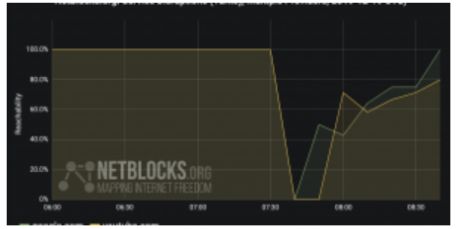
Could better testing have helped …

Examples



REPORTS · DECEMBER 19, 2019

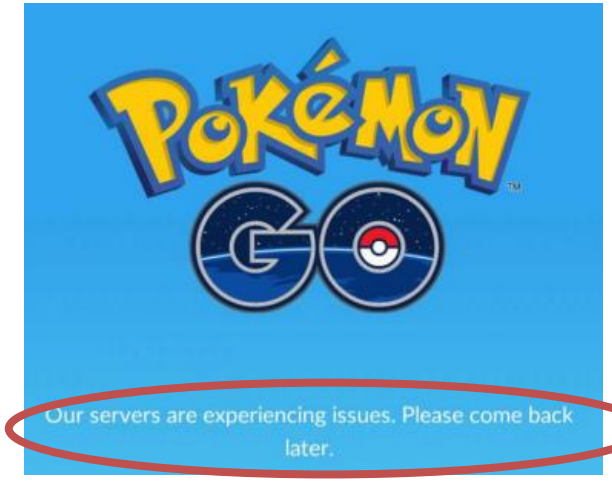
YouTube, GMail and Google services down in multiple countries

Network data from the NetBlocks internet observatory confirm outages of YouTube, GMail and other Google services in multiple countries. The disruptions are consistent with a

RBS and NatWest customers 'had loan repayments taken twice'

Current account holders advised to double-check balances as a spokeswoman admits a 'relatively small' number have had personal loan repayments taken twice

Royal Bank of Scotland BS fined £56m over IT meltdown in 2012



ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE

CLOUD ACADEMY —

Cloud Bottlenecks: How *Pokémon Go* (and other game dev teams) caught them all

Lesson: "Something that works with two million users doesn't always work with 10 million."

MATTHEW ROTHENBERG - 1/26/2017, 1:30 PM

CWE Top 25 Software Weaknesses

#1 CWE-787 Out-of-bounds Write	#2 CWE-79 Cross-site Scripting	#3 CWE-89 SQL Injection	#4 CWE-416 Use After Free	#5 CWE-78 OS Command Injection
#6 CWE-20 Improper Input Validation	#7 CWE-125 Out-of-Bounds Read	#8 CWE-22 Path Traversal	#9 CWE-352 Cross-Site Request Forgery	#10 CWE-434 Unrestricted Upload of File
#11 CWE-862 Missing Authorization	#12 CWE-476 NULL Pointer Deference	#13 CWE-287 Improper Authentication	#14 CWE-190 Integer Overflow or Wraparound	#15 CWE-502 Deserialization of Untrusted Data
#16 CWE-77 Command Injection	#17 CWE-119 Improper Restriction of Operations....	#18 CWE-798 Use of Hard-coded Credentials	#19 CWE-918 Server-Side Request Forgery	#20 CWE-306 Missing Authentication for Critical...
#21 CWE-362 Race Condition	#22 CWE-269 Improper Privilege Management	#23 CWE-94 Code Injection	#24 CWE-863 Incorrect Authorization	#25 CWE-276 Incorrect Default Permissions

Which is the most dangerous error?

