CSC 7426 : Basics of Software Engineering

J Paul Gibson, D311

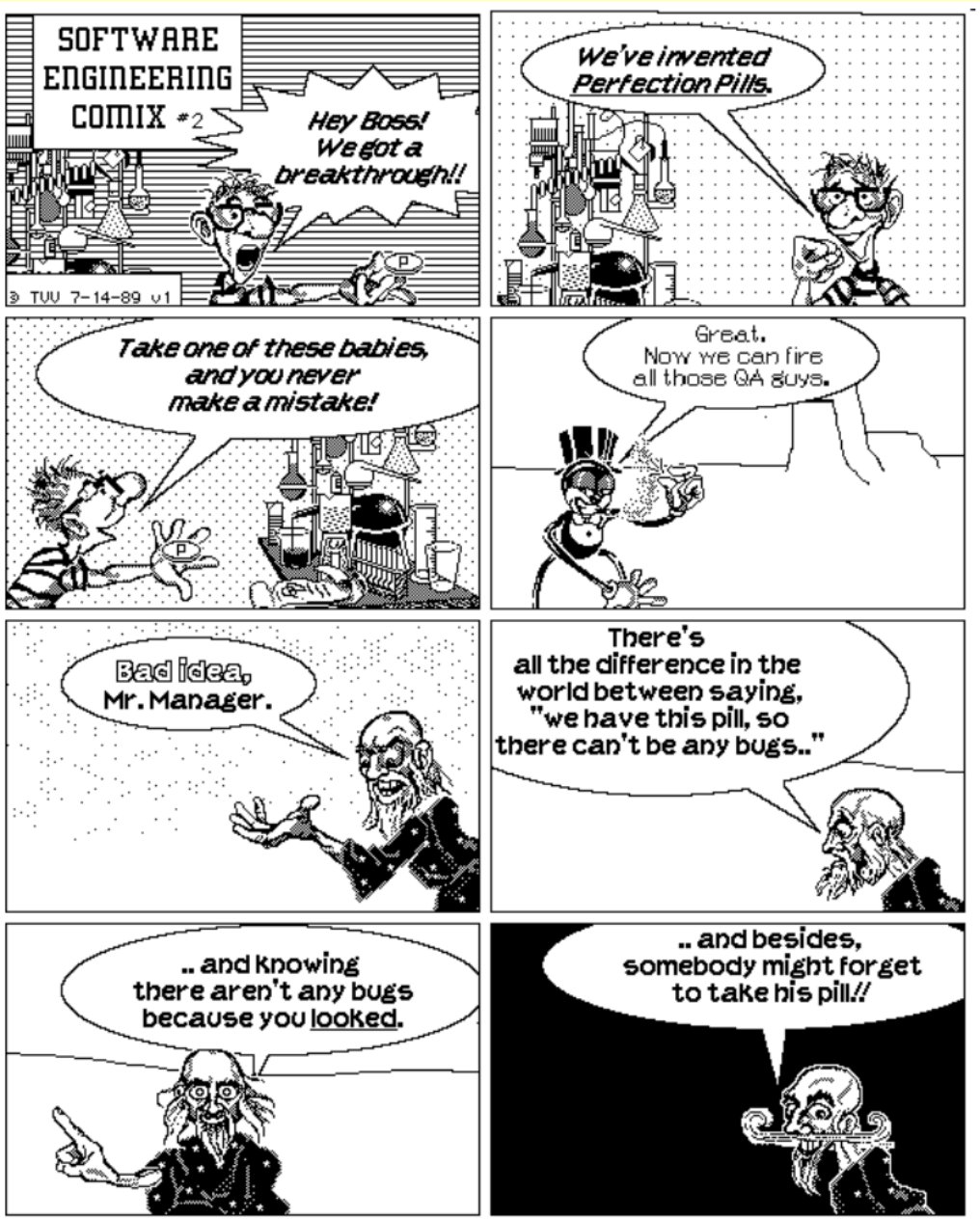
paul.gibson@telecom-sudparis.eu

http://jpaulgibson.synology.me/Teaching/TSP/CSC7426/

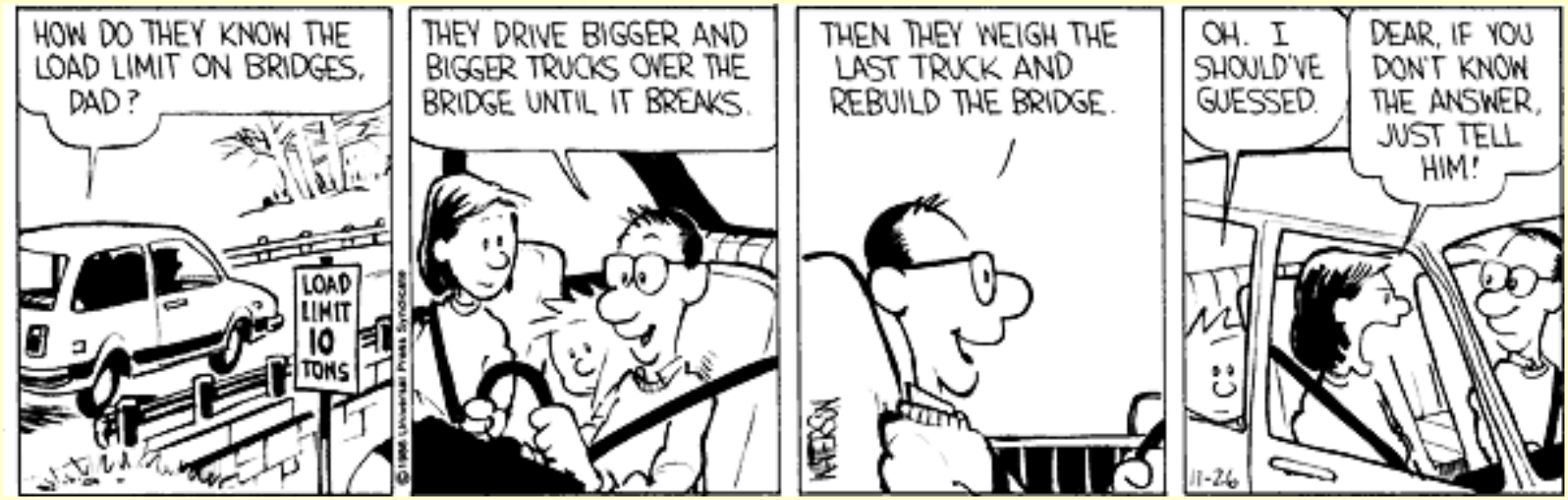
Testing

L6-TestingProblems.pdf

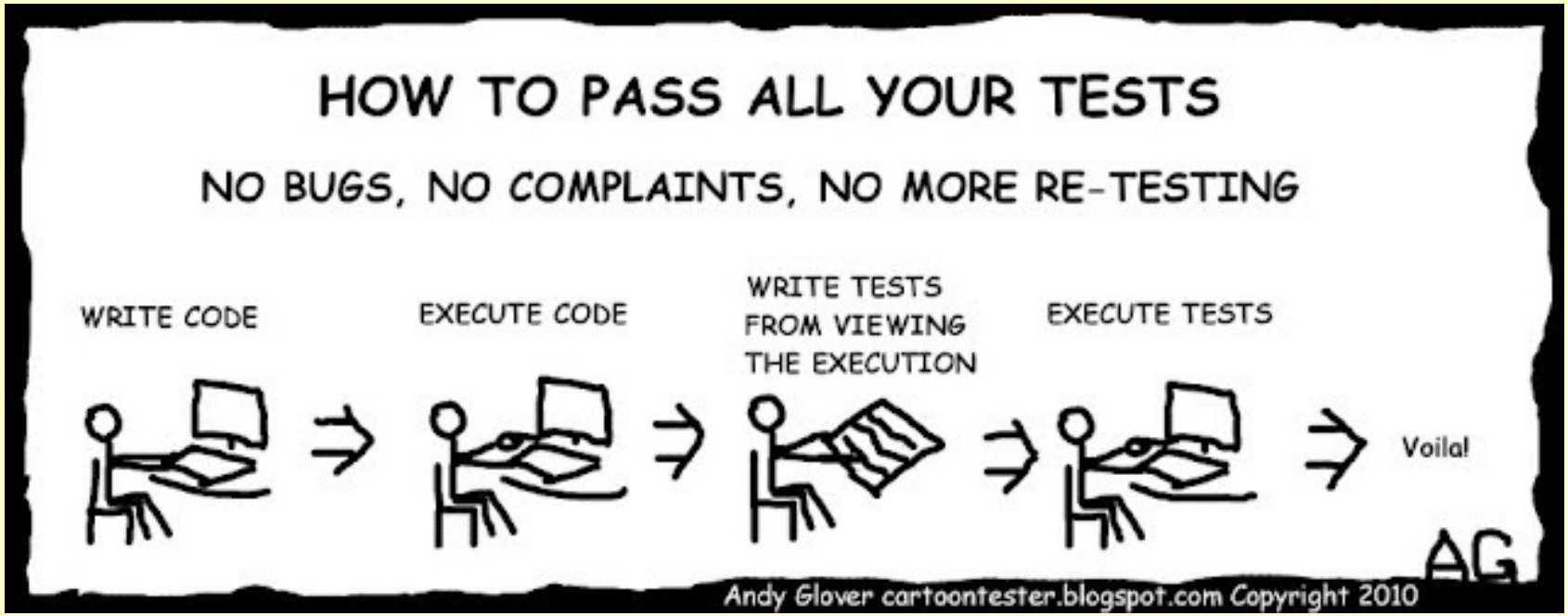
Why do we need tests?



