How to Code Better

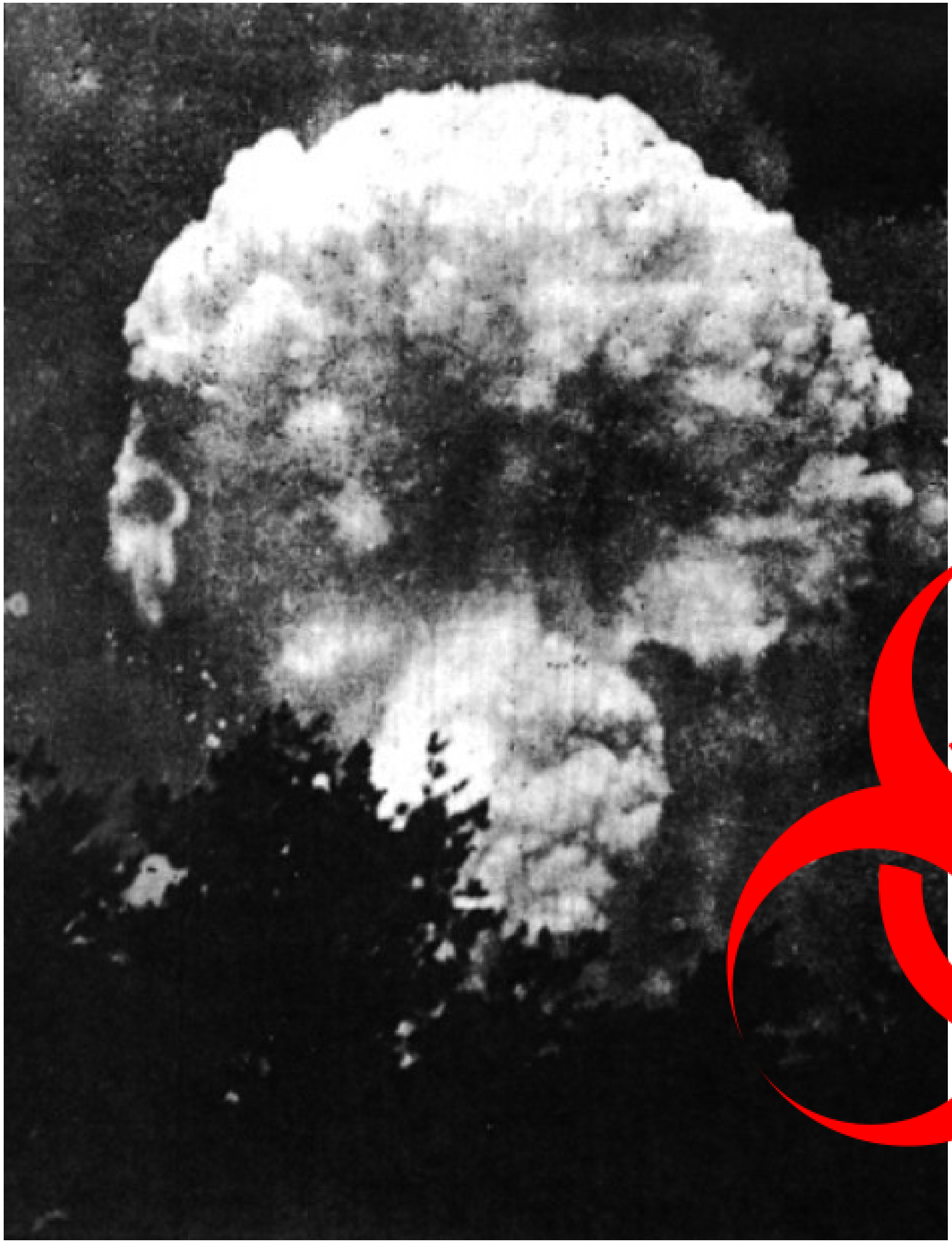
(especially with Eclipse)

Kaitlin ”Ducky” Sherwood, *ducky@webfoot.com*

You already know to use...

