CS301 Software Development

Instructor: Peter Kemper

Steve McConnell's Bag of Testing Tricks

Steve McConnell's Bag of Testing Tricks

- Incomplete testing
  - Not all inputs can be tested, but many are similar
  - Hence pick inputs that differ
- This idea shows up in various ways
  - Input partitioning
  - Picking representatives across equivalence classes
  - Error guessing

We'll focus on testing a single method.

Photo: flickr cc 70023venus2009
looking for a needle in a haystack

Structured Basis Testing (related: McCabe's cyclomatic complexity)

- Calculate lower bound for #paths
  1. Start with 1 for straight path through method.
  2. Add 1 for each of the following keywords, or their equivalents: if, while, repeat, for, and, and or.
  3. Add 1 for each case in a switch statement.
     - Default case counted with straight path.

- Create set of tests to cover calculated set of paths
  - Pay attention to same keywords as in 2. & 3.
  - Create 1 test per path

- Effect:
  - Each statement tested, some execution paths tested

Data Flow Testing

- Data can exist in one of 3 states:
  1. Defined: data initialized but not used yet
  2. Used: data has been used for computation
  3. Killed: data was once defined, but is undefined now

  + to describe enter-exit at method calls
  - Entered: control enters a method, initialized variable
  - Exited: control exits a method, assigned a return value

- Key idea:
  - Normal operation: defined -> used* -> killed
  - Other sequences (combinations) are suspicious works for code reviewing and testing

Suspicious Combinations

- Defined and then
  - defined: variable should not be set twice
  - exited or killed: variable is defined but not used
- Entered and then
  - used or killed: variable needs to be defined beforehand
- Killed and then
  - killed: variable should be not killed twice
  - used: logical error, accessing freed memory
- Used and then
  - defined: variable needs to be defined beforehand

Data Flow Testing

- Check suspicious combinations before testing
- Data flow testing
  - test all defined-used pairs
  - weaker variant: all definitions
- How to do this:
  - List all defined-used pairs for a method
  - For each def-use pair that is not covered yet:
    - Create additional test case(s)

