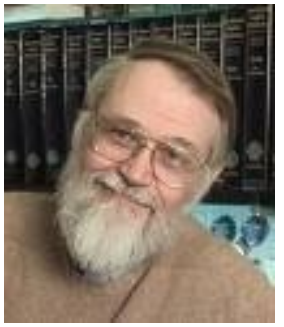
Topics for this Lecture

- Introduction to Debugging

Debugging

- Debugging is really hard – even with a good failing test case in hand
- One of the most time-consuming tasks in software development
- Locating the fault is the most time-consuming part of debugging
- Takes as much as 50% of development time on some projects
- Arguably the most scientific part of "computer science" practice
  - Even though it's usually done in a totally ad hoc way!

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." *- Brian Kernighan*



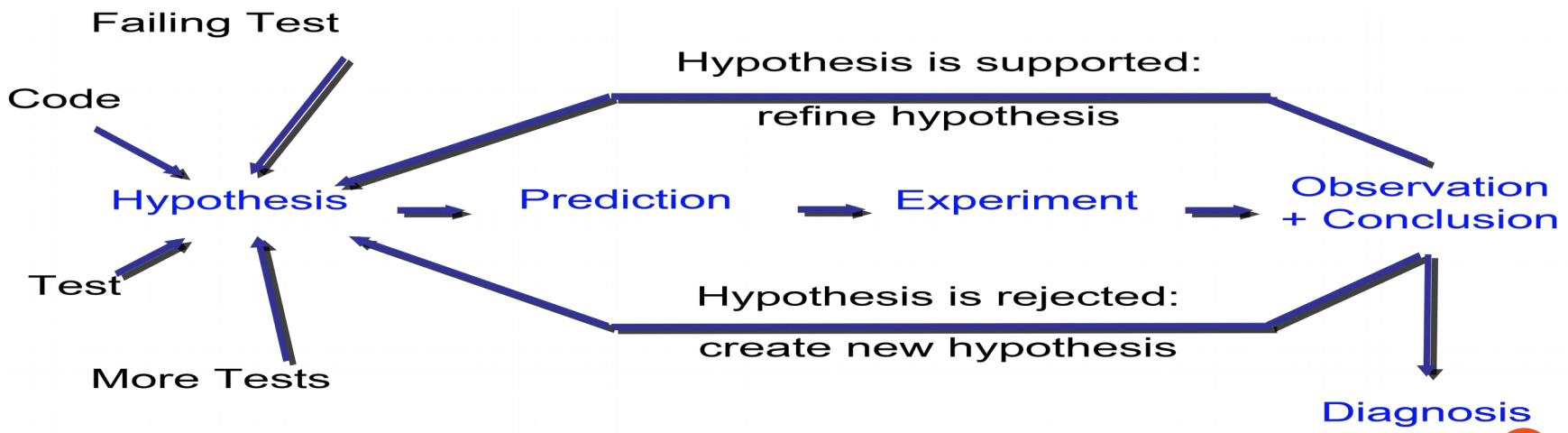
Oregon State University

Debugging and Testing

- **Testing** = detecting the presence of software faults
- **Debugging** = locating, diagnosing, and repairing faults
- What are test cases for, anyway?
  - Often: so we can locate and fix a fault
  - Or: so we can understand how serious the failure is
    - If we have many bugs, and some may not be important enough to merit resources
    - Or if there is a reason we can't change the code and have to work around the problem
- In either case, we have a "debugging" task at hand – must at least understand the failure, even if only to triage

Scientific Debugging

- Test cases (ones that fail and ones that succeed) can be the experiments we perform to verify our hypotheses
  - The failing test case informs us that there is a phenomenon to explain (apple on the head)
  - Generate (or examine) more test cases to find out more about what is going on in the program



Testing for Debugging

- Several ways to use test cases in debugging:
  - Test case minimization
    - Shrink the test case so we don't have to look at lots of irrelevant or redundant operations
  - Fault localization
    - Give suggestions about where the fault may be (based on test case executions)
  - Error explanation
    - Give a "story" of causality
      - (A causes B; B causes C; C causes failure)

attempt to automate part of scientific debugging

Popular Ways to Debug (how we debug)

- Using “printf”
  - Often mocked, viewed as unscientific
  - In fact, just an easy way to apply dynamic/log analysis and perform experiments
  - If you ask the right questions, print can be a great debugging tool
    - Supports scientific debugging
    - I have no lessons here, other than to print intelligently, not blindly grope around

Popular Ways to Debug

- Using a debugger
  - Usually thought of as “more scientific”
  - It can be!
  - A debugger is good when you want to:
    - Inspect closely what happens to some values
    - Slow down and carefully watch things
    - during one part of a run
    - Get information across a lot of state at once

- Hard to make guidelines, but in general printf is for “across time” and debuggers are for “across state”

Run-time Debugging Tools

- All modern deguggers ...
  - Allow inspection of program state
  - Pause execution
    - at selected points (breakpoints)
    - when certain conditions occur (watchpoints)
    - after a fixed number of execution steps
  - Provide display and control at the level of program source code
- Specialized debugging support may include
  - Visualization (e.g., for performance debugging)
  - Animation of data structures
  - Differential debugging compares a set of failing executions to other executions that do not fail

References:

http://classes.engr.oregonstate.edu/eecs/summer2015/cs362-002/Lecture15.pdf
http://classes.engr.oregonstate.edu/eecs/summer2015/cs362-002/Lecture16.pdf