- Source code control (sort of built-in to Eclipse)
- Unit tests

Confirmation bias



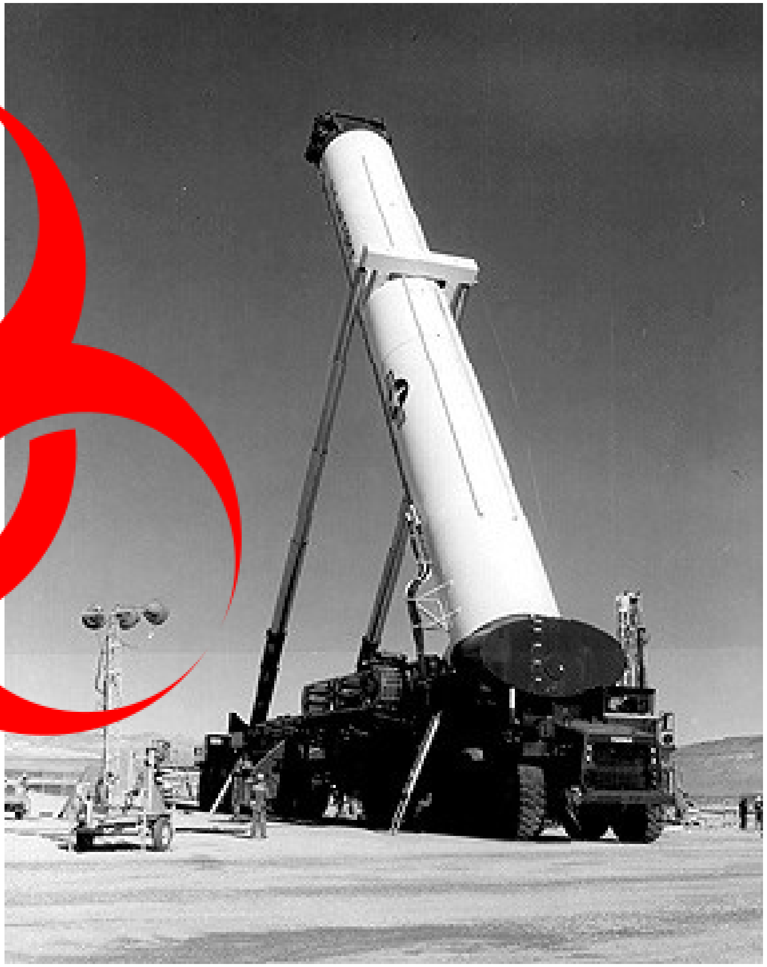
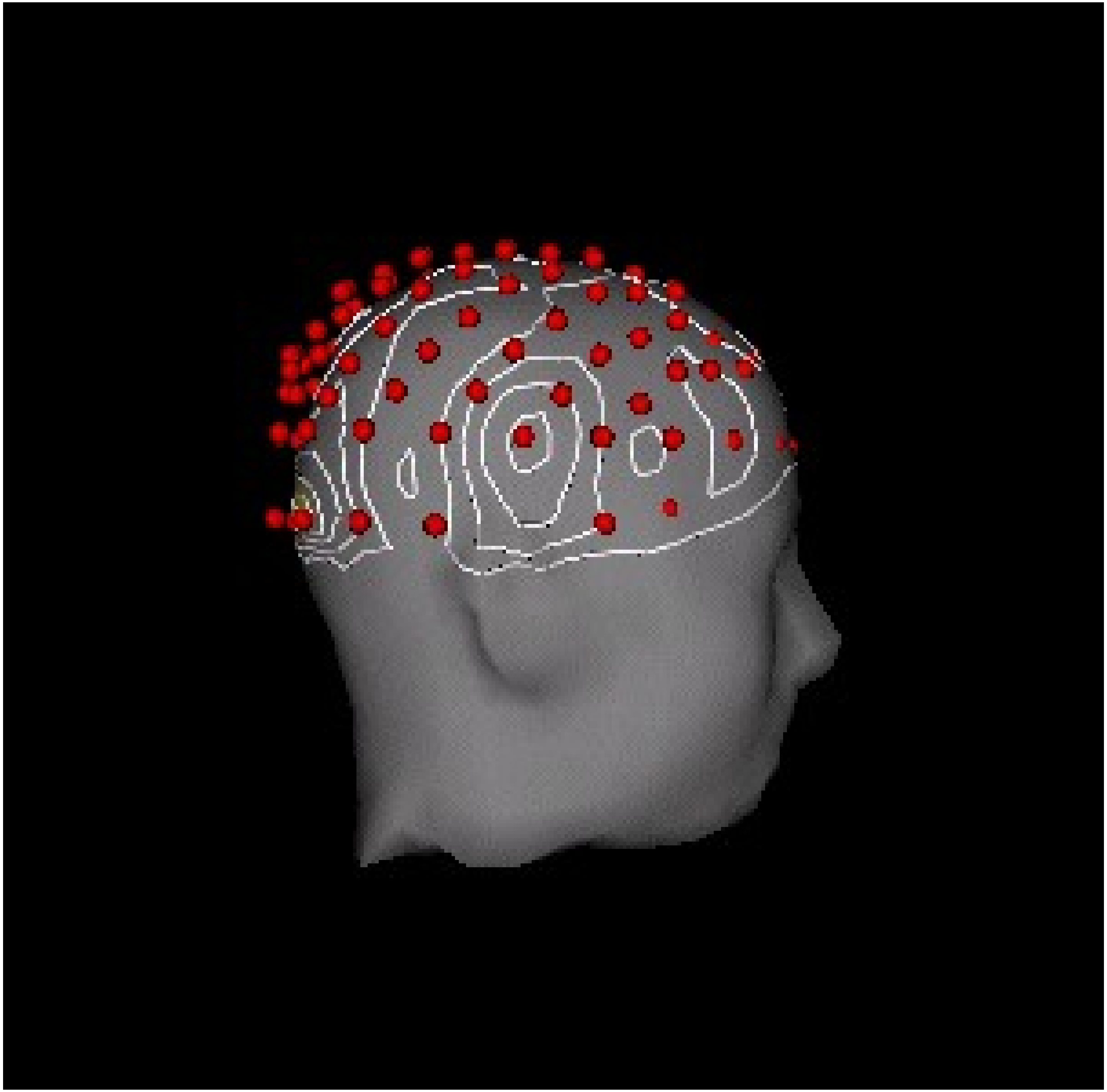


