Regression Testing

Lecture Objectives:

1) Define regression testing
2) Explain the advantages and disadvantages of automated testing.
3) Explain the concept of bug clustering.
4) Explain reasons why regression test suites tend to grow over time.

Final Exam is
(I believe)
Monday
Here

11:00 - 13:00

Introduction

- We've talked about testing all quarter, but what makes for good tests?

1. Reliable
2. Reusable
3. Automated (to a point)
4. Cost effective
5. Quick
6. Thorough

What are your thoughts on good tests?

White Box Versus Black Box

Which is better?

- **White box best for ensuring details are correct in implementation**
  - Tests are designed to ensure that code conforms to design
  - Exercises different nooks and crannies of code in a way black box usually can't
  - Best way to ensure good coverage metrics
- **Black box best for requirements-based testing**
  - Emphasis on meeting requirements and, in general, overall behavior
  - Tests are designed to ensure that the software actually works!

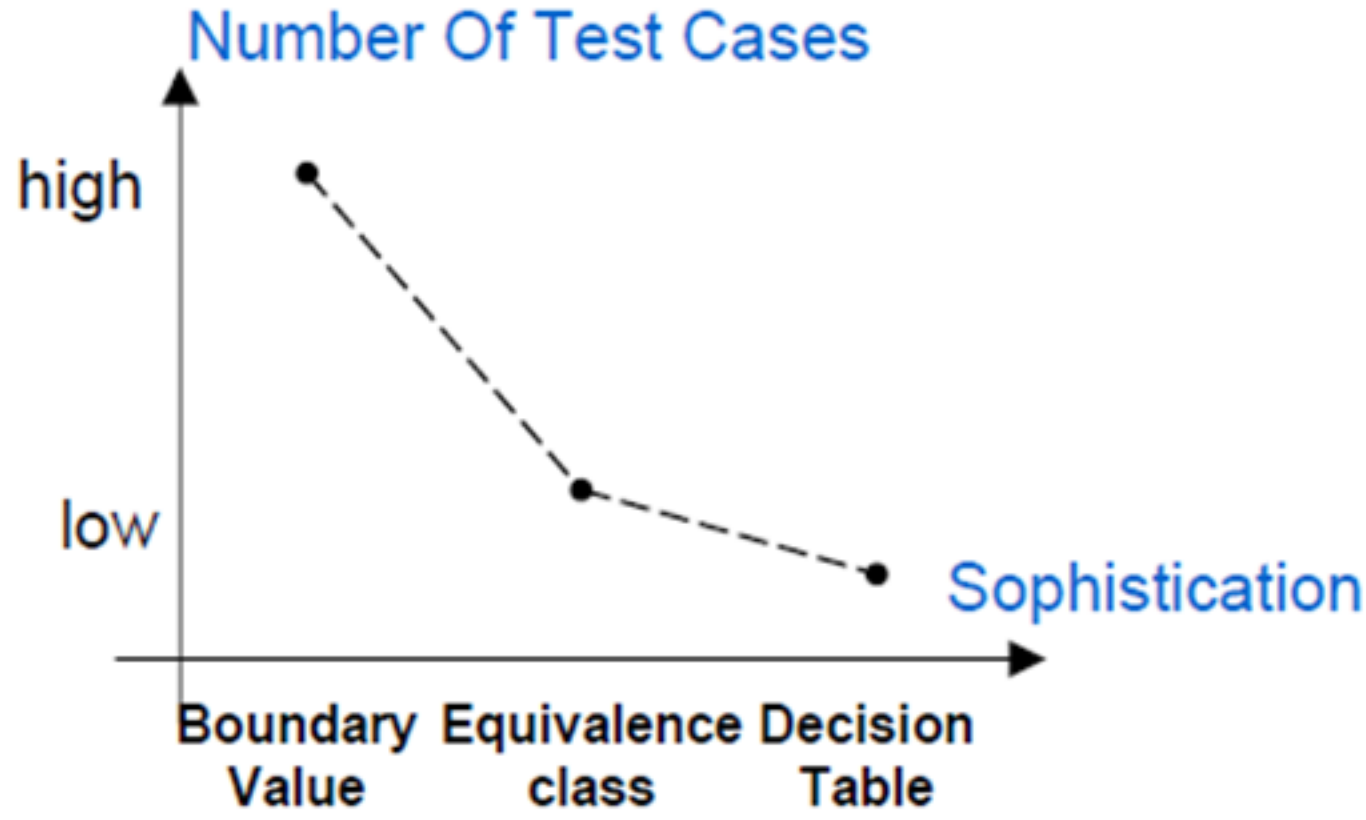
White Box Versus Black Box

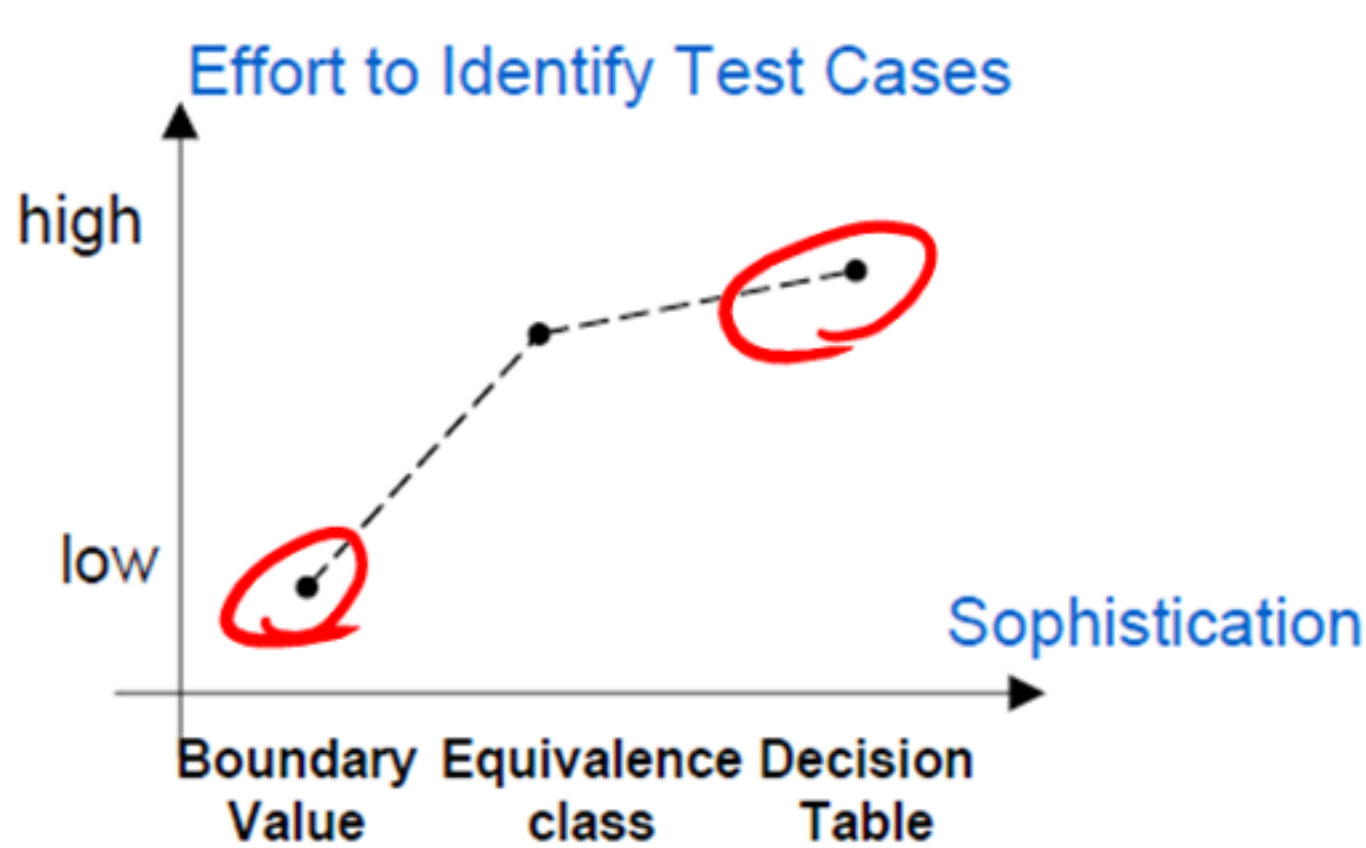
Which is better?

