Impossibility of Complete Testing

Cem Kaner
J.D., Ph.D.

November, 1997

Impossibility of Complete Testing

1. **This tutorial might give you ideas on how to explain the impossibility of complete testing to senior managers and lawyers.**

2. **"Complete coverage" is not complete testing. Coverage monitors are useful, but watch out for the snake oil salesmen.**

3. **The impossibility of complete testing is at the root of the "good enough testing" movement.**

The Impossibility of Complete Testing

If you test completely, then at the end of testing, there cannot be any undiscovered errors.

Complete testing is impossible for several reasons:

- **We can't test all the inputs to the program.**
- **We can't test all the combinations of inputs to the program.**
- **We can't test all the paths through the program.**
- **We can't test for all of the other potential failures, such as those caused by user interface design errors or incomplete requirements analyses.**

We Can't Test All The Inputs

The domain of possible inputs is too large. How can you test all:

- **Valid inputs**
  - » **Don't forget optimizations.**
- **Invalid inputs**
  - » **Don't forget Easter Eggs.**
- **Edited inputs**
  - » **These can be quite complex. How much editing is enough?**
- **Variations on input timing.**
  - » **In the client/server world, timing is essential.**

We Can't Test All The Combinations of Inputs

Variables interact.

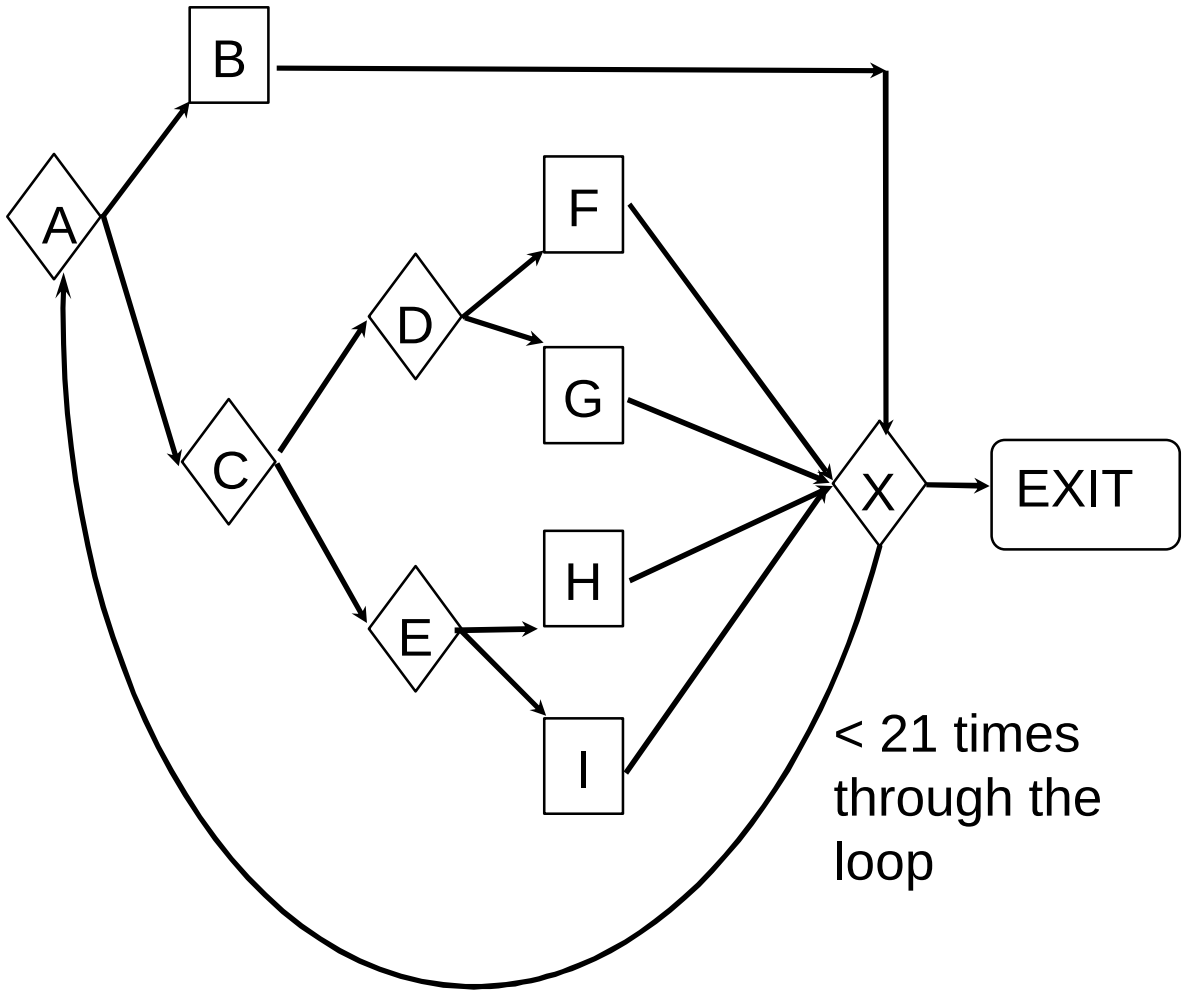
- **a program crashes on print preview of high resolution print image on a high resolution screen.**
- **a program fails when the sum of a series of variables is too large.**