Image sources: NASA, Wikimedia



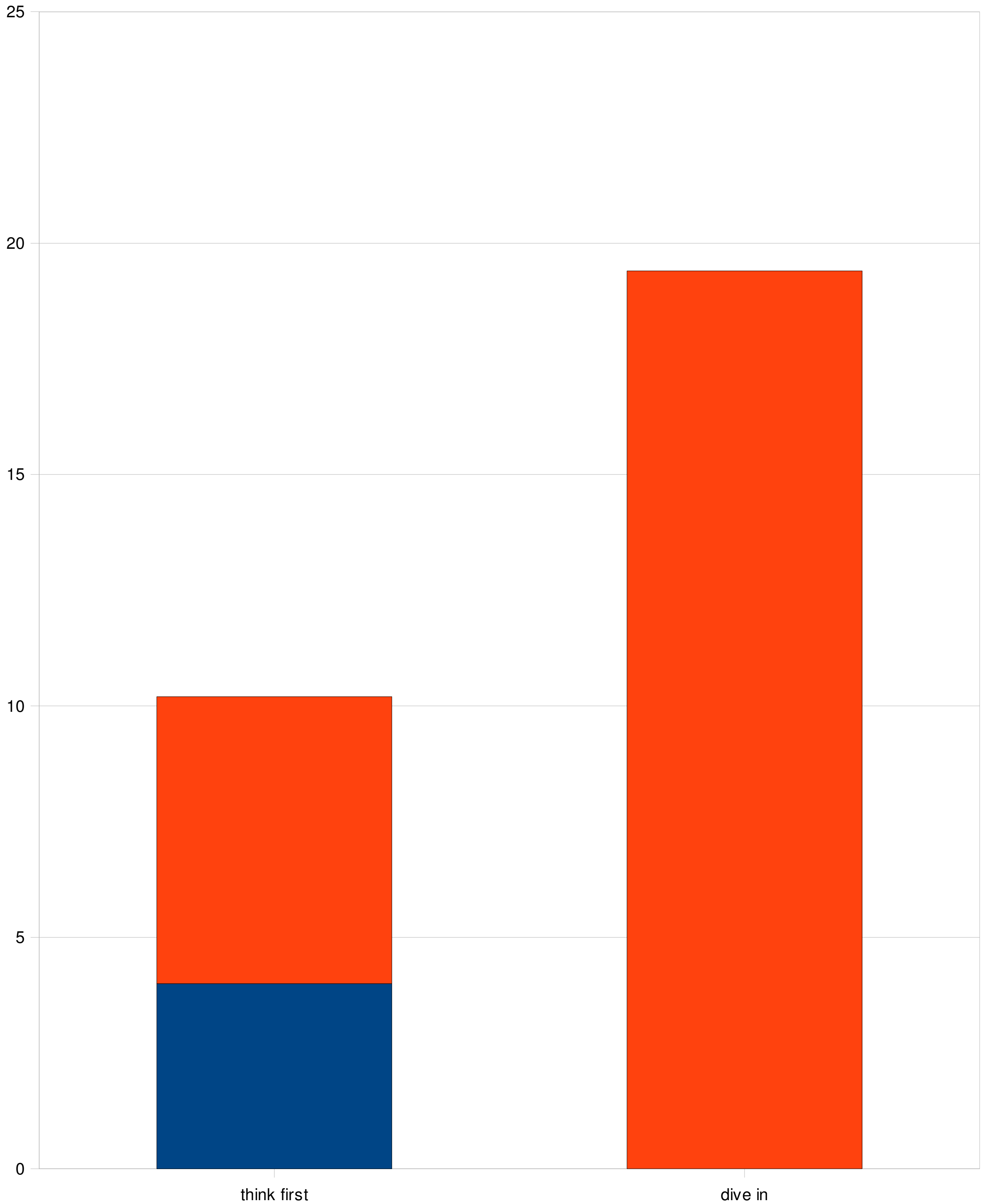
Insight
vs.
analysis

"Can't see the forest for the trees” is

causation,

not correlation!