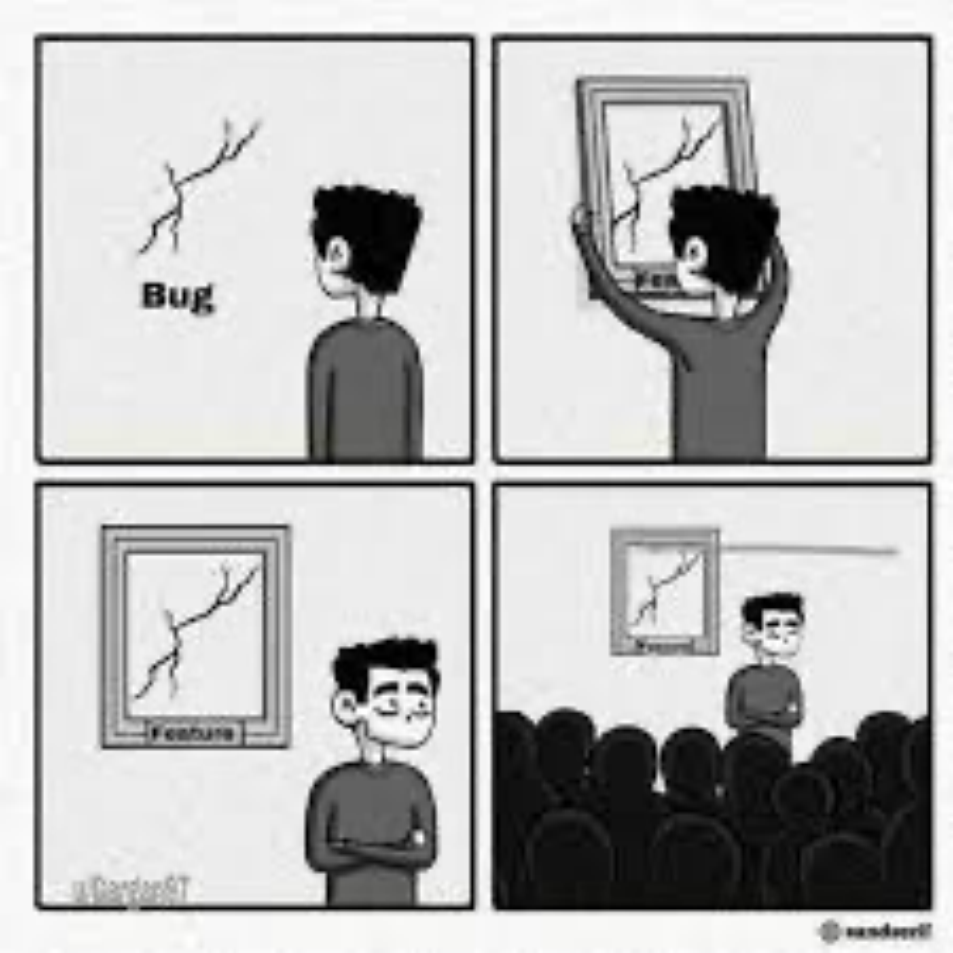
Do we test our software like this?



Do we test our software like this?



Software Testing Quotes



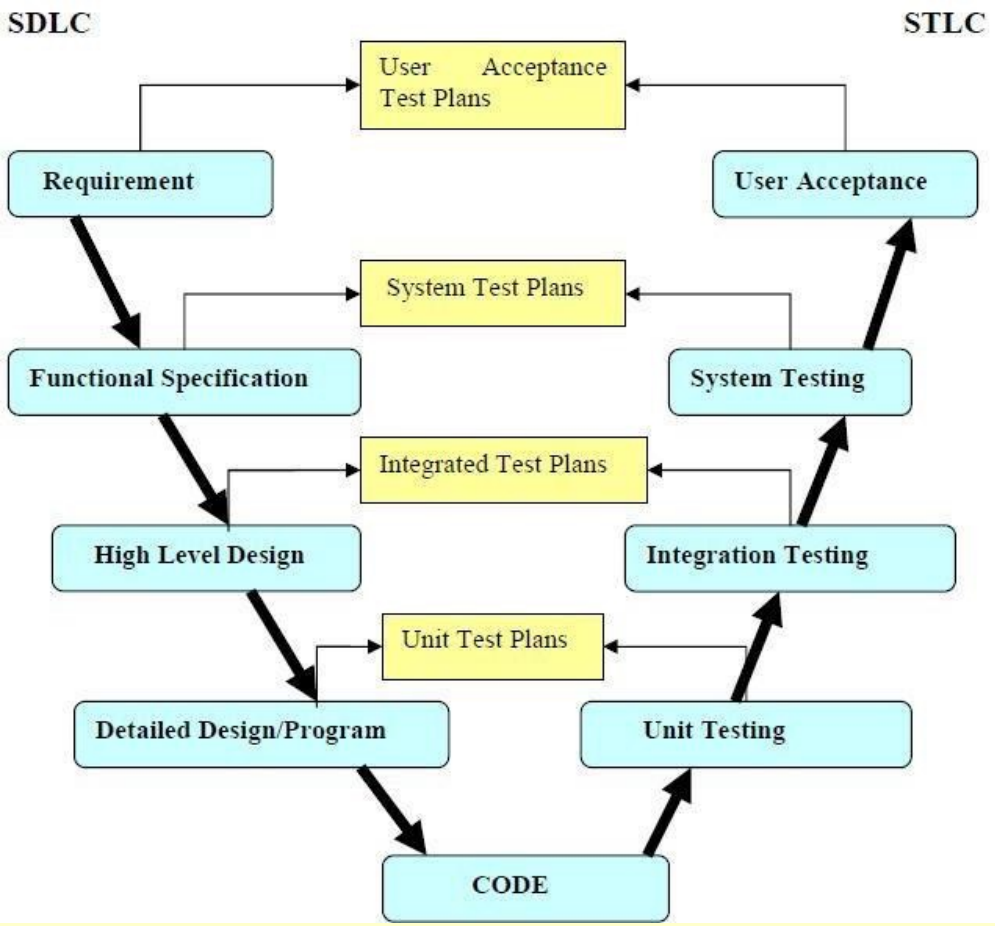
https://www.wired.com/story/its-not-a-bug-its-a-feature/

It's not a bug, it's a feature

http://testautomation.applitools.com/post/98802238427/41-awesome-quotes-about-software-testing