Suppose a program lets you add two numbers. The first number can be between 1 and 100, the second between 1 and 25. The total number of pairs you can test is 100 x 25 (2500).

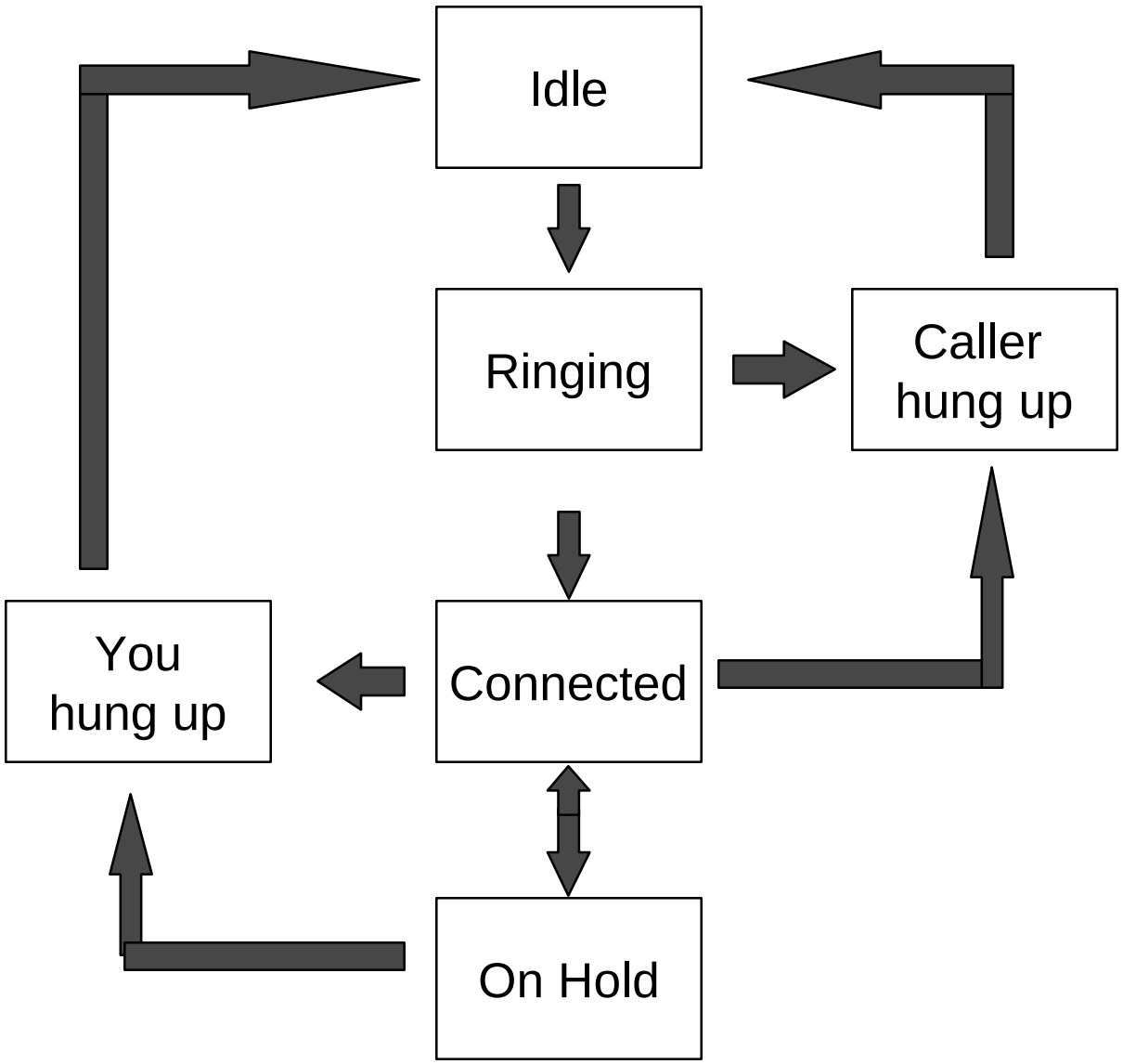
Suppose there are N variables. Suppose the number of choices for the variables are V_1, V_2, through V_N. The total number of possible combinations is $V_1 \times V_2 \times \cdots \times V_N$. This is huge.

