging.
  - Experts construct representations more accurately and quickly.
  - Experts shift between representations easily.

# Experts self-monitor




- Monitor their progress and comprehension more regularly
  - cycle from detailed reading and testing to overview
  - keep notes ... like a lab-notebook

# Experts test differently

- 
- Choose tests to disprove hypothesis.
  - Run cheap tests even if those tests only disprove weak hypothesis.
  - Attempt to predict the variety of possible test results.

# Expert localize differently

- 
- Don't single step until you must.
  - Isolate the relevant portion of the code
    - manual and mental slicing
    - more testing as part of characterization
    - no attempt to “correct” until fault is isolated.


# What are we doing about it?




- *“Debugging is not taught as a specific course in universities. Despite decades of literature suggesting such courses be taught, no strong models exist for teaching debugging.”*
  - [9]
- No textbooks, no endorsed curriculum, no practice exercises.



# Selected References

- 
- [1] J. F. Maranzano and S. R. Bourne, “A Tutorial Introduction to ADB.” 1977.
  - [2] B. W. Kernighan and P. J. Plauger, “The elements of programming style,” *The elements of programming style, by Kernighan, Brian W.; Plauger, PJ* New York: McGraw-Hill, c1978., 1978.
  - [3] E. Y. Shapiro, *Algorithmic Program DeBugging*. Cambridge, MA, USA: MIT Press, 1983.
  - [4] R. Ward, *Debugging* C. Que Corp., 1986.
  - [5] X. Zhang, “Fault location via precise dynamic slicing,” 2006.
  - [6] B. Liblit, *Cooperative bug isolation: winning thesis of the 2005 ACM doctoral dissertation competition*, vol. 4440. Springer, 2007.
  - [7] R. McCauley *et al.*, “Debugging: a review of the literature from an educational perspective,” *Computer Science Education*, vol. 18, no. 2, pp. 67–92, 2008.
  - [8] T. S. Kuhn, *The structure of scientific revolutions, 50th Anniversary Edition*, 4th ed. Chicago: University of Chicago Press, 2012.
  - [9] D. H. O'Dell, “The Debugging Mindset,” *Queue*, vol. 15, no. 1, p. 50, 2017.
  - [10] B. Beizer, “Software testing techniques. 1990,” *New York*, ISBN: 0-442-20672-0.
  - [11] J. Engblom, “Reverse History Part Three – Products – Observations from Uppsala.”.
  - [12] “Tutorial: Reverse debugging with GDB 7.” [Online]. Available: <http://jayconrod.com/posts/28/tutorial-reverse-debugging-with-gdb-7>. [Accessed: 28-Sep-2017].
  - [13] J. D. Sterman, “System dynamics modeling for project management,” *Unpublished manuscript, Cambridge, MA*, vol. 246, 1992.
  - [14] P. Bourque, R. E. Fairley, and others, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

# Selected References

- 
- [15] \_\_\_\_\_, “Debugging,” *Wikipedia*. 10-Oct-2017.
- [16] W. L. Johnson and E. Soloway, “PROUST: Knowledge-based program understanding,” *IEEE Transactions on Software Engineering*, no. 3, pp. 267–275, 1985.
- [17] D. E. Knuth, “The errors of TEX,” *Software: Practice and Experience*, vol. 19, no. 7, pp. 607–685, 1989.
- [18] IEEE Standard Classification for Software Anomalies. IEEE, 1994.
- [19] A. Robins, P. Haden, and S. Garner, “Problem distributions in a CS1 course,” in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, 2006, pp. 165–173.
- [20] J. F. Maranzano and S. R. Bourne, “A Tutorial Introduction to ADB.” 1977.
- [21] M. A. Linton, *The Evolution of Dbx*. 1990.
- [22] T.-D. B. Le, R. J. Oentaryo, and D. Lo, “Information retrieval and spectrum based bug localization: Better together,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 579–590.
- [23] H. Agrawal and J. R. Horgan, “Dynamic program slicing,” in *Acm Sigplan Notices*, 1990, vol. 25, pp. 246–256.
- [24] “Profiling (computer programming),” *Wikipedia*. 21-Sep-2017.
- [25] C. E. McDowell and D. P. Helmbold, “Debugging concurrent programs,” *ACM Computing Surveys (CSUR)*, vol. 21, no. 4, pp. 593–622, 1989.
- [26] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, “A systematic study of automated program repair: Fixing 55 out of 105 bugs for \$8 each,” in *Software Engineering (ICSE), 2012 34th International Conference on*, 2012, pp. 3–13.

# Selected References



- [27] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, “Genprog: A generic method for automatic software repair,” *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 54–72, 2012.
- [28] P. Machado, J. Campos, and R. Abreu, “MZoltar: Automatic debugging of android applications,” in *Proceedings of the 2013 International Workshop on Software Development Lifecycle for Mobile*, 2013, pp. 9–16.
- [30] R. Bednarik and M. Tukiainen, “Effects of display blurring on the behavior of novices and experts during program debugging,” in *CHI’05 Extended Abstracts on Human Factors in Computing Systems*, 2005, pp. 1204–1207.