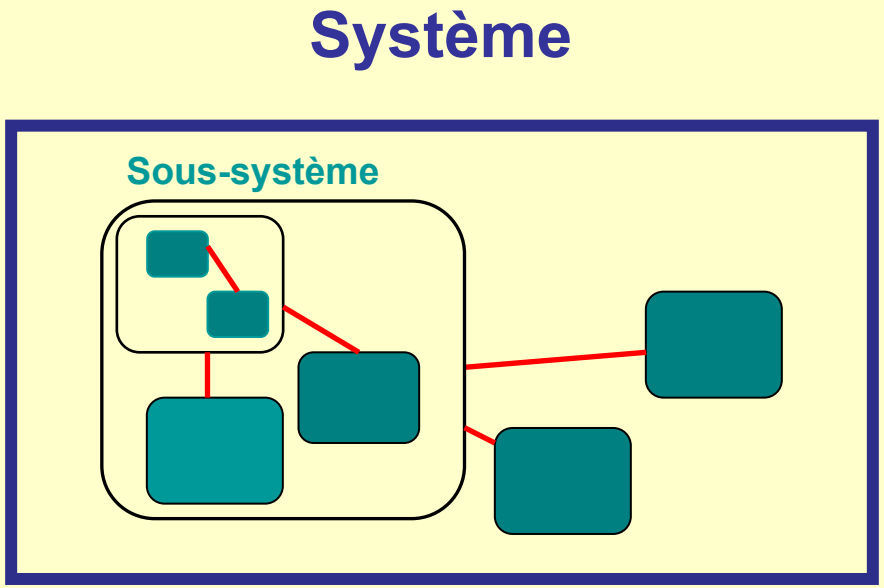
Testing : in the V life cycle

Software Development Life Cycle **Software Test Life Cycle**

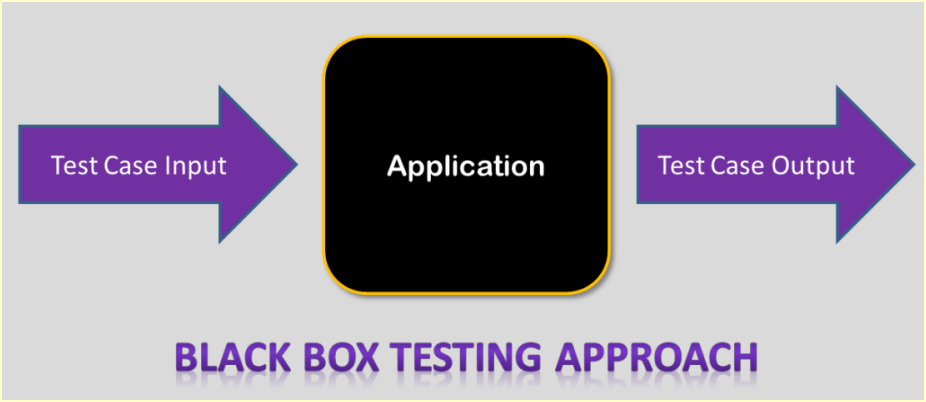


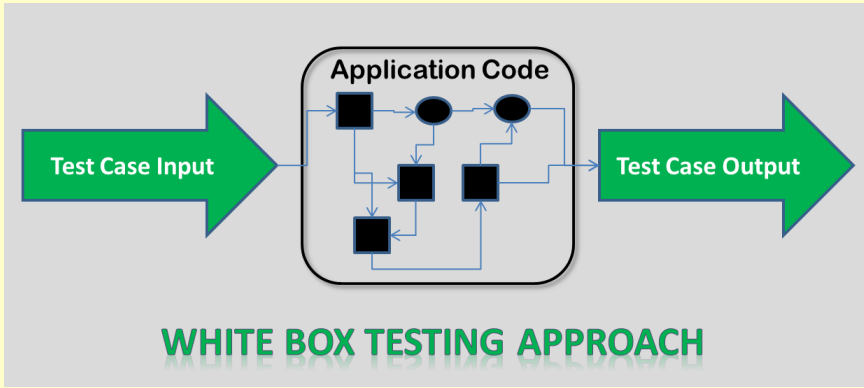
Testing : in the V life cycle

- Validation
- Intégration
- **Unit**



Black box or White box Testing

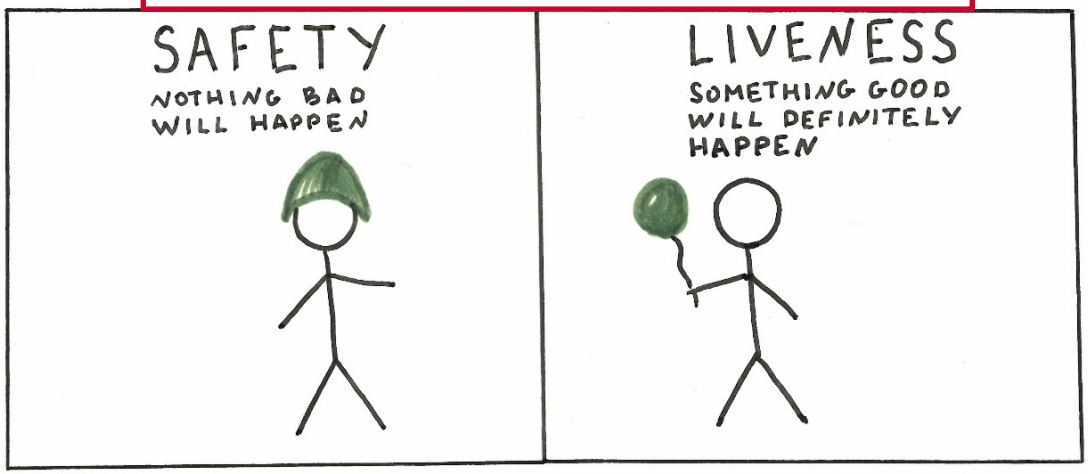




Question: advantages and disadvantages of each?

Black box or White box Testing

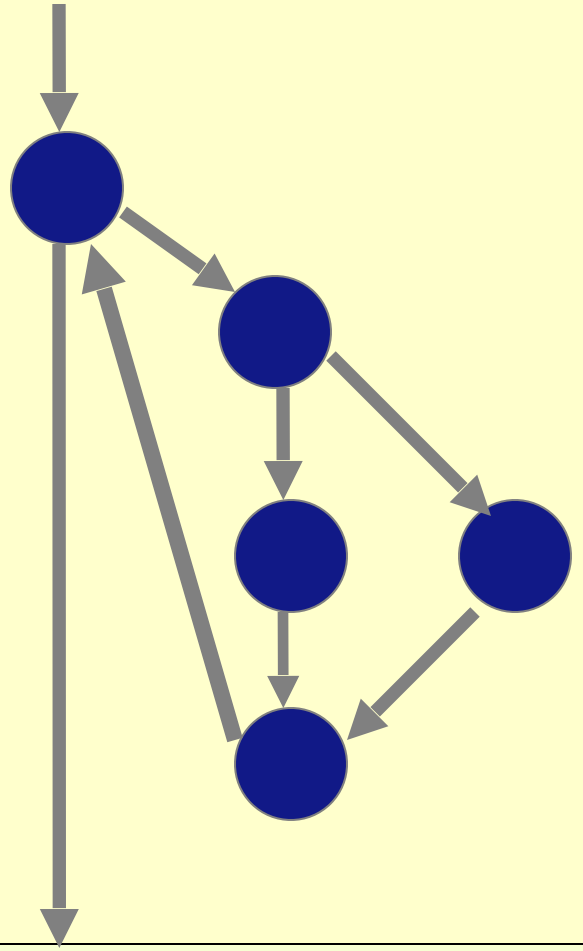
Testing safety is much easier than testing liveness



White box permits static analysis

```
for (int i=0; i<n; ++i) {
        if (a.get(i) ==b.get(i))
                x[i] = x[i] + 100;
        else
                x[i] = x[i] /2;
        }
```

n	paths
1	3
2	5
3	9
10	1025
20	1048577
60	>1,15 10^{18}



Unit Testing

Most (nearly all) programming languages have automated tool support for unit testing (as well as other types of testing)

JUnit CUnit xUnit etc ….