- Integer overflow
- SQL injection
- Cross-site scripting
- Null pointer dereference
- Out-of-bounds read
- Out-of-bounds write

SQL injection

Assume this code:

```
String query =
  "select from table where user='" + username + "'";
```

SQL injection

Assume this code:

```
String query =
  "select from table where user='" + username + "'";
```

If the user enters “mernst”, the value of `query` is:

SQL injection

Assume this code:

```
String query =
  "select from table where user='" + username + "'";
```

If the user enters "mernst", the value of `query` is:

```
  "select from table where user='mernst'"
```

SQL injection

Assume this code:

```
String query =
  "select from table where user='" + username + "'";
```

If the user enters “mernst”, the value of `query` is:

```
  "select from table where user='mernst'"
```

What if a user enters, as their username: ' or '='

The value of `query` is:

SQL injection

Assume this code:

```
String query =
  "select from table where user='" + username + "'";
```

If the user enters “mernst”, the value of `query` is:

```
  "select from table where user='mernst'"
```

What if a user enters, as their username: `' or "='`

The value of `query` is:

```
  "select from table where user=''' or ''='''"
```

SQL injection

Assume this code:

```
String query =
  "select from table where user='" + username + "'";
```

If the user enters “mernst”, the value of `query` is:

```
  "select from table where user='mernst'"
```

What if a user enters, as their username: `' or '='`

The value of `query` is:

```
  "select from table where user='' or ''=''"
```

CWE Problems Result in Major general Problems

CWE Top 25 Most Dangerous Software Weaknesses

Errors identified as root cause of reported vulnerabilities in OWASP Top 10

OWASP Top 10	CWE Top 25
Broken Access Control	CWE - 22, CWE - 352, CWE - 276
Cryptographic Failures	None
Injection	CWE - 79, CWE - 89, CWE - 20, CWE - 78, CWE - 77, CWE - 94
Insecure Design	CWE - 434
Security Misconfiguration	CWE - 611
Vulnerable and Outdated Components	CWE - 190
Identification and Authentication Failures	CWE - 287, CWE - 798, CWE - 306
Software and Data Integrity Failures	CWE - 502
Security Logging and Monitoring Failures	None
Server-Side Request Forgery (SSRF)	CWE - 918
Others	CWE - 119, CWE - 125, CWE - 362, CWE - 400, CWE - 416, CWE - 476, CWE - 787, CWE - 862

So let's test! Four categories of testing

1. Unit Testing
   - Does each module do what it is supposed to do in isolation?
2. Integration Testing
   - Do you get the expected results when the parts are put together?
3. Validation Testing
   - Does the program satisfy the requirements?
4. System Testing
   - Does the program work as a whole and within the overall environment? (includes full integration, performance, scale, etc.)

What are other testing-related terms?

Let's see if we can name at least 10:

- Regression testing
- Black box, white box testing
- Code coverage testing
- Boundary case testing
- Test-driven development
- Mutation testing
- Fuzz testing
- Performance testing
- Usability testing
- Acceptance testing

Testing vs. debugging

- **Testing**:

  is there a defect?

- **Debugging**:

  where is the defect?

  how to fix the defect?

Regression testing

- Whenever you find a defect
  - Store the input that triggered that defect, plus the correct output
  - Add these to the test suite
  - Verify that the test suite fails
  - Fix the defect
  - Verify the fix
- Ensures that your fix solves the problem
- Helps to populate test suite with good tests
- Protects against reversions that reintroduce the defect
  - It happened at least once, and it ~~might~~ will happen again

Also look for similar defects elsewhere!

Time out: How else can we build in quality?

What can we do beyond testing?
Hint: build in quality from the start ☺️

- Good architecture, design, and planning
- Reuse Architecture Pattern
- Use Design Principals
- Coding style guides, linting
- Code reviews
- Atomic commits
- Pair programming
- …

Today's outline

Software testing

- Motivating examples
- Categories of tests
- **Unit testing**
  - Black box testing
    - Boundary case testing
    - Test driven development
  - White-box testing
    - Static code analysis
    - Code coverage testing

We are here

Unit testing

- A **unit** is an **independently testable part** of the software system (e.g. in Python or Java, a method, a class, …).
- **Goal**: Verify that each software unit performs as specified.
- **Focus**:
  - Individual units (not the interactions between units).
  - Usually input/output relationships.

