Testing Process and Methods

CS 490MT/5555, Spring 2016, Yongjie Zheng

Context of Software Testing

- Verification & Validation

  - Verification: checking that the software conforms to its specification.

  - Validation: checking that the software meets the customer's expectations. – more general

- Techniques of verification and validation

  - Inspections – static

    - Automated code analysis

    - Formal verification

  - Software testing - dynamic

Definition of Software Testing

▸ Software testing is a fundamental technique used to determine errors. It <u>executes</u> the software using <u>representative data samples</u> and <u>comparing</u> the actual results with the <u>expected results</u>.

Software Test: Pass/Fail

- Given a software system *S* and a test case *t*:

  - Test case *t passes* if the actual output for an execution of *S* with *t* is the same as the expected output for *t*; otherwise *t* fails.

How to generate test cases

- **White-box testing – structural, program-based testing**
    - Test cases are designed, selected, and run based on structure of the source code (control-flow coverage, data-flow coverage).
    - Tests the nitty-gritty.
    - Drawbacks: need access to source.

- **Black-box testing – functional, specification-based testing**
    - Test cases are designed, selected, and run based on specifications.
    - Tests the overall system behavior.
    - Drawbacks: less systematic.

CS 490MT/5555 Software Methods and Tools

What to test

▸ Unit test

  ▸ Testing individual modules.

▸ Integration test

  ▸ Testing the composition of modules.

▸ System (acceptance) test

  ▸ Testing the whole system against requirements specification.

▸ Regression test

  ▸ Retesting a modified program.

V-Model

V-Model of Development & Testing
(the big picture)

CS 490MT/5555 Software Methods and Tools
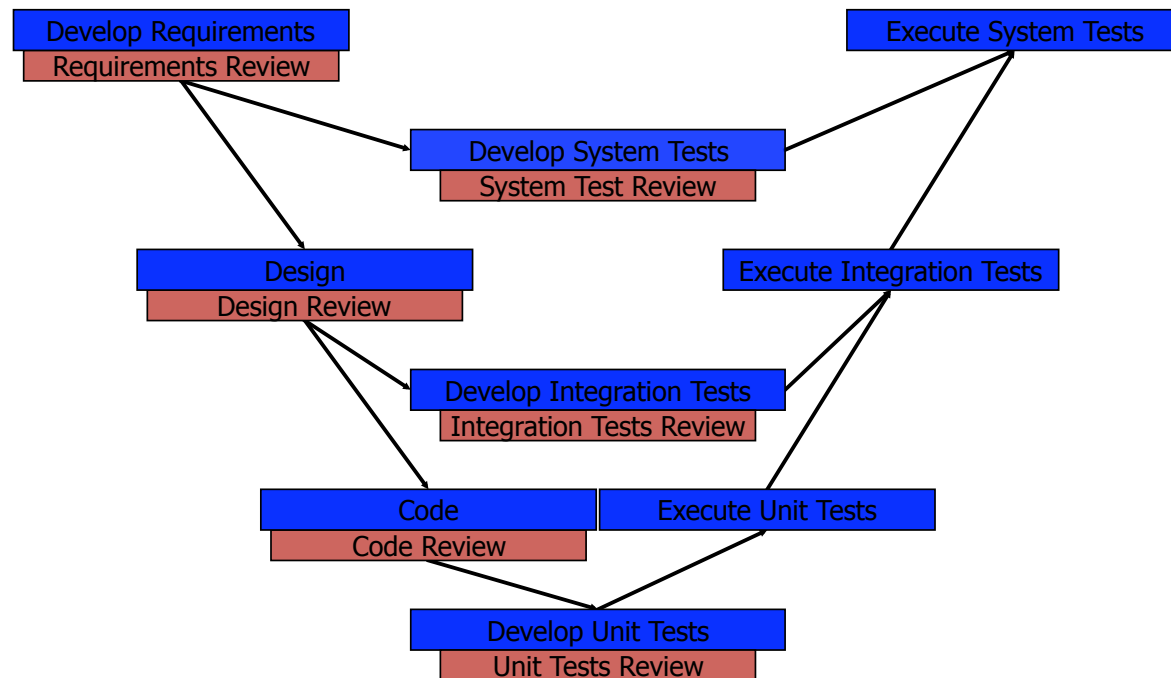
Let's get them summarized a little bit …