Whenever you learn a new programming language, learn the testing tool(s) that come with it

Automated Unit testing is very valuable and beginners to programming need to learn it ASAP

https://smartbear.com/blog/test-and-monitor/a-short-lecture-on-the-value-and-practice-of-unit/

Equivalence classes + boundary values

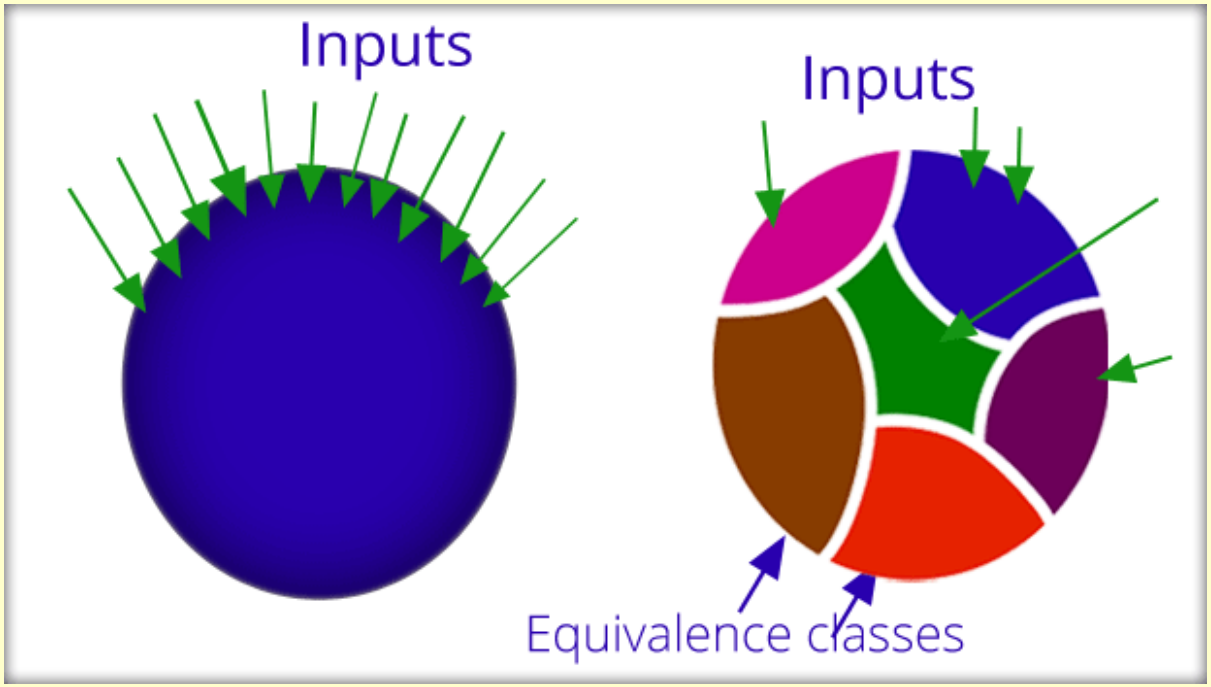
Partition the input domain into equivalence classes to be covered

Classes determined from the functional requirements (set of values for which functional behavior is the same)

Consider both valid and invalid classes (robustness testing)

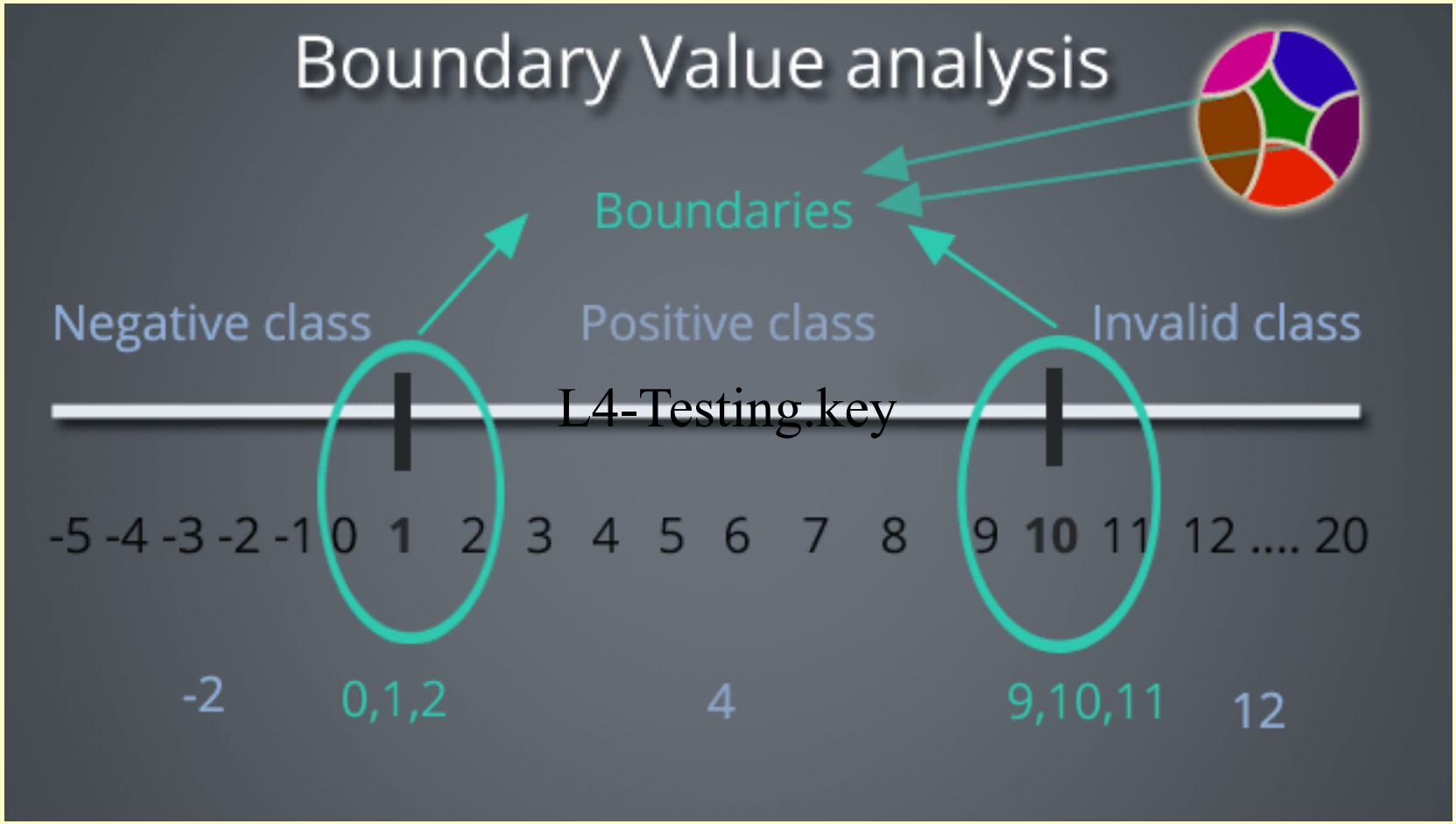
Identify boundary values for dedicated tests E.g., -1, 0, 1, +/- MAXINT

Unit Testing



http://www.testnbug.com/2015/01/equivalence-class-partitioning-and-boundary-value-analysis-black-box-testing-techniques/

Unit Testing



Integration Testing

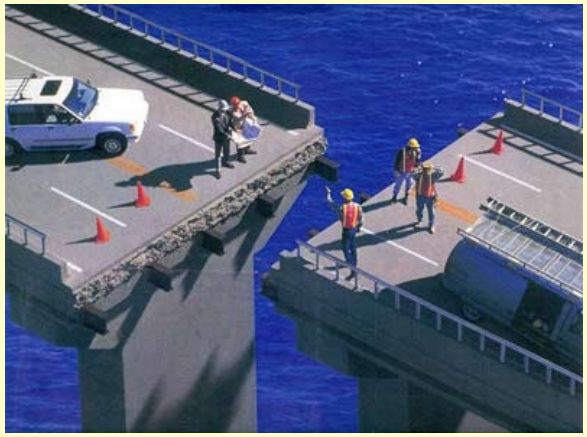
Why do we not just do unit tests?

"2 unit tests, zero integration tests"

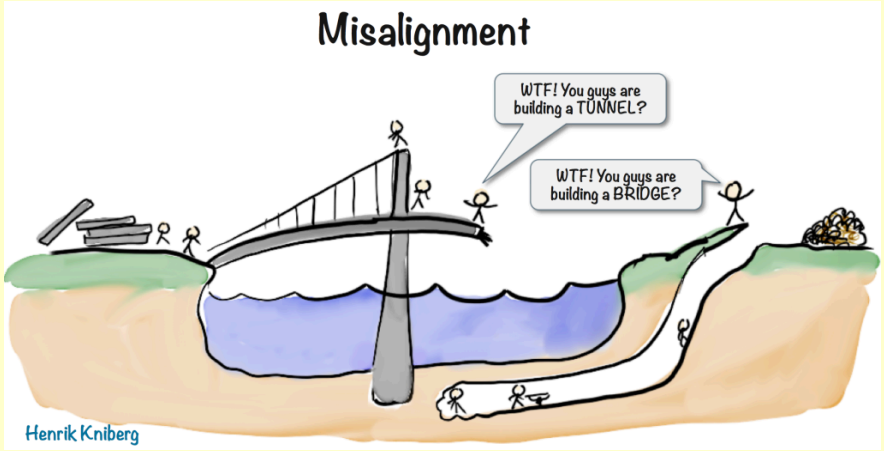
http://i.imgur.com/qSN5SFR.gifv

Why do we not just do validation tests?