Testing best practices: motivating example 1

Average of the absolute values of an array of doubles

```
public double avgAbs(double ... numbers) {

  // We expect the array to be non-null and non-empty
  if (numbers == null || numbers.length == 0) {
    throw new IllegalArgumentException("Array numbers must not be null or empty!");
  }

  double sum = 0;
  for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
      sum -= d;
    } else {
      sum += d;
    }
  }

  return sum/numbers.length;
}
```

What tests should we write for this method?

Lab t-01

- Implement the method from the slide before in Python.
- Write a couple of tests as simple methods in Python.
- Print the test results to the console

Two types of unit testing

Black box testing

Written without knowledge of the code

Treats the module/system as atomic

Best simulates the customer experience

White box testing

Written with knowledge of the code

Examines the module/system internals

Aims to maximize a measure of test suite quality

Black-box testing

- Black-box is based on specifications (requirements and functionality), not code
- Tester does not look at the code while constructing the tests
- Work from the user or client's perspective

How do you know when you are done?

You have tested all the behaviors, according to the specification?

How do you know when you have tested all the behaviors?

What if the behavior differs from the specification?

Approach:

- build tests according to the text of the specification
  - “cover” the specification
- guess about what errors the programmer might have made
  - add more tests based on these guesses/heuristics

Black box: boundary case testing

Boundary case testing:

- What: test edge conditions
- Why?
  - #1 and #7 Most Dangerous Software Weakness!
  - Off-by-one defects are common (< vs. <=, etc.)
  - Requirement specs may be fuzzy about behavior on boundaries
  - Often uncovers internal hidden limits in code

Lab t-02

- Implement black box testing

Black box: boundary case example #1

- Write test cases based on paths through the **specification**

```
int find(int[] a, int value) throws Missing
// returns: the smallest i such that a[i] == value
// throws:  Missing if value not in a[]
```

- Two obvious tests:
  ([4, 5, 6], 5) ⇒ 1
  ([4, 5, 6], 7) ⇒ throw Missing
- Have we captured all the behaviors?

  ([4, 5, 5], 5) ⇒ 1

Boundary case #3

```
public int abs(int x)
  // returns: |x|
```

- What are some values or ranges of x that might be worth probing?
  - `x < 0, x ≥ 0`
  - `x = 0 (boundary condition)`
  - `Specific tests: say x = -1, 0, 1`

Boundary case #3 (arithmetic overflow)

```
public int abs(int x)
  // returns: |x|
```

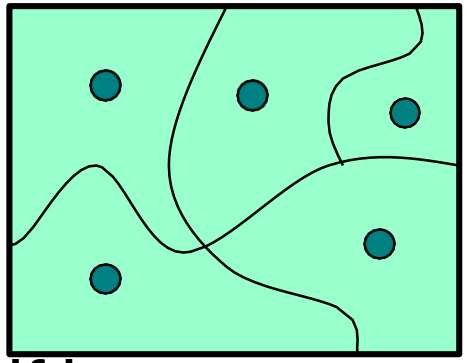
- What are some values or ranges of x that might be worth probing?
  - `x < 0, x ≥ 0`
  - `x = 0 (boundary condition)`
  - `Specific tests: say x = -1, 0, 1`
- How about…

```
int x = -2147483648;                // this is Integer.MIN_VALUE
System.out.println(x<0);            // true
System.out.println(Math.abs(x)<0); // also true!
```

Javadoc of **abs** says …if the argument is equal to the value of Integer.MIN_VALUE, the most negative representable int value, the result is that same value, which is negative

Theory explains why boundary testing works

- Divide the input into **subdomains**
  - **A subdomain is a subset of possible inputs**
  - Identify input sets with same behavior
  - Try one input from each set



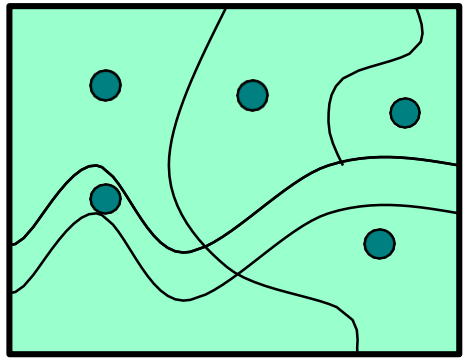
- A program has the "same behavior" on two inputs if it:
  1) gives correct result on both, or
  2) gives incorrect result on both
  - "Same behavior" is unknowable.
  - A subdomain is revealing for an defect, E, if each test input fails.