	Unit Test	Integration Test	System Test	Regression Test
White-box test	★	★		★
Black-box test		★	★	★

Important Activities in Software Testing

▸ **Test Data (Test Case) Generation**

    ▸ Input

        ▸ White-box: structure coverage.

        ▸ Black-box: specification based.

    ▸ Expected Output

        ▸ Test Oracle: the tester or an external mechanism that can accurately decide whether or not the output produced by a program is correct.

        ▸ Non-testable.

            ☐ No oracle: for example, to test a program that calculates the total volume of water on earth.

            ☐ Have oracle, but hard to decide: for example, to test a program that sorts an integer array whose size is over 1,000.

▸ **Localizing fault based on test results**

Testing Techniques

- ## Test Data Generation

  - Ostrand, T. J. and Balcer, M. J. 1988. The category-partition method for specifying and generating fuctional tests. Commun. ACM 31, 6 (Jun. 1988), 676-686.

- ## Fault Localization

  - Jones, J. A., Harrold, M. J., and Stasko, J. 2002. Visualization of test information to assist fault localization. In Proceedings of the 24th international Conference on Software Engineering (Orlando, Florida, May 19 - 25, 2002). ICSE '02. ACM, New York, NY, 467-477.

Fault Localization Using Visualization Technique

▸ Intuition

▸ Statements that are primarily executed by failed test cases are more suspicious than statements that are primarily executed by passed test cases.

$$suspiciousness(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}}$$

CS 490MT/5555 Software Methods and Tools

Fault Localization Using Visualization Technique

- **Input** (S: a software system; t: test case; T: test suite)
  - The source code for S
  - The pass/fail results for executing S with each t in T
  - The code coverage of executing S with each t in T
    - The code coverage for t consists of the source code statements that are executed when S is run with t.
- **Output**
  - Suspicious score for each executed source code statement
  - Visualization of the source code in which the individual statements are colored according to their participation in the testing

Fault Localization Using Visualization Technique

▸ The following five slides are provided by Prof. James A. Jones.

$$suspiciousness(s) = \cfrac{\dfrac{failed(s)}{totalfailed}}{\dfrac{passed(s)}{totalpassed} + \dfrac{failed(s)}{totalfailed}}$$

```
mid() {
  int x,y,z,m;
1:  read("Enter 3 numbers:",x,y,z);
2:  m = z;
3:  if (y<z)
4:  if (x<y)
5:     m = y;
6:  else if (x<z)
7:     m = y; // bug
8:   else
9:  if (x>y)
10:    m = y;
11:else if (x>z)
12:    m = x;
13: print("Middle number is:", m);
}
```

	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	suspiciousness
1	•	•	•	•	•	•	0.5
2	•	•	•	•	•	•	0.5
3	•	•	•	•			0.5
4							0.6
5							0.0
6							0.7
7							0.8
8				•	•		0.0
9				•	•		0.0
10							0.0
11							0.0
12							0.0
13							0.5
Pass Status	P	P	P	P	P	F	

$$susp(1) = \cfrac{\frac{1}{1}}{\frac{5}{5} + \frac{1}{1}} = 0.5$$

$$susp(7) = \cfrac{\frac{1}{1}}{\frac{1}{5} + \frac{1}{1}} = 0.8$$

13

Fault Localization Using Visualization Technique

For statement *s*:

Hue summarizes pass/fail results of test cases that executed *s*

Brightness presents the "confidence" of the hue assigned to *s*

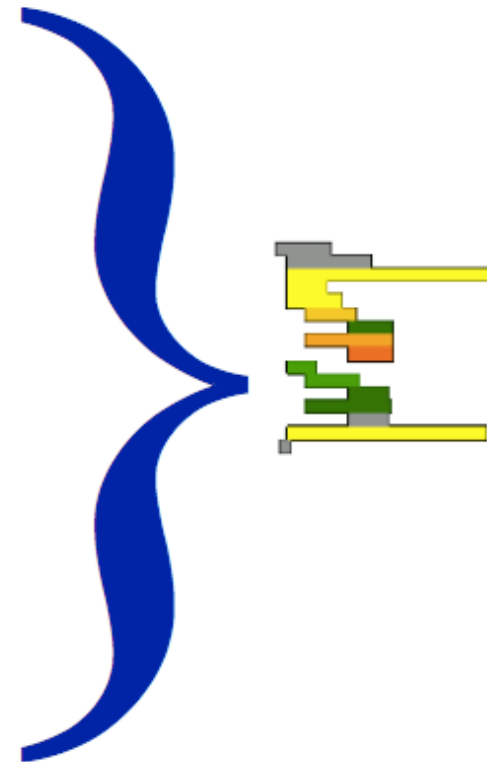# Fault Localization Using Visualization Technique

CS 490MT/5555 Software Methods and Tools

Fault Localization Using Visualization Technique

SeeSoft view

- each pixel represents a character in the source

```
mid() {
    int x,y,z,m;
    read("Enter 3 numbers:",x,y,z);
    m = z;
    if (y<z)
        if (x<y)
            m = y;
        else if (x<z)
            m = y;
    else
        if (x>y)
            m = y;
        else if (x>z)
            m = x;
    print("Middle number is:", m);
}
```