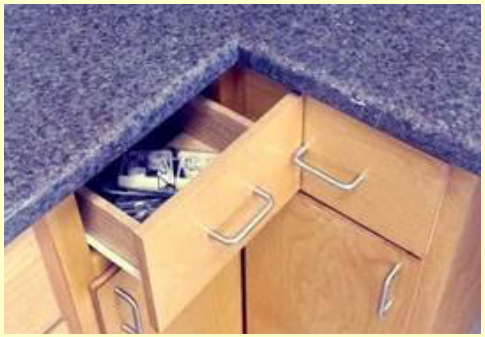
Integration Testing



Fixing problems later in development can cost much more than fixing them earlier - but you have to detect them first



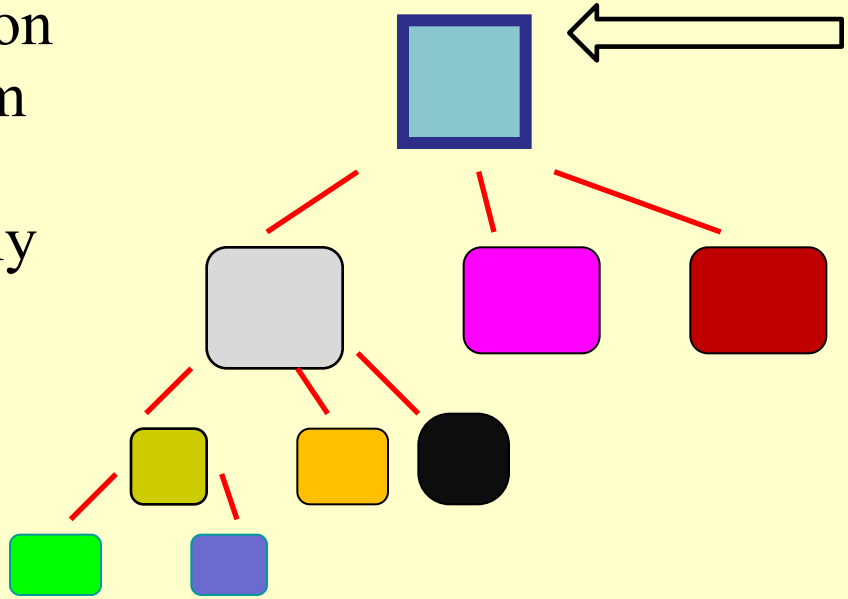
Integration Testing - more examples



Integration Testing

Big Bang – system validation – « if the system works then it must be properly integrated? »

Test at the system interface



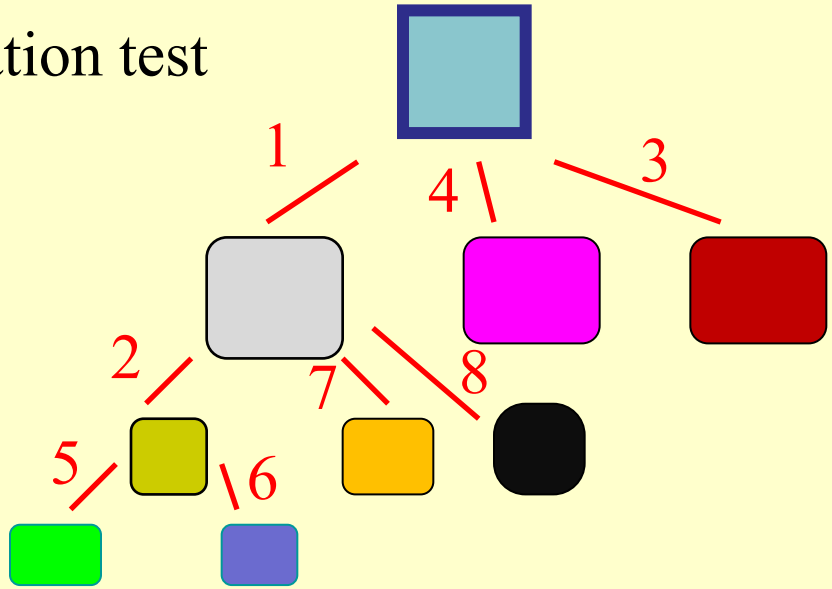
MAIN PROBLEMS:

- Testing produces errors, but what caused them?
- Testing misses bugs
- Testing starts after all components are coded

Integration Testing

Top Down– test all interactions between components (*from root to leaves*).

Possible integration test sequence



NOTES:

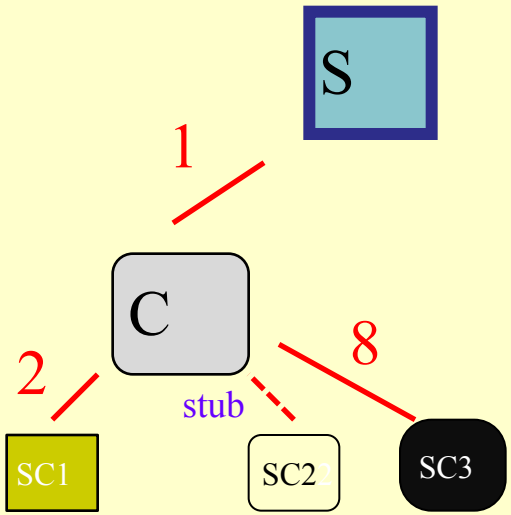
- In reality, one will execute many different tests for each branch of the tree.
- *Top Down* integration is usually performed breadth first or depth first.

Integration Testing

Top Down– the need for stubs

In top down testing, we may start the tests before all components are coded. In such a case, we may have to write stubs – pieces of dummy code that simulate behaviour that is not yet coded. In OO, stubs simulate called methods.

NOTE: When code for other lower level components is completed, the relevant stubs are no longer needed.



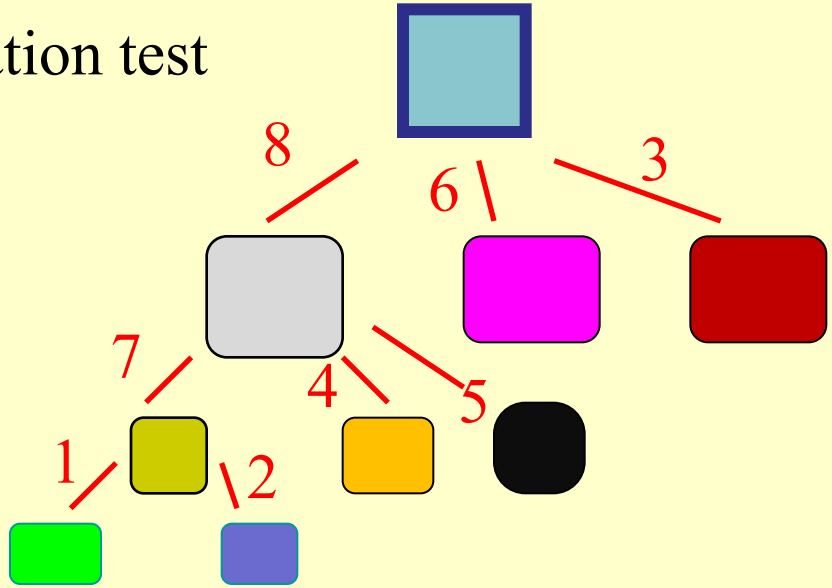
To execute an integration test on branch 1 we may have to simulate method calls between C and lower level components that are not yet implemented.

For example, if SC2 is not yet coded we create a stub (dummy code) which simulates the method call between C and SC2

Integration Testing

Bottom-Up– test all interactions between components (*from leaves to root*).