If the program has a defect, it is revealed by a test in its revealing subdomain

Lab t-03

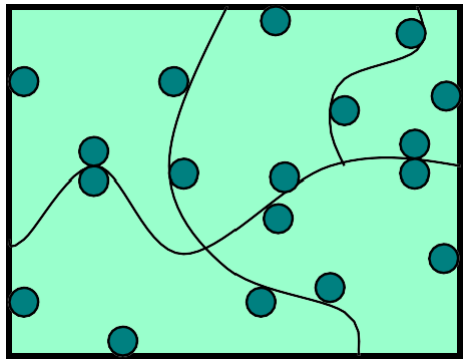
- Come together in groups and collect examples for boundaries of the Python function of lab t-03.

What if you mis-drew the boundaries?



Boundary case testing heuristic

- Create tests at the **boundaries** of subdomains

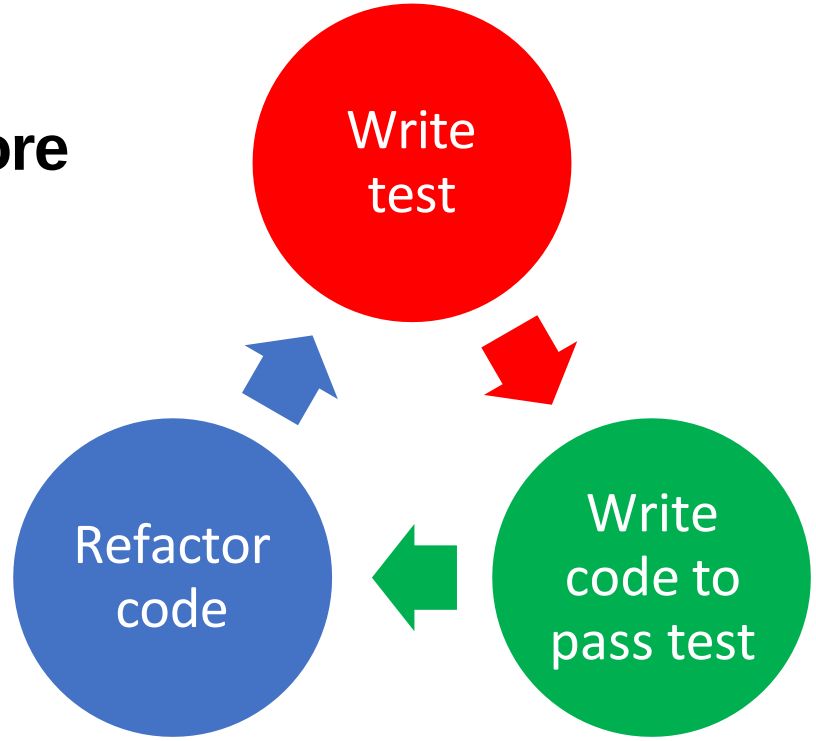


- Catches common boundary case defects:
  - **Arithmetic**
    - Smallest/largest values
    - Zero
  - **Objects**
    - Null
    - Circular
    - Same object passed to multiple arguments (aliasing)
  - **You drew the boundaries wrong**

Black box: test driven development

Test driven development (TDD):