- **White box best for ensuring details are correct in implementation**
  - Tests are designed to ensure that code conforms to design
  - Exercises different nooks and crannies of code in a way black box usually can't
  - Best way to ensure good coverage metrics
- **Black box best for requirements-based testing**
  - Emphasis on meeting requirements and, in general, overall behavior
  - Tests are designed to ensure that the software actually works!

Both types of test have their place (of course!)

Test Approaches and Effort





Automatic

~~thought~~ thorough

Repeatable

Independent

Professional

A-TRIP

- We want our tests to run automatically
  - From an execution standpoint
  - From a results comparison standpoint

- Manual execution is tedious!

Automatic

JUnit / CPPUnit / TestNG

How do we automate?

- Junit is a good start!
- Mock objects are a great aid!
- But,
  - Automate DB connections (if you have them)
  - Automate network connections (if you have them)
- Automate Unit tests in your development process
  - JUnit can be run out of Ant!

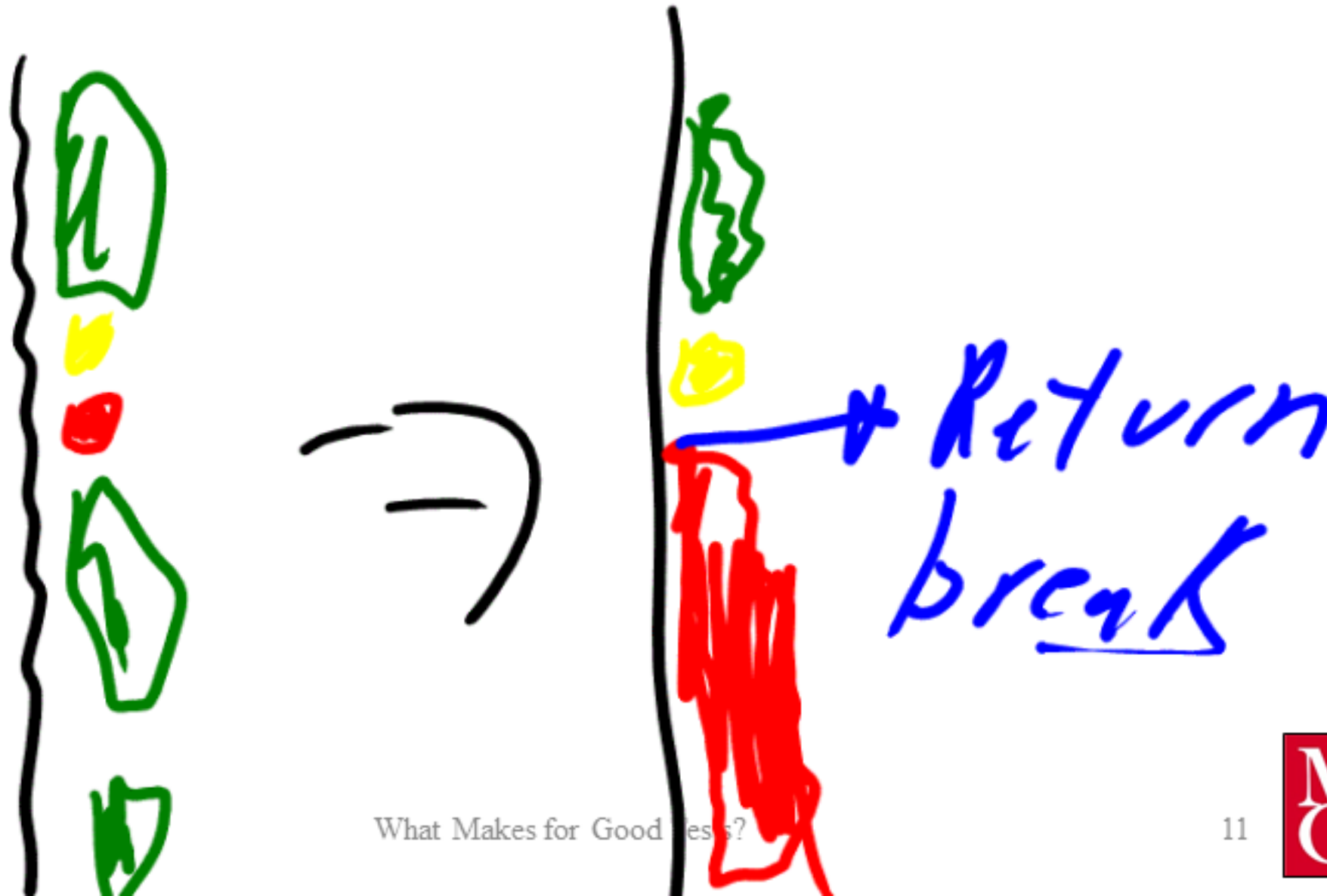
Thorough

- Test every thing that is likely to break!
  - Every line of coded, every branch, every exception?
  - Boundary conditions, invalid data inputs, etc.
  - The scope depends on the criticality of your software

Right!