Possible integration test sequence



NOTES:

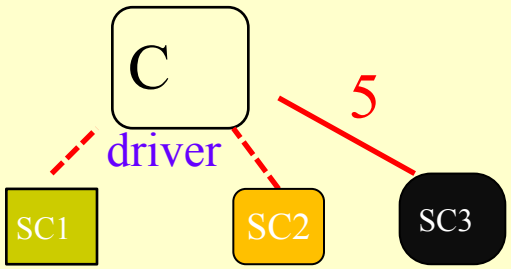
- In reality, one will execute many different tests for each branch of the tree.
- *Bottom Up* integration is usually performed breadth first or depth first.

Integration Testing

Bottom-Up– the need for drivers

In bottom up testing, we may start the tests before all components are coded. In such a case, we may have to write drivers– pieces of dummy code that simulate behaviour that is not yet coded. In OO, drivers simulate calling methods

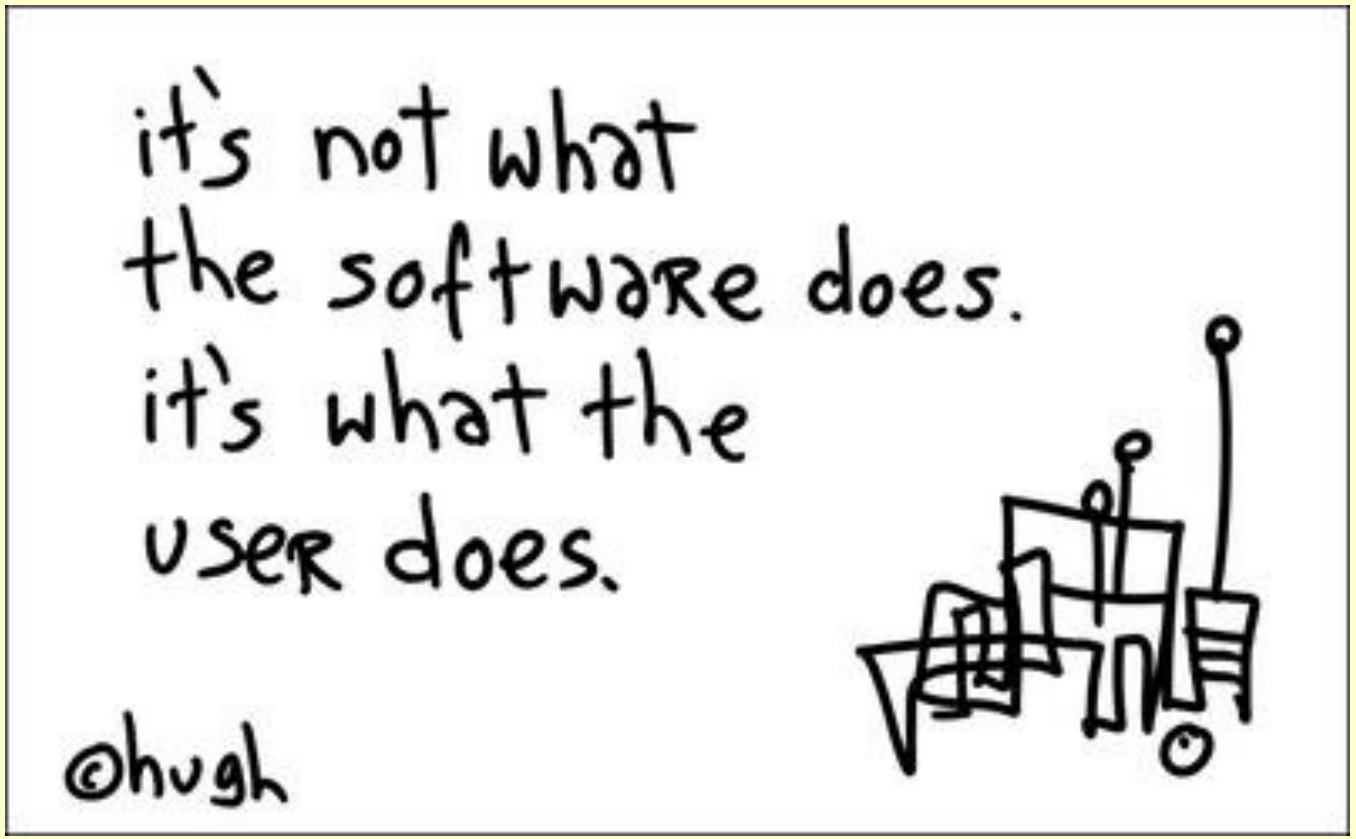
NOTE; When code for other higher level components is completed, the relevant drivers are replaced.



For example, to execute an integration test on branches 7 and 4 we may have to simulate method calls from C (which is not yet coded) to SC1 and SC2.

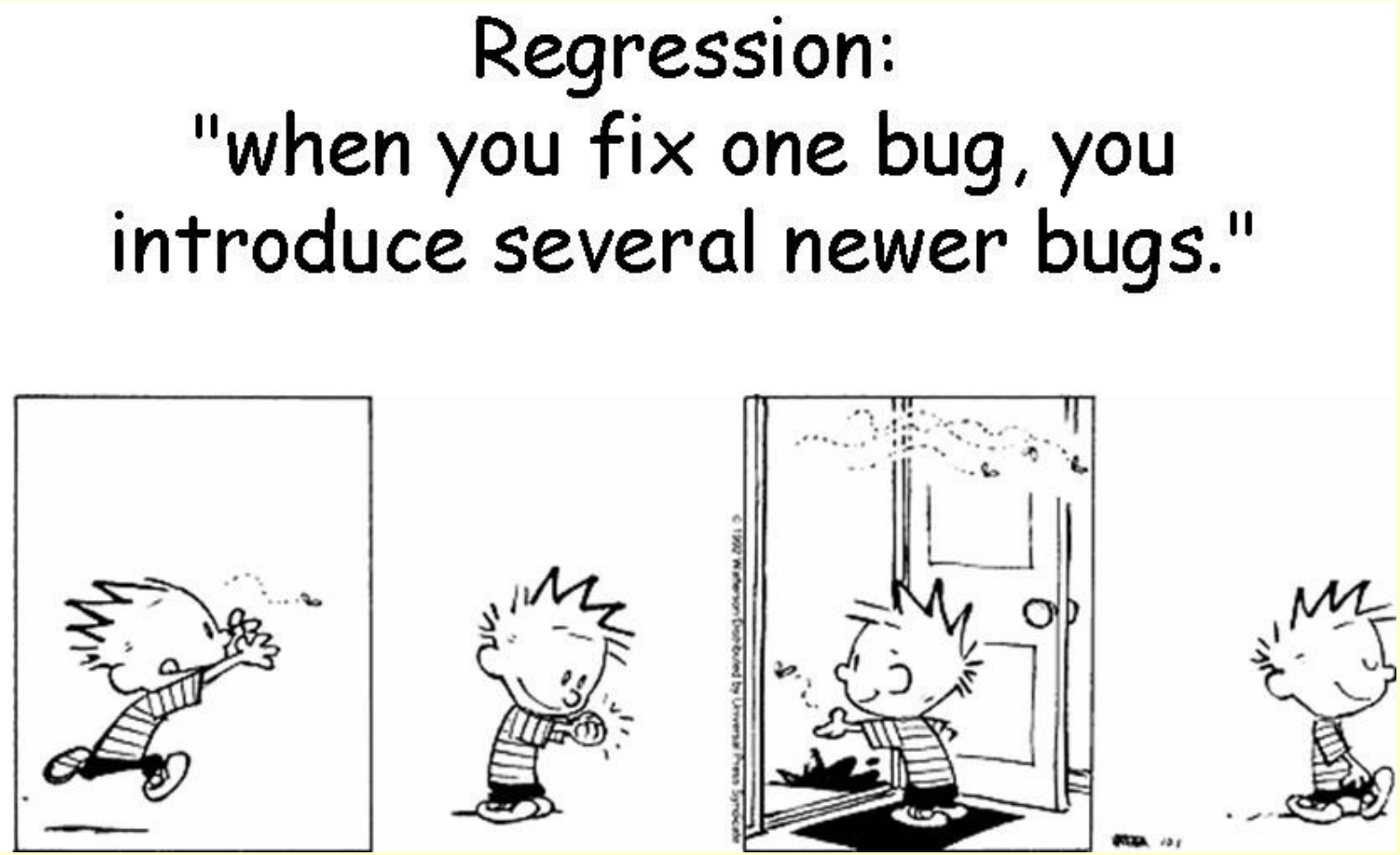
So, we create a driver (dummy code) which simulates the sequence of method calls from C to SC1 and SC2.