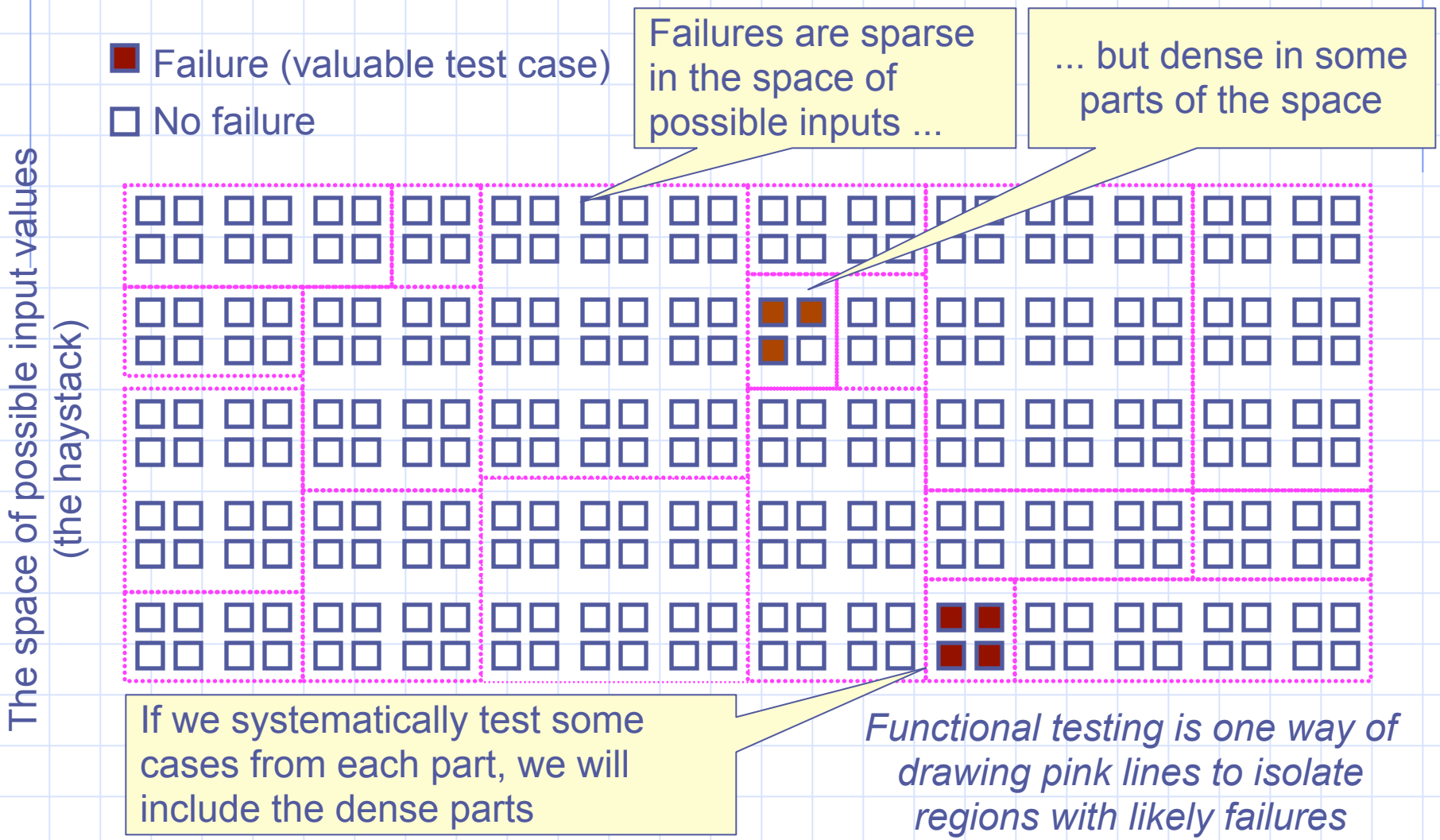
Systematic Partition Testing

- Equivalence Partitioning
  - Partition parameter/value ranges into groups that would have the same effect
  - For each group create only one test case with a representative value chosen from the equivalence set

- Particularly useful when looking at code from outside

Photo: flickr cc Perry Mc Kenna headless horse

Systematic Partition Testing



The Partition Principle

- Exploit some knowledge to choose samples that are more likely to include "special" or trouble-prone regions of the input space
  - Failures are sparse in the whole input space ...
  - ... but we may find regions in which they are dense

- (Quasi*-)Partition testing: separates the input space into classes whose union is the entire space

  *Quasi because: The classes may overlap

- Desirable case: Each fault leads to failures that are dense (easy to find) in some class of inputs
  - sampling each class in the quasi-partition selects at least one input that leads to a failure, revealing the fault
  - note: this is a heuristic not a law of nature

Steve McConnell's Bag of Tricks

- Error Guessing
  - Based on past experience, tester guesses what cases the implementation may not handle correctly
- Sources for sophisticated guesses
  - Use lists of common errors obtained from testing
  - Check common pitfalls known for certain data structures or algorithms
  - Check boundary values
  - Check corner cases and exception handling

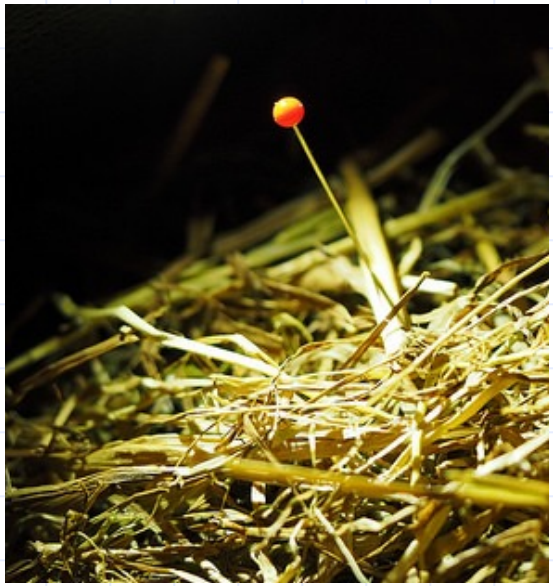
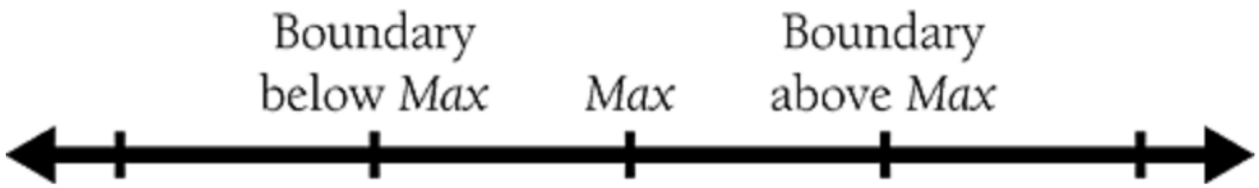


Photo: flickr cc marina noordegraf finding a needle

Boundary Analysis

- Check cases around limits and extreme cases



- Example 1: Check for off-by-one errors
- Check cases with max-1, max, and max+1
- Example 2: Combinations of extreme values
  - What if all factors in a product are large, negative, 0?
  - What if all strings pushed into a data structure are extremely long?

