Random Testing: Not Just for Unit Tests

Random Testing: How Good is It?

- Random testing is clearly sometimes both easy and effective
  - For what are essentially unit tests of a small piece of code
  - Can it work on larger pieces of software?

YES!

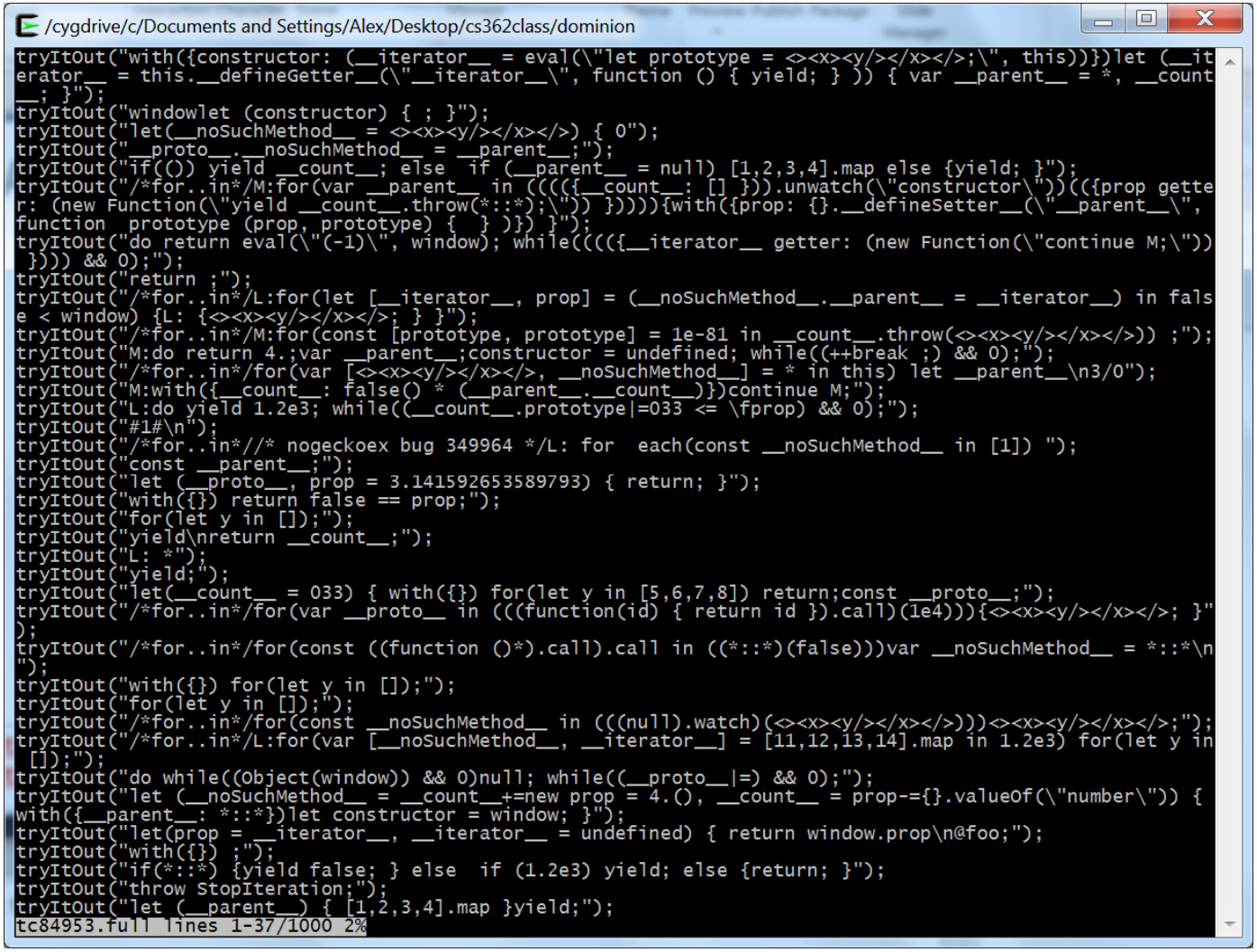
Random Testing in the Real World

- Random testing (usually called “fuzzing” in this context) is highly effective
  - Mozilla and Google use random testing to detect critical security flaws in JavaScript engines and browsers
    - Search for Google’s “ClusterFuzz”
  - Extremely effective for finding bugs in C compilers, including over 400 in GCC and LLVM
    - Search for the Csmith compiler testing project
  - Used to find bugs in Apache commons, Java core libraries, other widely used code
  - At NASA, random testing is used to test file systems for missions with costs well over $100M

Industrial Strength Random Testing

- No longer a quick afternoon's effort
  - Engineering random testers for compilers, interpreters, large libraries is a major undertaking
    - Requires testing expertise and domain expertise
  - Have to consider new issues such as:
    - How long should tests be?
    - How much to test each API call or function?
    - How to identify different bugs when an overnight testing run produces 5,000+ failing tests?
      - But there are only 20 different bugs
    - Can developers debug test cases consisting of thousands of random operations, mostly irrelevant?

A Very Bad Test Case




```
tryItOut("do;while(__parent__)__proto__=__parent__")
tryItOut("while(Error(this)){}")
```

Test Case Minimization

- Solution: automatic *minimization* of test cases as they are generated
- Minimized test case: subset of original sequence of operations such that
  - Test case still fails
  - Removing any one operation makes the test case successful
- Typical improvement: order of magnitude or greater reduction in length of a test case
  - Highly effective technique, essential for quick debugging

Test Case Minimization

- Based on Zeller's d*elta-debugging* tools
  - Automated debugging state-of-the-art
  - Set of Python scripts easily modified to automatically minimize tests in different settings
  - Requires that you be able to
    - Play back test cases and determine success or failure automatically
    - Define the subsets of a test case – provide a test case decomposition

REMEMBER DELTA DEBUGGING: IT CAN SAVE YOU ENDLESS HOURS OF EFFORT

Test Case Minimization

- Based on a clever modification of a “binary search” strategy