We Can't Test All The Paths

There are $5^1 + 5^2 + ... + 5^{19} + 5^{20} = 10^{14} = 100$ trillion paths through the program to test. At one test per second, it would take longer than 3,000,000 years to run all these tests.

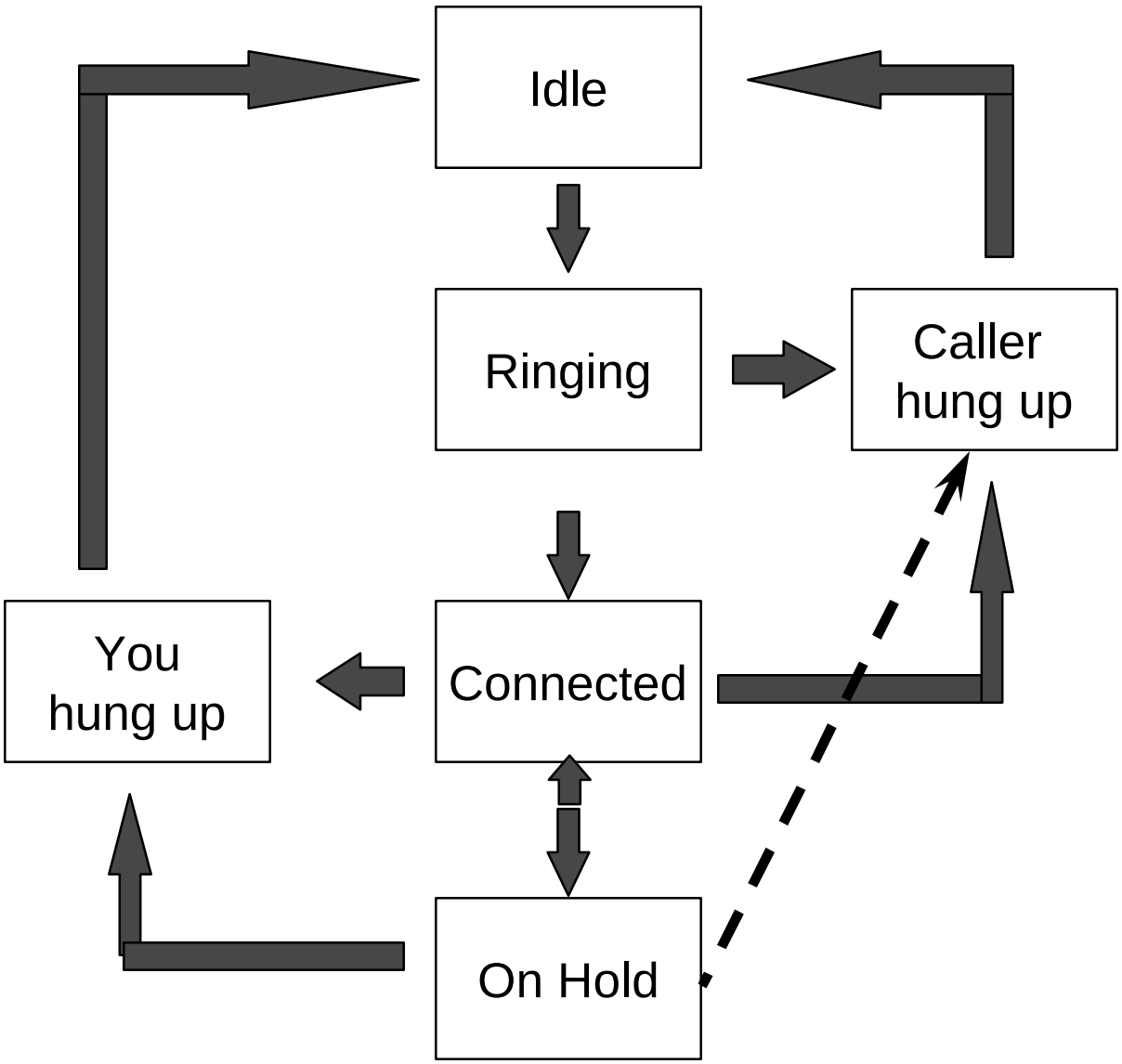


We Can't Test All The Paths



Refer to Testing Computer Software, pages 20-21

We Can't Test All The Paths



Refer to Testing Computer Software, pages 20-21

You Can't Test a Program Completely So What is Coverage?

***Coverage* measures the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors:**

- ***100% line coverage* means that you tested for every bug that can be revealed by simple execution of a line of code.**
- ***100% branch coverage* means you will find every error that can be revealed by testing each branch.**
- ***100% coverage* means that you tested for every possible error. This is obviously impossible.**

So what kind(s) and level(s) of coverage should we consider appropriate? There is no magic answer.