- Determining which code you are testing helps to assess the thoroughness of your tests
  - Can be measured with a code coverage tool (nounit, quilt, EMMA, etc.)

Code Coverage



- Bugs tend to cluster ()
  - In methods
  - In classes
  - In packages

Bug clumping

=) problem one can cause another

- You thoughts? — C/C++

Why do bugs cluster?

```
if(x = y)
if( x = z)
⋮
if( x = q)
```

Nasa Spaceshuttle
Space station

new . . . .

new . . . .

new . . . .

new . . . .

Never-deleted
Allocated
memory

new . . . .

new . . .

new . . . .

C++ code

Bug clumping

- Bugs tend to cluster ()
  - In methods
  - In classes
  - In packages

- If you have a buggy module, it may be worthwhile to replace it or re-write it.

Repeatable

- It must be possible to reproduce the same results exactly if a test is run again
  - Environment must be controlled
  - Development "sandbox" environment
- If a bug is random in its occurrence, it probably is a problem with your testing environment, NOT the code under test

- Only test one thing at a time
- Write tests so that they can execute in any sequence
  - Don't rely on other tests to set up pre-conditions or tear-down after a test

Independent

JUnit

Professional

- Write your tests with the same quality as production code
- Review your tests in the same manner you would review source code
- Your test code will be at least as long (if no longer) than the production code you are testing

- Create meaningful tests

Test what you need to test not necessarily everything

Professional

Accessor getPI()

{ return 3.14; }

void setName(String name)
{this.name = name;}

Regression Testing

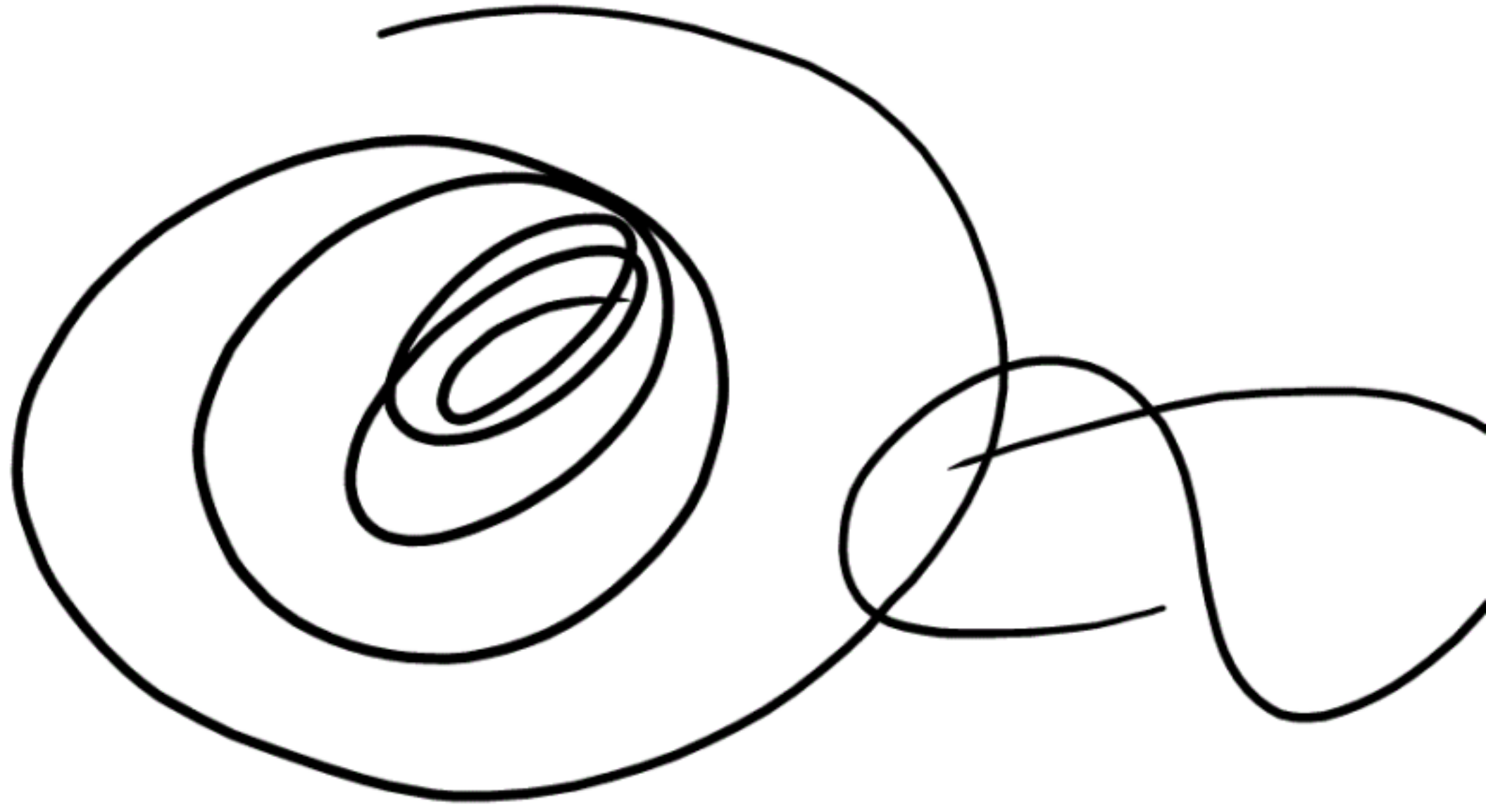
Serves to verify to not find bugs have been injected back into the system.