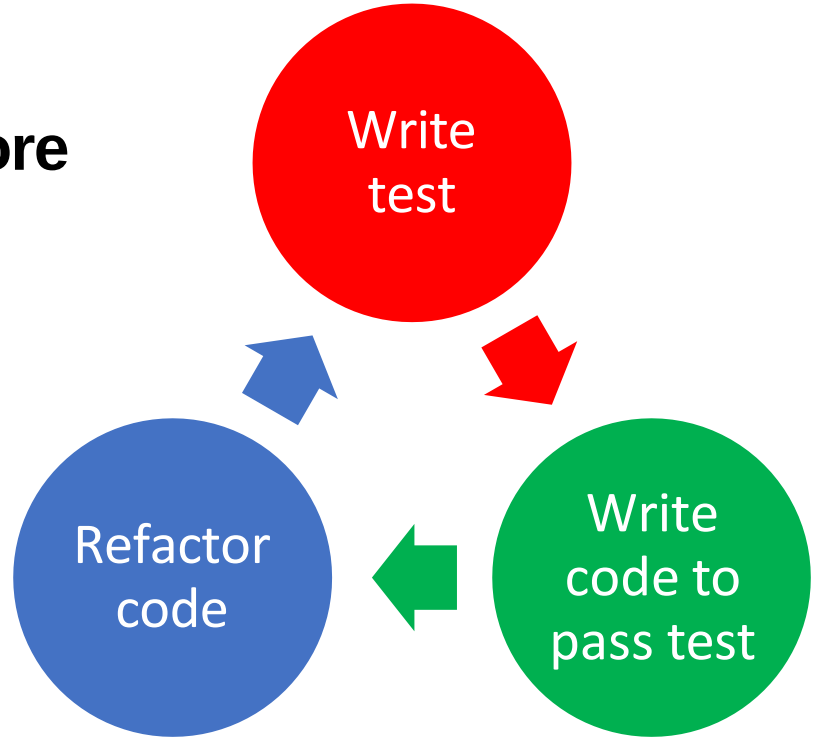
- What:
  - Test is based on the spec and is developed **before** the code is written
  - Will fail initially
  - Write just enough code to make it pass!
- Why?



Black box: test driven development

Test driven development (TDD):

- What:
  - Test is based on the spec and is developed **before** the code is written
  - Will fail initially
  - Write just enough code to make it pass!
- Why?
  - In practice, significantly less defect rate
  - Improved understanding of requirements and ability to influence design
  - Not influenced by implementation choices



Let's try it out with this avgAbs spec

```
double avgAbs(double ... numbers)
   // Average of the absolute values of an array of doubles
```

TDD – what tests need to pass in order for us to sign off on the coding?

- `assertEquals(2.0, avgAbs({1.0, 2.0, 3.0}));`
- `assertEquals(2.0, avgAbs({1.0, -2.0, 3.0}));`
- `assertEquals(2.0, avgAbs({2.0}));`
- …

Lab t-04

- Practice TDD in lab t-04

Lab t-05

- Team up “what matters about a test suite”?

More on code coverage

code coverage: What fraction of the code is executed by the tests

- This is a metric of test suite quality

  - statement coverage - tries to execute every line (practical?)
  - path coverage - follow every distinct branch through code
  - condition coverage - every condition that leads to a branch
  - function coverage - treat every behavior / end goal separately