System Testing

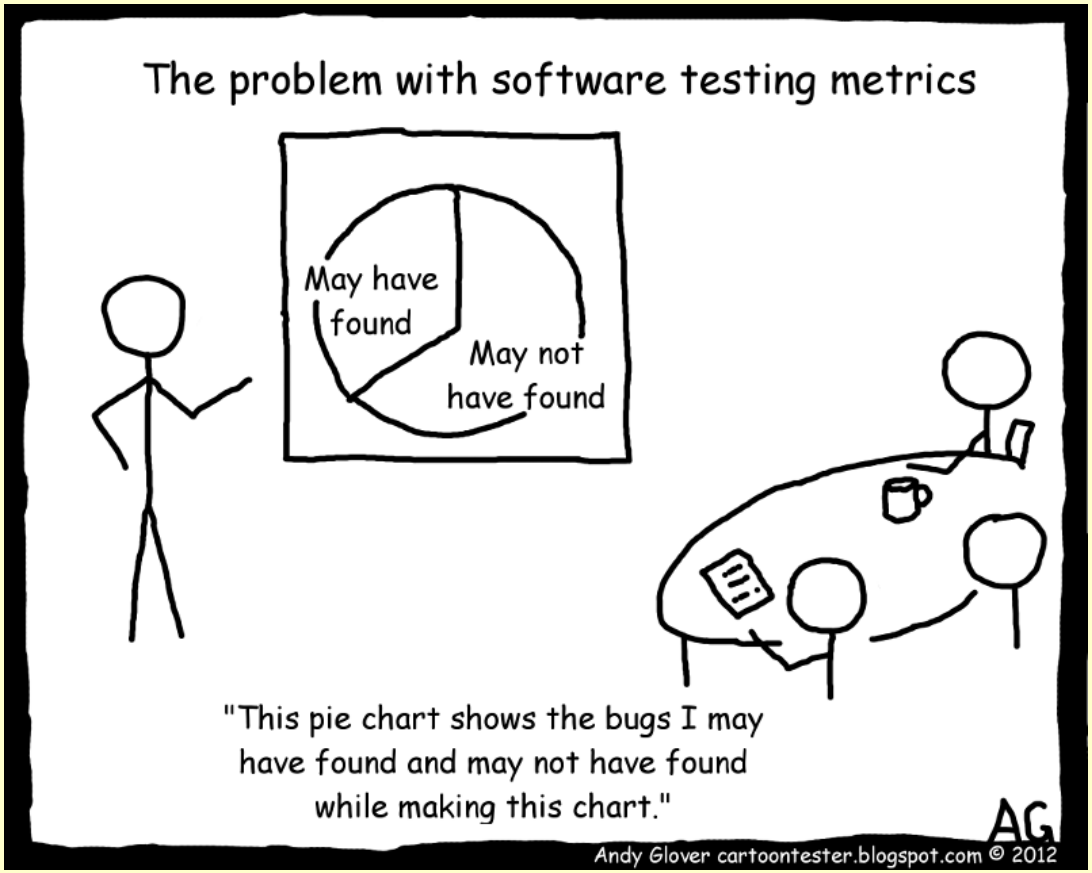


Never underestimate the users' ability to surprise

Regression Testing



Testing Metrics



Testing Code Coverage

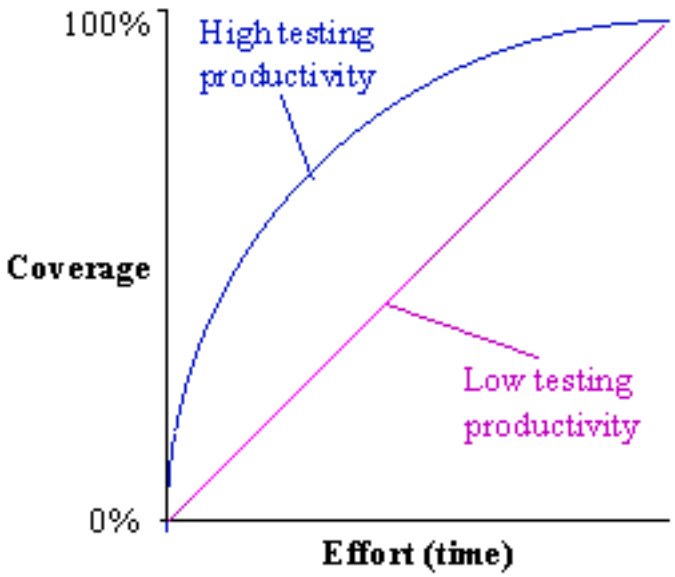


Figure 1: Coverage rate

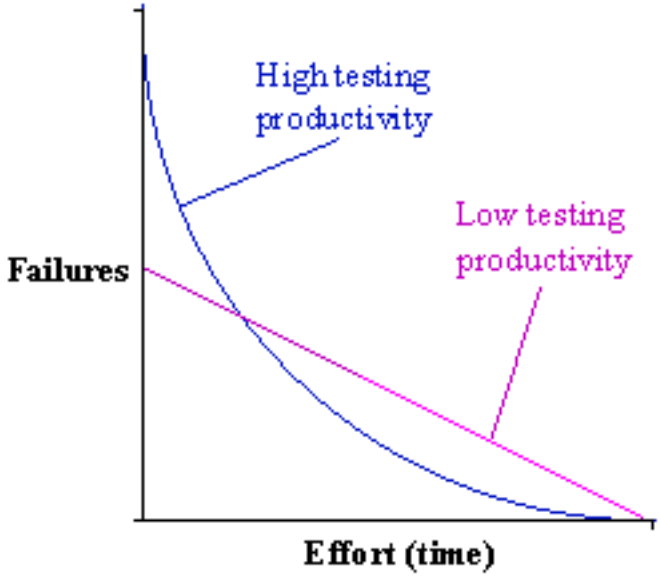
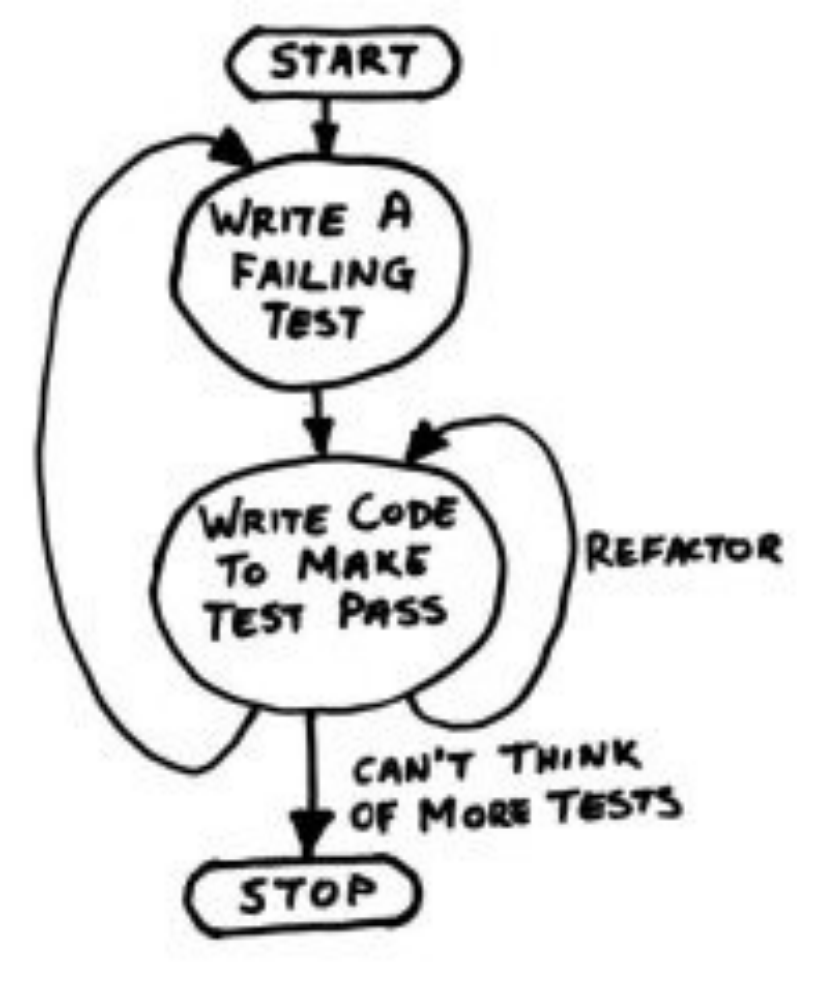