The Problem of Coverage

IEEE Unit Testing Standard is

100% Statement Coverage

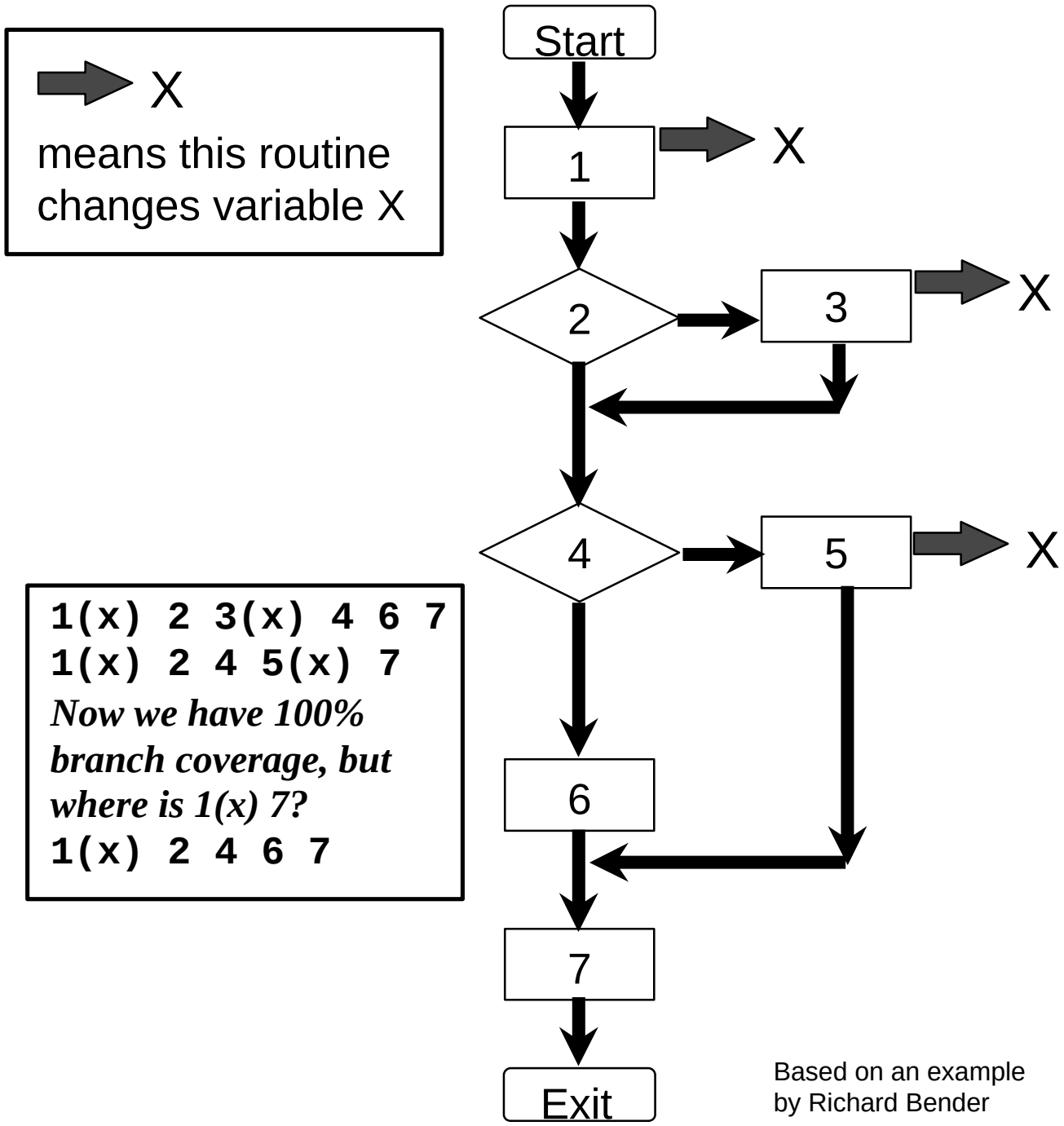
and 100% Branch Execution

(IEEE Std. 982.1-1988, § 4.17, "Minimal Unit Test Case Determination")

Most companies don't achieve this (though they might achieve 100% of the code they actually write.)

Several people seem to believe that complete statement and branch coverage means complete testing. (Or, at least, sufficient testing.)

Coverage Misses Data Flows



Line & Path Coverage Are Not Complete

All you're testing is the flowchart. You're not testing:

- **data flow**
- **tables that determine control flow in table-driven code**
- **side effects of interrupts, or interaction with background tasks**
- **special values, such as boundary cases.**
- **unexpected values (e.g. divide by zero)**
- **user interface errors**
- **timing-related bugs**
- **compliance with contracts, regulations, or other requirements**
- **configuration/compatibility failures**
- **volume, load, hardware faults**

More on Coverage

Line and branch coverage are easy to measure, but they not the only available measures of the extent of testing.

See my article, "Software Negligence and Testing Coverage" for a discussion of 101 different coverage measures.

Implications / Conclusions

- The core problem underlying testing is that we can run only a tiny sample of the set of possible tests.
- Test planning is really the development of a sampling strategy.
- Our sampling strategy is governed by heuristics, some of which are inappropriately put into standards.

Common Themes in Good Enough Testing

- Relentless shortcutting
- Forward scanning
- Ongoing extension of testing depth
- Heuristics for sampling
- Aversion to redundancy
- Risk-oriented discussions
- No obvious stopping rules

The first consequence of this fact is that any set of tests that your testers run will be only a tiny sample of the set that they could run. Testers have to come up with a powerful enough sample to find most of the defects. To create a good sample, testers generally apply some rules of thumb, like these:

- Cover every line of code with at least one test case. Some people call this "complete coverage" but this is not close to being an adequate test of the program.
- Include tests for every variable (input or output), that check whether the variable can handle extreme values and special cases. (If they can handle these, they can probably handle easier values too.)
- Include a test for every data flow or for every pair of variables that can possibly