More on code coverage

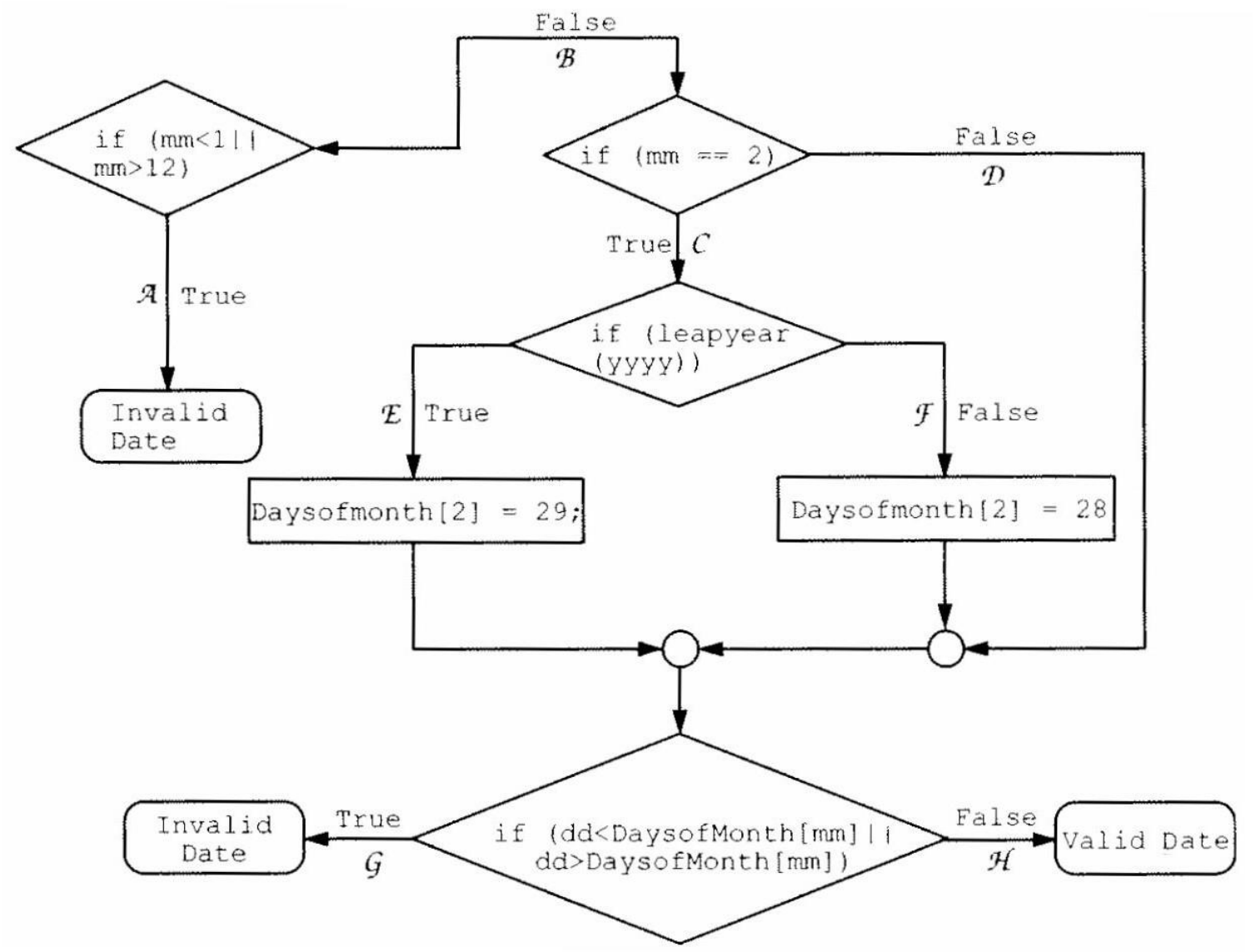
code coverage: What fraction of the code is executed by the tests