Fault Localization Using Visualization Technique

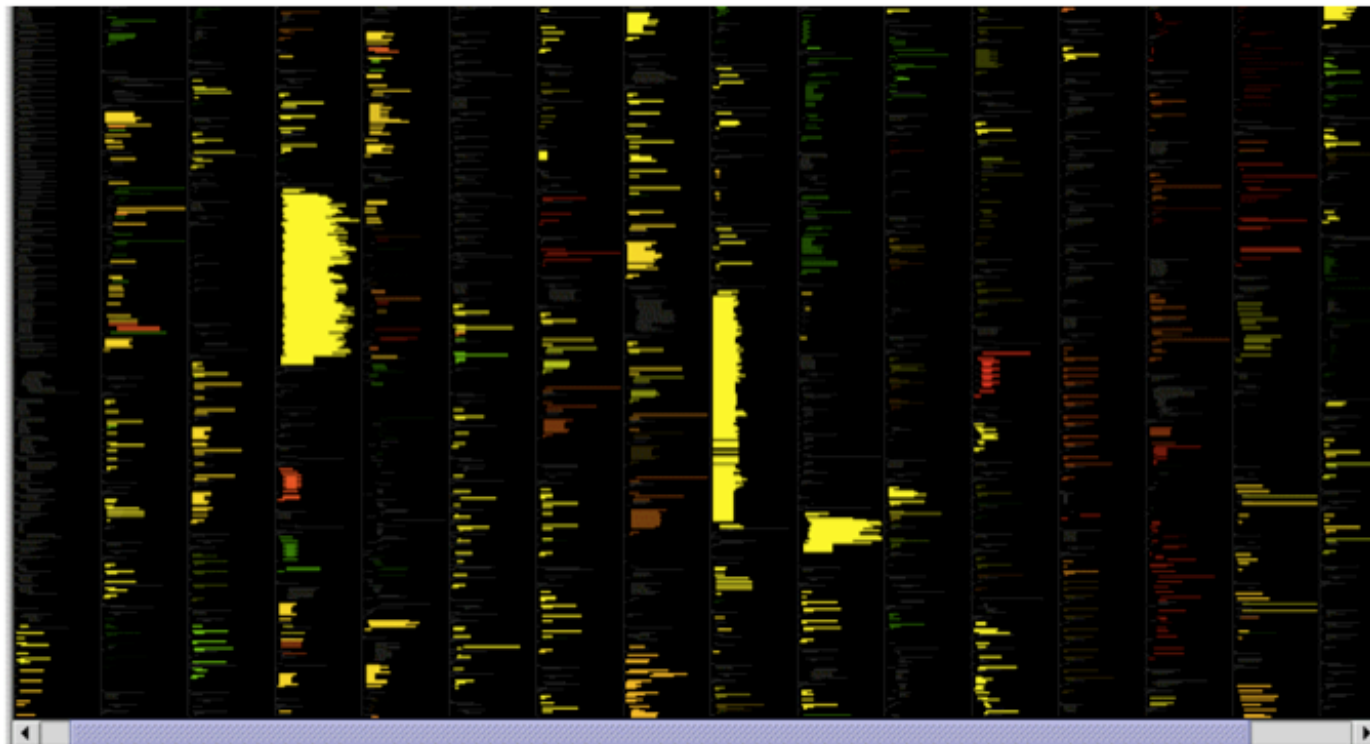**SeeSoft view**

- each pixel represents a character in the source

Category-Partition Method for Generating Functional Tests

- ▶ **Goal**
  - ▶ The tests are executed for all of the systems' functions
  - ▶ The tests are designed to maximize the chances of finding errors in the software.
    - ▶ Testing boundary conditions, special cases, error handlers, and cases where inputs and the system environment interact in potentially dangerous ways.

- ▶ **Approach**
  - ▶ Partition the input domain of a function being tested, and then select test data from each class of the partition.
  - ▶ The idea is: all elements within an equivalence class are essentially the same for the purpose of testing.

CS 490MT/5555 Software Methods and Tools

Category-Partition Method for Generating Functional Tests

▸ **Specific Steps of Category-Partition Method**

- ▸ Decompose functional specification into **functional units** that can be tested independently.

- ▸ Identifying **categories** based on parameters and environment conditions.

  - ▸ Parameters are the explicit inputs to a functional unit

  - ▸ Environment conditions are characteristics of the system's state at the time of executing a functional unit

  - ▸ A category is a major property or characteristic of a parameter or environment condition.

- ▸ Partition each category into distinct **choices**.

- ▸ Determine **constraints** among the choices.

- ▸ Generate **testing specifications**.

CS 490MT/5555 Software Methods and Tools

Category-Partition Method for Generating Functional Tests

▸ **Rule of Thumb**

  ▸ Choose test cases on the boundaries of the partitions plus cases close to the mid-point of the partition.

  ▸ While the categories are derived entirely from information in the specification, the choices can be based on

    ▸ The specification

    ▸ The tester's past experience

    ▸ Knowledge about the system

Category-Partition Method for Generating Functional Tests

▸ **Example: Search**

procedure Search (Key : ELEM ; T: ELEM_ARRAY;
    Found : **out** BOOLEAN; L: **out** ELEM_INDEX) ;

Pre-condition
    The array has at least one element
    T'FIRST <= T'LAST
Post-condition
    The element is found and is referenced by L
    (Found and T (L) = Key)

or

    The element is not in the array
    (**not** Found **and**
    not (**exists** i, T'FIRST >= i <= T'LAST, T (i) = Key))

Category-Partition Method for Generating Functional Tests

- **Category Partitions for Search**
  - Number of occurrences of key in the sequence
    - none, exactly one (assuming no duplicates in the sequence)
  - Position of key in the sequence
    - first element, middle element, last element
  - Sequence size
    - single value, many values

Category-Partition Method for Generating Functional Tests

Array	Element
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

Input sequence (T)	Key (Key)	Output (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Facts About Testing

- Testing can only show the presence of errors, not their absence.

- Automatic test cases generation is impossible.

- A major problem that arises during integration testing is localizing errors.