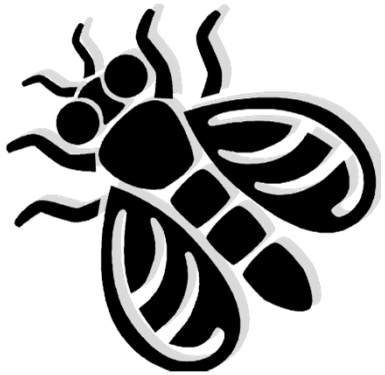
Basic Definitions: Testing

- What is software testing?
  - Running a program
  - In order to find faults
    - a.k.a. defects
    - a.k.a. errors
    - a.k.a. flaws
    - a.k.a. faults
    - a.k.a. **BUGS**

Bugs

Hopper's "bug" (moth stuck in a relay on an early machine)



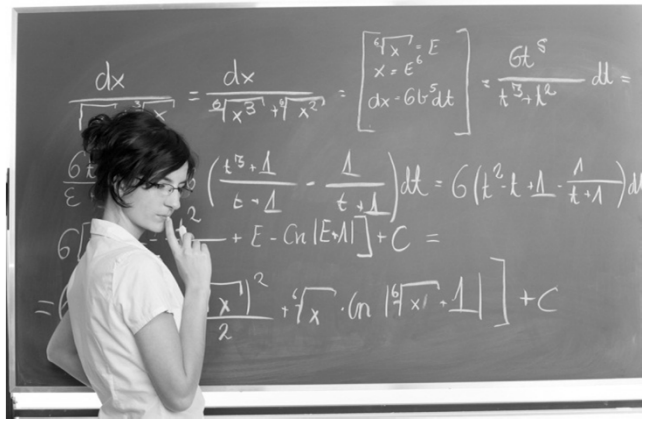
"an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders. " – Ada, Countess Lovelace (notes on Babbage's Analytical Engine)

"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and *[it is]* then that 'Bugs'—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite. . ." – Thomas Edison

Testing

- What *isn't* software testing?
  - Purely static analysis: examining a program's source code or binary in order to find bugs, but not executing the program
    - Good stuff, and very important, but it's not testing

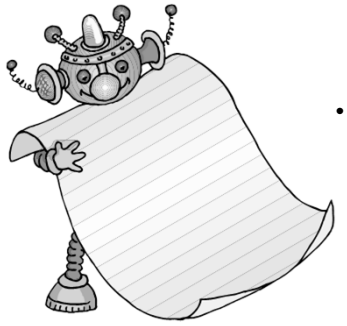
Why Testing?



- Ideally: we *prove* code correct, using formal mathematical techniques (with a computer, not chalk)
  - Extremely difficult: for some trivial programs (100 lines) and many small (5K lines) programs
  - Simply not practical to prove correctness in most cases – often not even for safety or mission critical code

Why Testing?

- Nearly ideally: use symbolic or abstract *model checking* to prove the system correct
  - Automatically extracts a mathematical abstraction from a system
  - Proves properties over all possible executions



- **In practice, can work well for very simple properties ("this program never crashes in this particular way"), but can't handle complex properties ("this is a working file system")**
- **Doesn't work well for programs with complex data structures (like a file system)**

As a last resort…

- … we can actually run the program, to see if it works

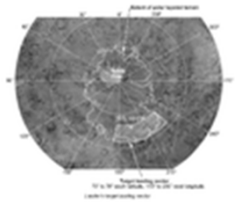
- This is software testing
  - Always necessary, even when you *can* prove correctness – because the proof is seldom directly tied to the actual code that runs

"Beware of bugs in the above code; I have only proved it correct, not tried it" – Knuth

Why Does Testing Matter?

- NIST report, “The Economic Impacts of Inadequate Infrastructure for Software Testing” (2002)
  - Inadequate software testing costs the US alone between $22 and $59 billion annually
  - Better approaches could cut this amount in half
- Major failures: Ariane 5 explosion, Mars Polar Lander, Intel’s Pentium FDIV bug
- Insufficient testing of safety-critical software can cost *lives*: THERAC-25 radiation machine: 3 dead
- We want **our** programs to be **reliable**
  - Testing is how, in most cases, we find out if they are

Ariane 5: exception-handling bug : forced self destruct on maiden flight (64-bit to 16-bit conversion: about 370 million $ lost)



Mars Polar Lander crash

THERAC-25 design

Why is Testing Hard?

- Because the only way to be SURE a program has no bugs is to run all possible executions

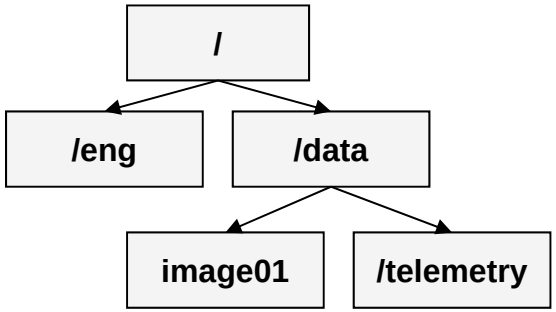
- We can’t do that

Example: File System Testing

- File system is a library, called by other components of the flight software
- Accepts a fixed set of operations that manipulate files:

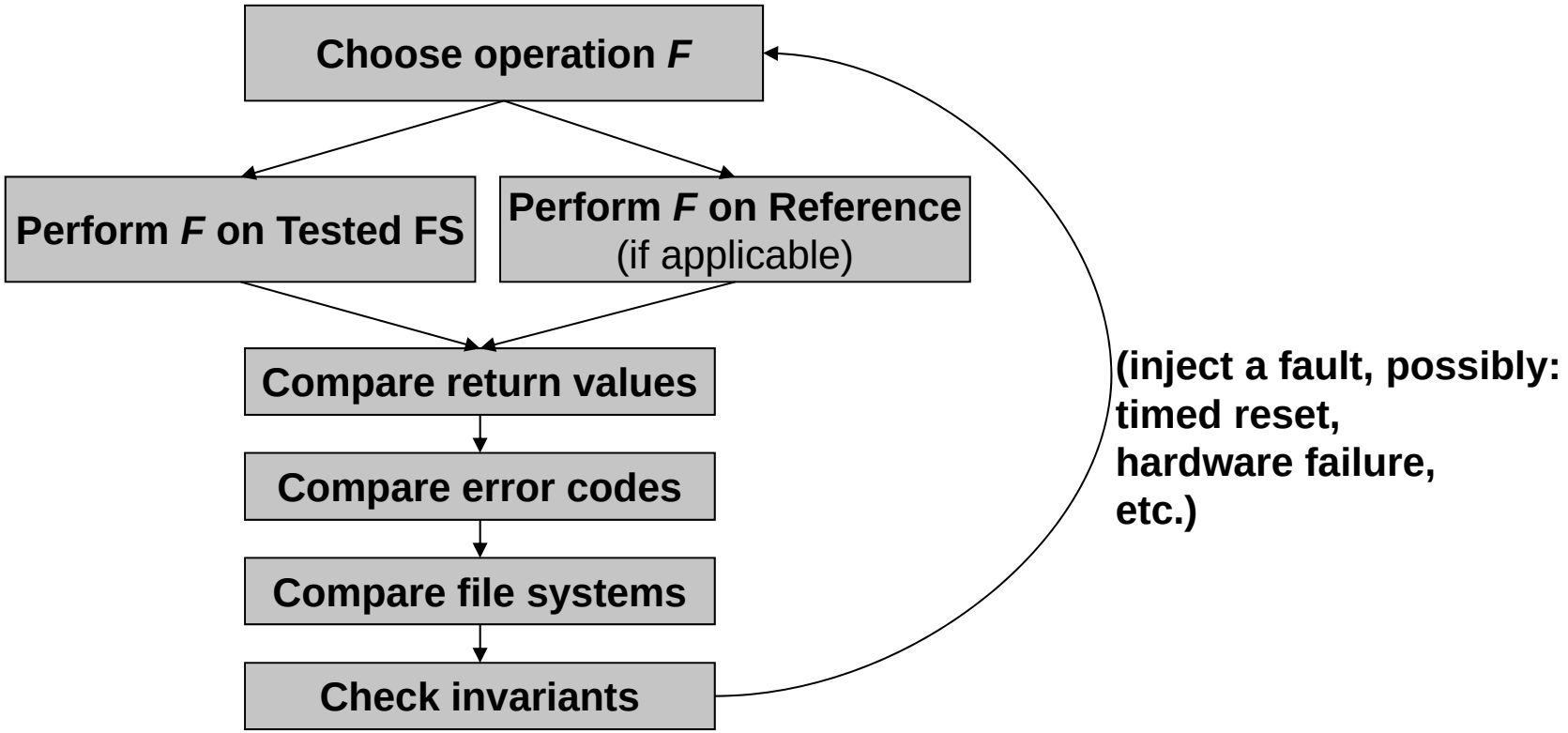
Operation	Result
mkdir (“/eng”, …)	SUCCESS
mkdir (“/data”, …)	SUCCESS
creat (“/data/image01”, …)	SUCCESS
creat (“/eng/fsw/code”, …)	ENOENT
mkdir (“/data/telemetry”, …)	SUCCESS
unlink (“/data/image01”)	SUCCESS

File system



Example: File System Testing

- Easy to detect many errors: we have access to many working file systems, and can just compare results



Example: File System Testing

- How hard would it be to just try “all” the possibilities?
- Consider only core 7 operations (`mkdir, rmdir, creat, open, close, read, write`)
  - Most of these take either a file name or a numeric argument, or both
  - Even for a “reasonable” (but not provably safe) limitation of the parameters, there are $\mathbf{266^{10}}$ executions of length 10 to try
  - Not a realistic possibility (unless we have $\mathbf{10^{12}}$ years to test)

The Testing Problem

- This is a primary topic of this class: what "questions" do we pose to the software, i.e.,
  - *How do we select a small set of executions out of a very large set of executions?*
  - Fundamental problem of software testing research and practice
  - An open (and essentially unsolvable, in the general case) problem

Faults, Errors, and Failures

- **Fault**: a static flaw in a program
  - What we usually think of as “a bug”
- **Error**: a bad program state that results from a fault
  - Not every fault always produces an error
- **Failure**: an observable incorrect behavior of a program as a result of an error
  - Not every error ever becomes visible

To Expose a Fault with a Test

- **Reachability**: the test much actually reach and execute the location of the **fault**
- **Infection**: the fault must actually corrupt the program state (produce an **error**)
- **Propagation**: the error must persist and cause an incorrect output – a **failure**

- **Which tests will achieve all three?**

What is Testing?

- What is software testing?
  - Running a program
  - In order to find **faults**
  - But also, in order to
    - Increase our confidence that the program has high quality and low risk
    - Because we can never be sure we caught all bugs
  - How does a set of executions increase confidence?
    - Sometimes, by algorithmic argument
    - Sometimes by less formal arguments