Source: NOAA



🟥 Experimental time

🟦 Hypothesizing time(?)

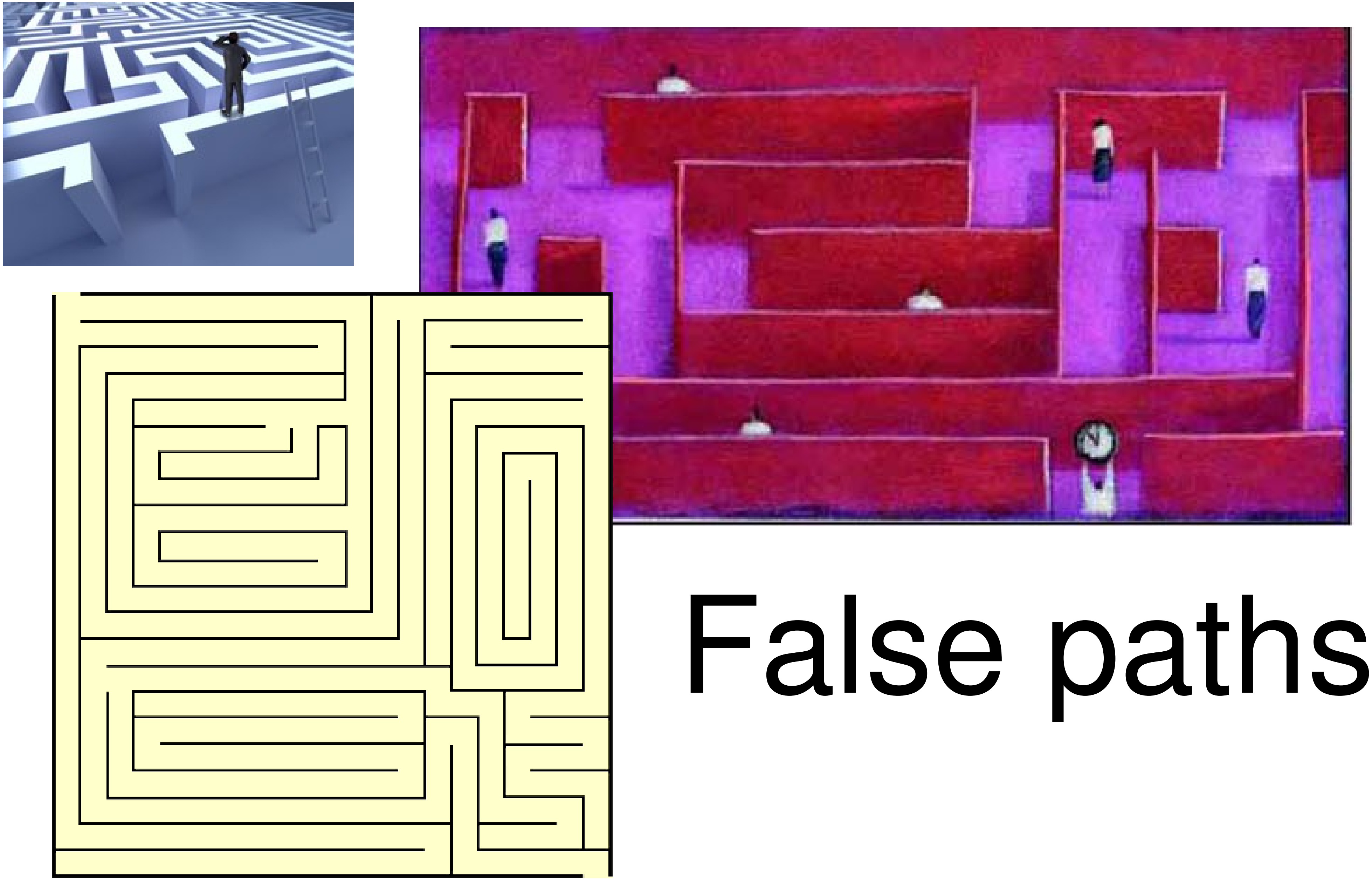
Image source: U.S. Consumer Products Safety Commission