Steve McConnell's Bag of Tricks

- Classes of **Bad** Data

- Typical bad-data test cases include
  - Too little data (or no data)
  - Too much data
  - The wrong kind of data (invalid data)
  - The wrong size of data
  - Uninitialized data

Photo: flickr cc frits ahlefeldt-laurvig
Looking for a needle

Steve McConnell's Bag of Tricks

- Classes of **Good** Data
- Typical good-data test cases include
  - Nominal cases—middle-of-the-road, expected values
  - Minimum normal configuration
  - Maximum normal configuration
  - Compatibility with old data
- Structured basis testing typically covers good data cases

Steve McConnell's Bag of Tricks

- Use test cases that make hand checks convenient
  - Values like 10,000 are usually as good as 12,345
- Goes back to the Oracle problem
  - Given a test case, what is the correct answer?
  - How can you reliably get that answer?

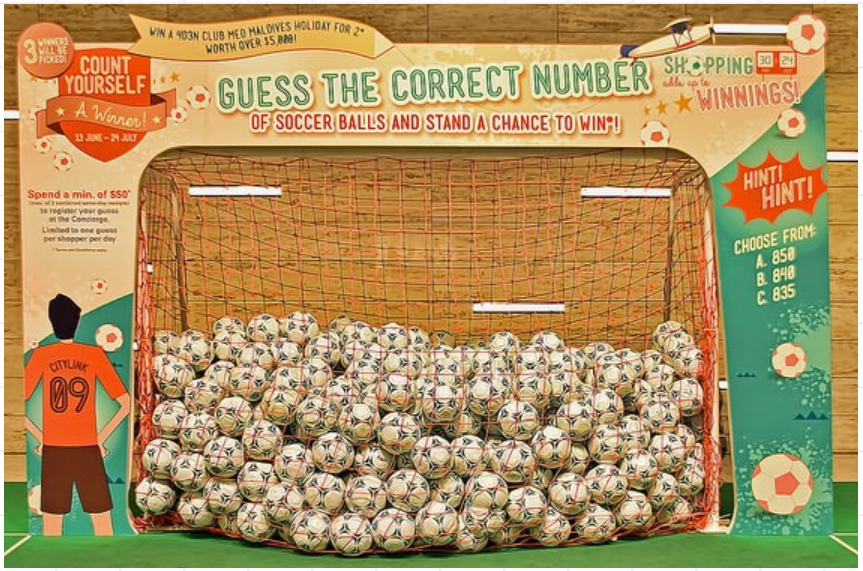


Photo: flickr cc frits ahlefeldt-laurvig
Looking for a needle

Final check list on test cases from Steve McConnell

1. Does each requirement that applies to the class or routine have its own test case?

2. Does each element from the design that applies to the class or routine have its own test case?

3. Has each line of code been tested with at least one test case? Has this been verified by computing the minimum number of tests necessary to exercise each line of code?

4. Have all defined-used data-flow paths been tested with at least one test case?

Final check list on test cases, continued

5. Has the code been checked for data-flow patterns that are unlikely to be correct, such as defined-defined, defined-exited, and defined-killed?

6. Has a list of common errors been used to write test cases to detect errors that have occurred frequently in the past?

7. Have all simple boundaries been tested: maximum, minimum, and off-by-one boundaries?

8. Have compound boundaries been tested—that is, combinations of input data that might result in a computed variable that's too small or too large?

Final check list on test cases, continued

9. Do test cases check for the wrong kind of data—for example, a negative number of employees in a payroll program?

10. Are representative, middle-of-the-road values tested?

11. Is the minimum normal configuration tested?

12. Is the maximum normal configuration tested?

13. Is compatibility with old data tested? And are old hardware, old versions of the operating system, and interfaces with old versions of other software tested?

14. Do the test cases make hand-checks easy?

Summary

- Structured basis testing
- Data flow testing
- Input partitioning
- Error guessing, the usual suspects:
  - Boundary analysis
  - Classes of good data
  - Classes of bad data
  - Complex methods
- Statistics:
  - Errors not distributed uniformly!

Photo: flickr, cc lee shaver
day216, feisty neighbors