"Badness ometer"

⇒ Every evening or periodically

- Do we need to test the tests with more tests?

Testing the tests



Testing the tests

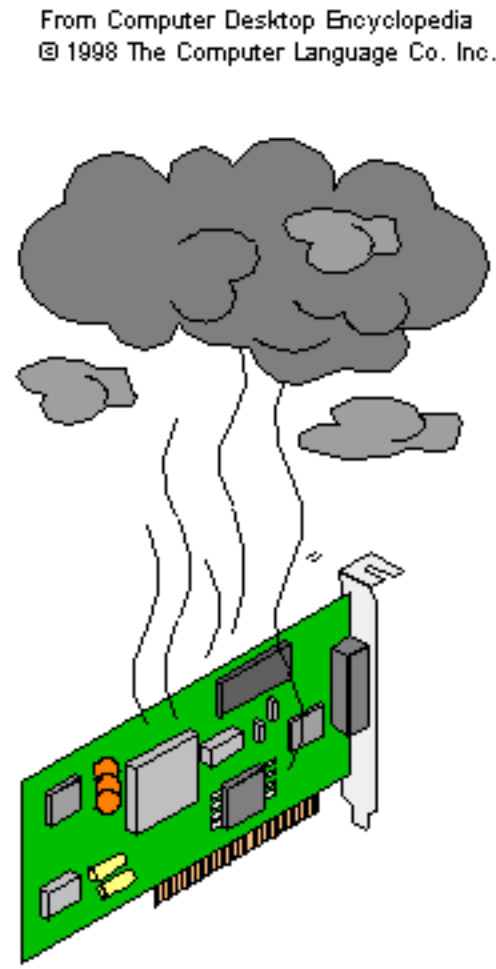
- Bug fixes
  - Identify the bug
  - Write a unit test case that fails to prove the bugs existence
  - Fix the code that is broke
  - Verify that nothing else broke