So what does this mean?

- Think of multiple designs *before* you start coding.
- When you get stuck, stop and **write down** three hypotheses for where the bug is.
  - Learn to embrace being stuck (or rather, embrace *recognizing* when you are stuck)
  - Observe what hypotheses are correct.

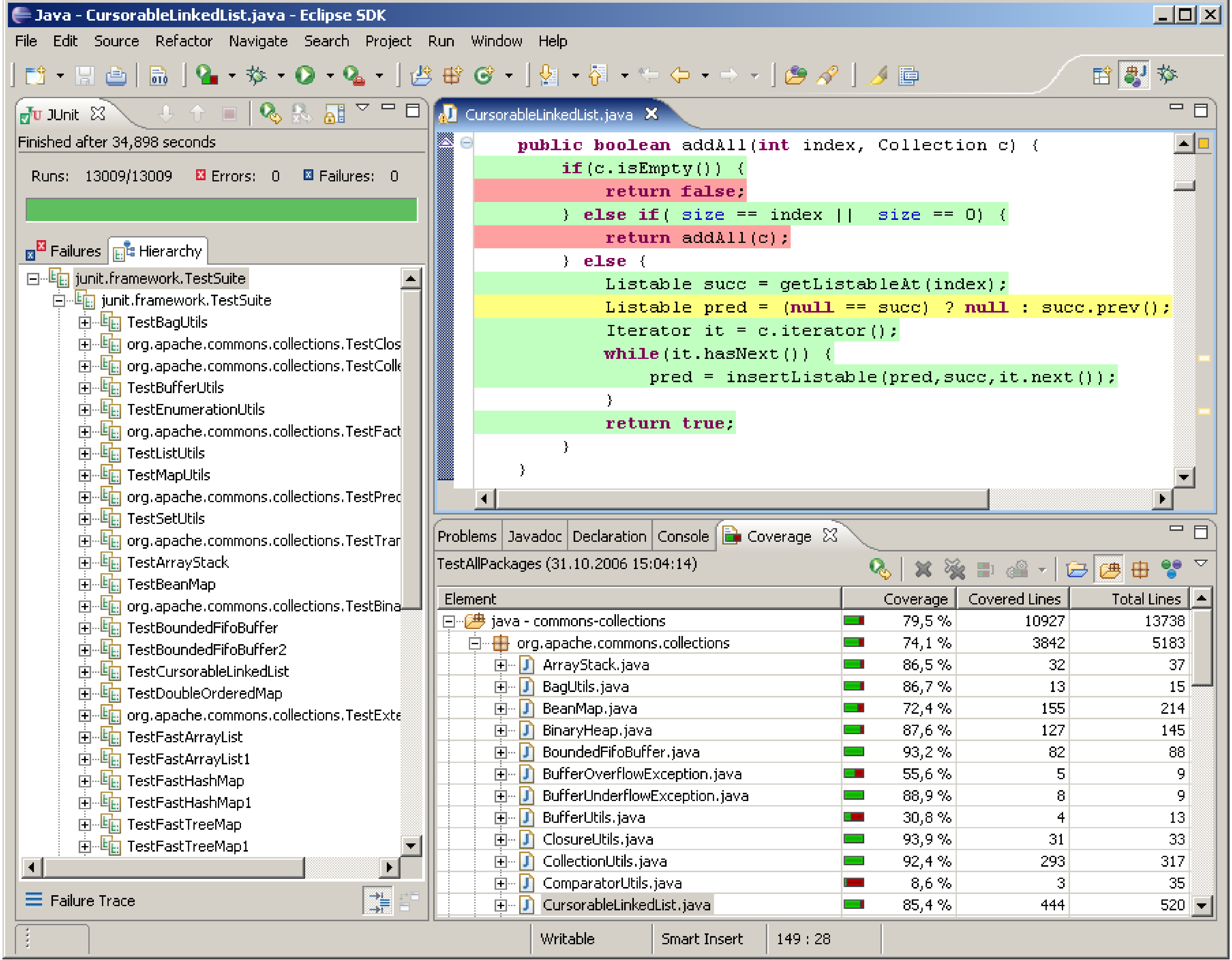
My common hypotheses:

- **Variable incorrectly passed** (passed foo1 instead of foo2)
- **Results incorrectly interpreted** (thought I was printing out bar1, in fact I was printing out bar2)
- **Variable set incorrectly** (foo * 2 instead of foo^2)
- **Variable *incorrectly not reset***



False paths

Image sources: State of Maine, FBI



Finished after 34,898 seconds

Runs: 13009/13009 Errors: 0 Failures: 0

- junit.framework.TestSuite
  - junit.framework.TestSuite
    - TestBagUtils
    - org.apache.commons.collections.TestClos
    - org.apache.commons.collections.TestColle
    - TestBufferUtils
    - TestEnumerationUtils
    - org.apache.commons.collections.TestFact
    - TestListUtils
    - TestMapUtils
    - org.apache.commons.collections.TestPred
    - TestSetUtils
    - org.apache.commons.collections.TestTrar
    - TestArrayStack
    - TestBeanMap
    - org.apache.commons.collections.TestBina
    - TestBoundedFifoBuffer
    - TestBoundedFifoBuffer2
    - TestCursorableLinkedList
    - TestDoubleOrderedMap
    - org.apache.commons.collections.TestExte
    - TestFastArrayList
    - TestFastArrayList1
    - TestFastHashMap
    - TestFastHashMap1
    - TestFastTreeMap
    - TestFastTreeMap1

```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if( size == index ||  size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,succ,it.next());
        }
        return true;
    }
}
```

TestAllPackages (31.10.2006 15:04:14)

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520

Writable | Smart Insert | 149 : 28

Differential Code Coverage

- Run your code once, making it show the error. Save the code coverage run.
- Run your code again, without hitting the error. Save the code coverage run.
- **Diff the two runs**. In *most* cases, this will show you the code where your error is (and not show you where it isn't).

Where might you use differential code coverage?

- GUI applications.
- Other People's Code.
- Very rare bugs.

Using the debugger

- Low observed use:
  - Boring and tedious
- When to use?
  - Step through (duh)
  - Binary search to narrow down bug location
  - Find where hangs are

Using the debugger to find hangs

- Run until it hangs.
- Pause the run.
- Put a breakpoint at all the current lines in all the current frames.
- Repeat until hang:
  - Run.
  - Remove breakpoint.
- Pause the run; the frame with the lowest breakpoint is where you want to start looking.

IDEs vs. vi/emacs

- vi/emacs allow you multiple views on same code more easily: just open another xterm!
- Eclipse (and other IDEs) force you into following one path at a time. (Remember: one hypothesis is bad!)

Noting different locations

- Bookmarks, only IFF using Mylyn (per-task)
- Open another pane (IDEs)
- Open an xterm on the location
- Write it down (paper works!)

Findbugs

Uses heuristics to find probable bugs. Many false-positives, alas. Java-only.

Eclipse tricks

- (Vi/emacs people, you can probably leave now)

URLs

- Findbugs: http://findbugs.sourceforge.net/
- EclEmma: http://www.eclemma.org/