- This is a metric of test suite quality

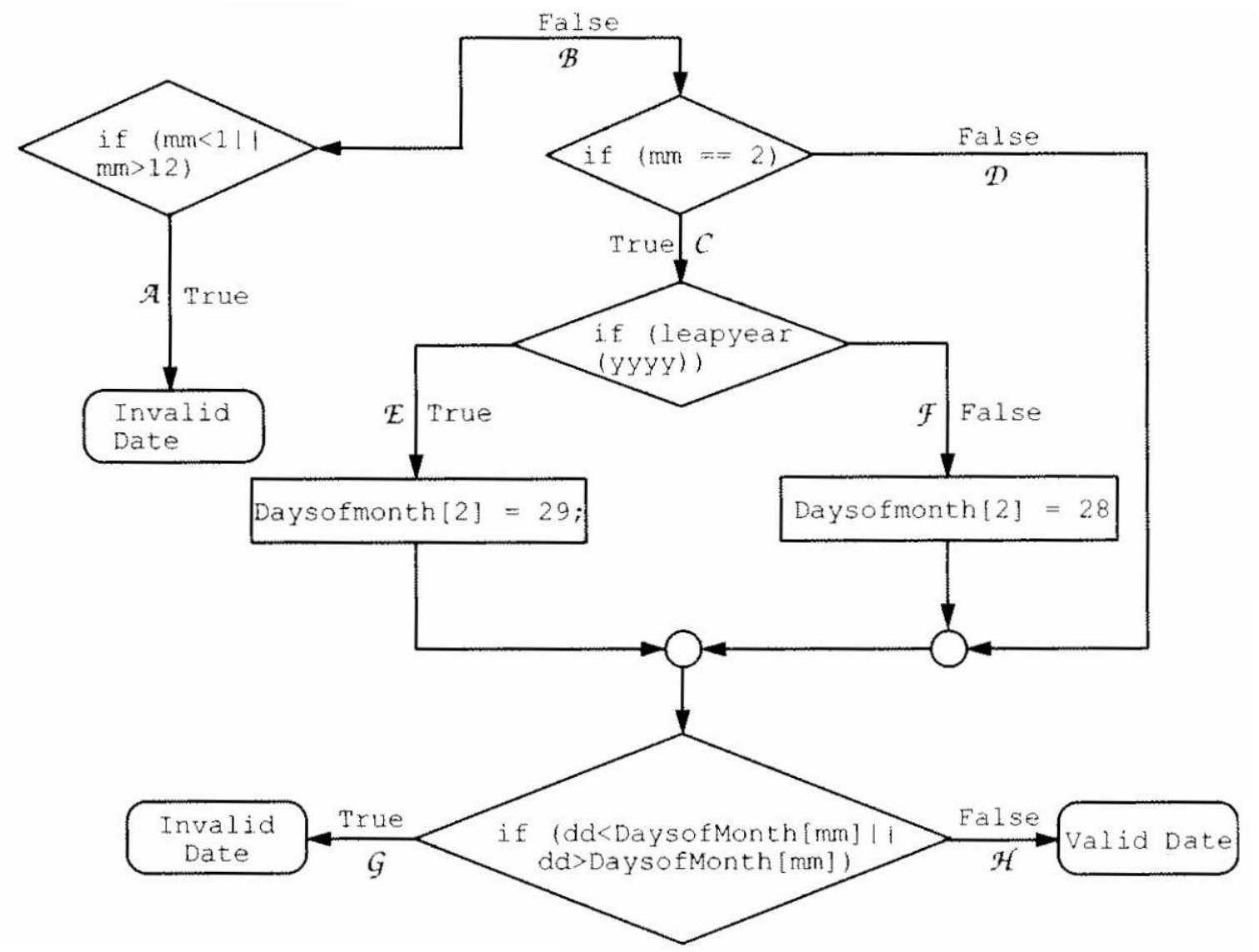
  - statement coverage - tries to execute every line (practical?)
  - path coverage - follow every distinct branch through code
  - condition coverage - every condition that leads to a branch
  - function coverage - treat every behavior / end goal separately

What about dead code?

Consider tests to cover all paths for the `Date` class



Consider tests to cover all paths for the `Date` class



You must use a code coverage tool for your project

How much coverage is enough? 100%?

May be subject to the law of diminishing returns.
Common advice: shoot for 80%.

Good resource on code coverage and code coverage tools:
https://www.atlassian.com/continuous-delivery/software-testing/code-coverage
And a good list of coverage tools:
https://www.browserstack.com/guide/code-coverage-tools

Ending with some Rules of Testing

- First rule of testing: ***Do it early and do it often***
  - Best to catch defects soon, before they have a chance to hide
  - Automate, automate, automate the process

- Second rule of testing: ***Be systematic***
  - If you randomly thrash, defects will hide until you're gone
  - Writing tests is a good way to understand the spec
    - Think about revealing domains and boundary cases
    - If the spec is confusing, write more tests
  - Spec can be defective too
    - If you find incorrect, incomplete, ambiguous, and missing corner cases, fix it!
  - When you find a defect: write a regression test, fix it, look for similar defects