Testing the tests

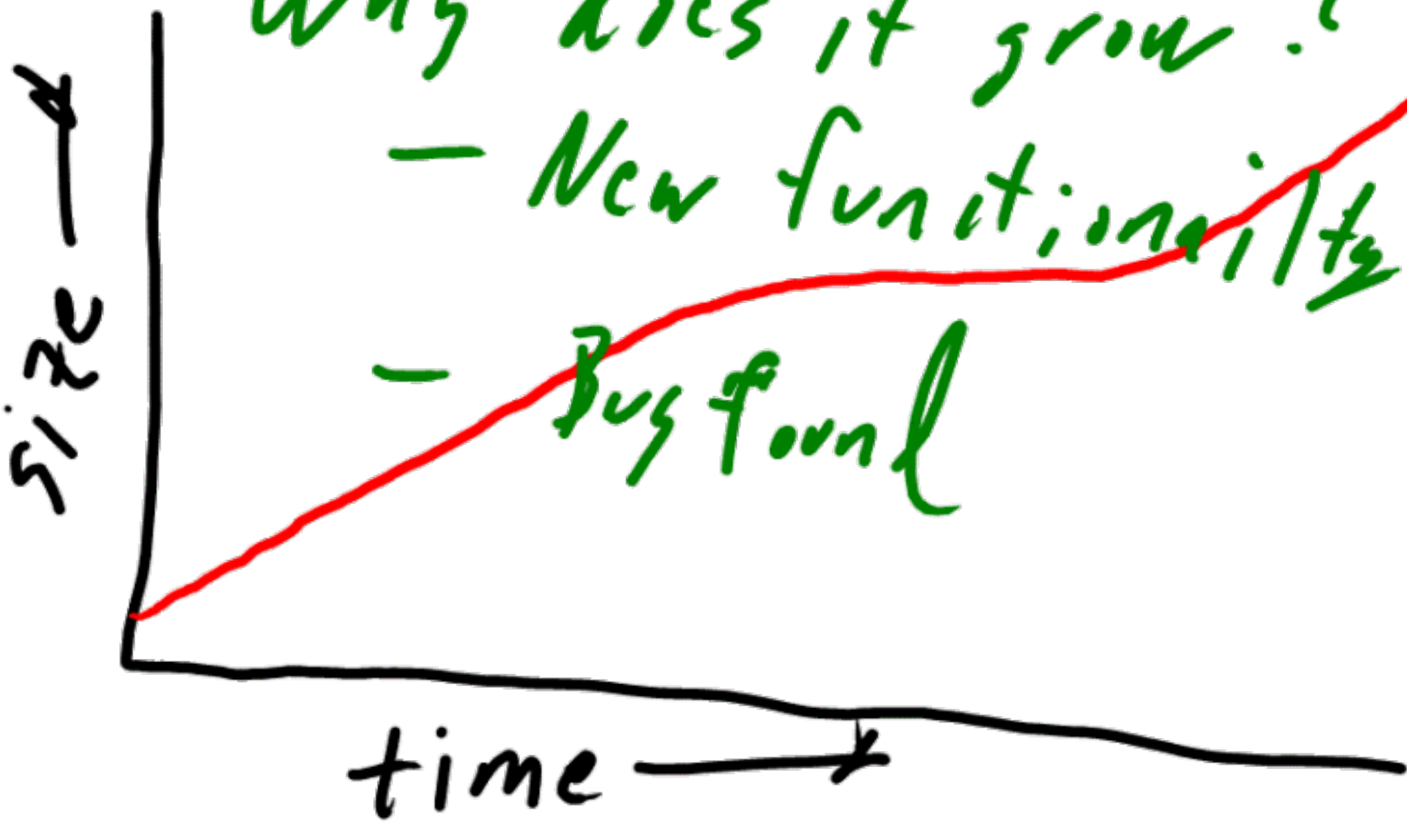
- Fault injection
  - Occasionally force a bug into your code to make sure the tests catch it
  - Use a mutation tool
    - Jester
    - MuClipse
    - Others

Daily "smoke" tests

- Smoke test
  - Borrowed from hardware testing
    - A relatively simple check to see whether the product "smokes"
  - Check basic functionality of software
    - Not exhaustive
- Daily/nightly build
  - Software is compiled, linked and (re)tested on a daily basis
  - "Good" build if pass all smoke tests



What happens to the size of our test suite over time?



Continuous Integration

- Pioneered by Martin Fowler; part of Extreme Programming
- Ten principles:
  - maintain a single source repository
  - automate the build
  - make your build self-testing
  - everyone commits to mainline every day
  - every commit should build mainline on an integration machine
  - keep the build fast
  - test in a clone of the production environment
  - make it easy for anyone to get the latest executable
  - everyone can see what's happening
  - automate deployment

Run unit tests

Routinely

⇒ Build code on commit

SDL Topic

SE 3800 in winter