Figure 2: Failure discovery rate

Most testing tools come/work with coverage tools

Test first development



Some other testing types

Functional - Nonfunctional
Performance
Usability
Security
Accessibility
Internationalisation/ localisation etc …

Security “goto fail” – Apple’s SSL bug

sslKeyExchange.c

```
. . .
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
. . .
```

if `err` is zero and there is actually no error to report.
The result is that the code leaps over the vital call to `sslRawVerify()`, and exits the function.
This causes an immediate “exit and report success”, and the TLS connection succeeds, even though the verification process hasn’t actually taken place.
A skilled attacker can easily exploit this

Security

Intel's AMT Vulnerability

In April 2017, Intel announced a critical privilege escalation bug that was laying around its Active Management Technology (AMT) login page for the past seven years. The exploit allows a remote attacker to take control of vulnerable devices with ease.

The login code for the AMT web interface *incorrectly* used the strncmp function, which allowed users to gain access when inserting an empty password at the login screen.

Security

The buggy code

```
int main () {
  string realpass = "secret";
  string userpass = "user-secret";
  int equal = strncmp(realpass.c_str(), userpass.c_str(), userpass.size());
  if (equal == 0) {
     printf ("'%s' equals to '%s'", realpass.c_str(), userpass.c_str());
  }
  return equal * equal; // make sure it's positive
}
```

Question: can you see the problem?

The Mega Conspiracy: using The Ken Thomson Hack

https://oded.ninja/2017/05/14/amt-n-ken-hack/

What if someone hacked into Intel's servers a few years ago, and updated their compiler to replace this:

```
strncmp(realpass.c_str(), userpass.c_str(), realpass.size())
```

C++Copy

with this:

```
strncmp(realpass.c_str(), userpass.c_str(), userpass.size())
```

C++Copy

Essentially adding a backdoor? What if the same attacker added code that turned off the attack when test runners were used? or when the compiler was running inside Intel's LAN? This might sound crazy and far-fetched, but there are threat actors out there with the skill-set to pull this off. But hey, I'm not that paranoid. I do believe the vulnerability was introduced as a result of a human mistake... or not?

Your test code needs testing?

Some PBL

1. “Simple” Unit test
2. Test first Programming
3. Equivalence Classes - Line Overlap
4. Invariants

Some PBL "Simple" Unit test

Write test code to test a function/method that calculates the average of 2 numbers

Test-First Programming

The tests should drive you to write the code, the reason you write code is to get a test to succeed, and you should **only write the minimal code** to do so. Note that test-first-design is more than just unit testing. Unit testing by itself does not change the design of the code. In addition to documenting how code should be used, test-first-design helps you keep the design simple right from the start, and keeps the design easy to change.

Test-First Programming

As project complexity grows, you may notice that writing automated tests gets harder to do. This is your early warning system of overcomplicated design. Simplify the design until tests become easy to write again, and maintain this simplicity over the course of the project.

Test-Code-Simplify Cycle

- Write a single test
- Compile it. It shouldn't compile, because you haven't written the implementation code it calls
- Implement just enough code to get the test to compile
- Run the test and see it fail
- Implement just enough code to get the test to pass
- Run the test and see it pass
- Refactor for clarity and "once and only once"
- Repeat

Some PBL

Test first Programming

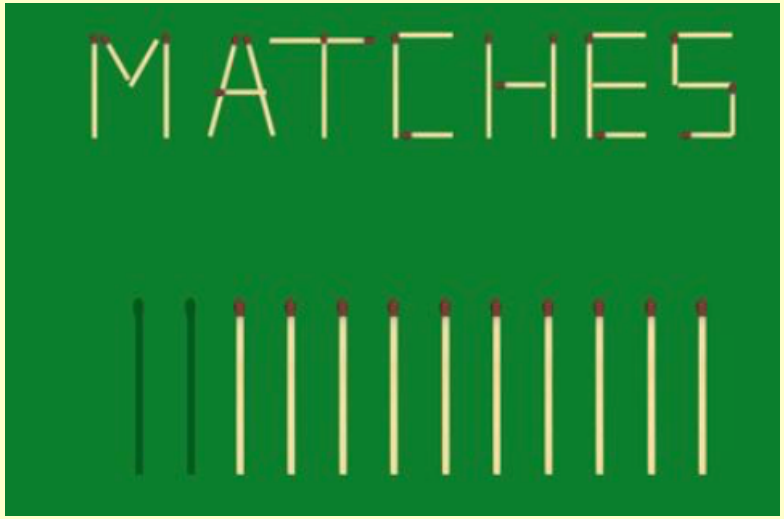
Putting It All Together

Working Example: http://blog.differentpla.net/blog/2004/01/12/test-first-roman-numeral-conversion

10 iterations to …

```
std::string toRoman(int n)
{
    std::string r;

    struct TO_ROMAN {
        int num;
        const char *str;
    } to_roman[] = {
        { 1000, "M", },
        { 900, "CM", },
        { 500, "D", },
        { 400, "CD", },
        { 100, "C", },
        { 90, "XC", },
        { 50, "L", },
        { 40, "XL", },
        { 10, "X", },
        { 9, "IX", },
        { 5, "V", },
        { 4, "IV", },
        { 1, "I", },
    };

    for (int q = 0; q < sizeof(to_roman) / sizeof(to_roman[0]); ++q)
    {
        TO_ROMAN *t = &to_roman[q];

        while (n >= t->num)
        {
            n -= t->num;
            r += t->str;
        }
    }

    return r;
}
```

The matches game



Rules: This is a 2-player game. The game starts with a random number of matches (between 15 and 30, say) on the table. Each player must remove 1,2 or 3 matches. They play alternately until only 1 match is left on the table. The player who is left with a single match loses the game.

6.b. Develop a matches game player following a TFP approach

The player code should be a single function that takes as input the number of matches left on the table, and returns the number of matches to be removed. The player code should be as intelligent as possible so that it will always win the game (if a win is possible).

Please submit your solution as a sequence of code and test pairs. The final pair should be your best solution that passes all the tests. (I expect the sequence will have a length between 4 and 8).

Equivalence Classes - The Line Overlap Problem

Consider the integer number line:

… -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 …

We can define a segment on this line by a range (minimum … maximum)

Below, we illustrate 2 segments: **(-3, 1)** and **(0,6)**

… -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 …

In this example, the 2 segments are said to overlap on the line because they share at least 1 point in common.
The overlap in this case is the segment (0, 1).

The Line Overlap Problem

Requirements and Tests

The problem is to write a program that can calculate whether any 2 segments overlap on the integer line. It is to return the segment overlapped as the result of the program (an "empty" segment if there is no overlap)

6.c) You are to specify and implement a test set for this problem. You are not to code a working solution until after your tests are coded.

Your test code should be written in the same programming language as the solution(s) which you will be expected to test.

The code (including tests) must be well documented.

Illustrate that your tests can find errors in an incorrect solution

The Line Overlap Problem

Complete Tests

Given the minimum and maximum values, how many tests must be executed if we wish to test (exhaustively) every possible case?

Invariants to be tested

Invariants are boolean properties on the state of a system (or system part) that should always be true.

For example:

- *if an integer is used to represent the age of somebody then this integer must always be non negative.*
- *if a pair of colours (with 3 possible values - red, amber, green) represent 2 traffic lights at a junction then it must always be true that if one colour is amber or green then the other must be red*
- *in the game of poker it is always true that no card can appear more than once in the hand of a player*

Invariants that are false/broken correspond to unsafe states of systems. Testers must check that systems cannot arrive in such unsafe states.

Invariants

6.d. Specify (and code) the invariant for a game of connect-4 with 2 opposing players. The code should be a boolean function which takes as input the state of the game and returns true if it is in a safe state (respects the game rules) and false otherwise.