

SOFTWARE ENGINEERING

Week 09
Software Testing

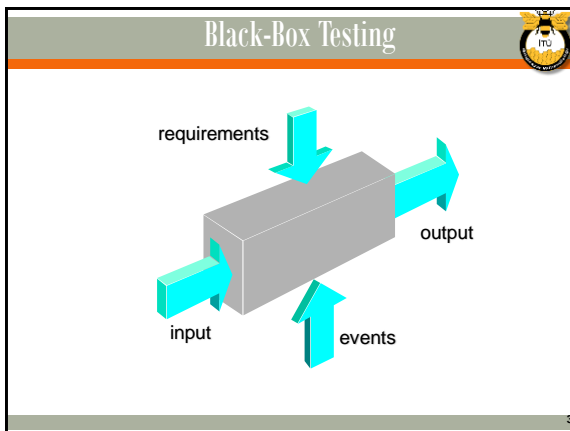
Dr. A. Cüneyd TANTUĞ
Istanbul Technical University
Computer Engineering Department

Dr. Tolga OVATMAN

1. Implementation/Programming Guidelines
2. Software Testing Concepts
3. Unit Testing
4. Module Integration and Testing Strategies
5. Testing Approaches
 1. White-Box Testing
 2. Black-Box Testing ←
 - Other Types of Testing
- 6.

Black-Box Testing

§9.5.203



Black Box Testing

- §3 Black-box testing is conducted at the software interface to demonstrate that correct inputs results in correct outputs.
- §3 Testing software against a specification of its external behavior without knowledge of internal implementation details
- §3 It is not an alternative to White Box testing, but complements it.
- §3 Focuses on the functional requirements.
- §3 Attempts to find errors on
 - Incorrect or missing functions
 - Interface errors
 - Errors in data structures or external database access
 - Performance errors
 - Initialization and termination errors

4

Examples: Input Data

Valid data:

- user supplied commands
- responses to system prompts
- file names
- computational data
 - physical parameters
 - bounding values
 - initiation values
- responses to error messages
- mouse picks

Invalid data:

- data outside bounds of the program
- physically impossible data
- proper value supplied in wrong place

5

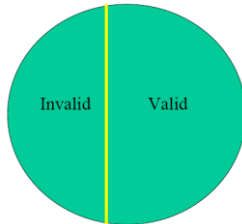
Examples of Input Domain

- §3 Boolean value
 - True or False
- §3 Numeric value in a particular range
 - $99 \leq N \leq 999$
 - Integer, Floating point
- §3 One of a fixed set of enumerated values
 - {Jan, Feb, Mar, ...}
 - {VisaCard, MasterCard, DiscoverCard, ...}
- §3 Formatted strings
 - Phone numbers
 - File names
 - URLs
 - Credit card numbers

6

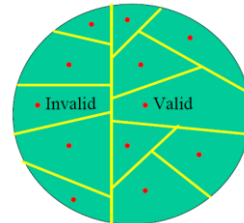
Equivalence Partitioning (1)

- First-level partitioning: Valid vs. Invalid input values



Equivalence Partitioning (2)

- Partition valid and invalid values into equivalence classes
- Create a test case for at least one value from each equivalence class



Equivalence Partitioning - Examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	?	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?	?

Equivalence Partitioning - Examples

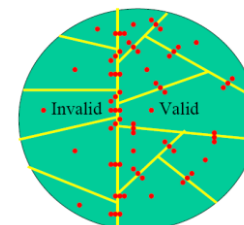
Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, ...} Non-numeric strings {junk, 1E2, \$13} Empty value
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	555-5555 (555)555-5555 555-555-5555 200 <= Area code <= 999 200 < Prefix <= 999	Area code < 200 Area code > 999 Area code with non-numeric characters <i>Similar for Prefix and Suffix</i> Invalid format 5555555, (555)(555)5555, etc.

Boundary Value Analysis (2)

- When choosing values to test, use the values that are most likely to cause the program to fail
- If $(200 < \text{areaCode} \ \&\& \ \text{areaCode} < 999)$
 - Testing area codes 200 and 999 would catch the coding error
- In addition to testing center values, we should also test boundary values
 - Right on a boundary
 - Very close to a boundary on either side

Boundary Value Analysis (3)

- Create test cases to test boundaries between equivalence classes.



Boundary Value Analysis - Examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?

13

Boundary Value Analysis - Examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	-100, -99, -98 -10, -9 -1, 0, 1 9, 10 98, 99, 100
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	Area code: 199, 200, 201 Area code: 998, 999, 1000 Prefix: 200, 199, 198 Prefix: 998, 999, 1000 Suffix: 3 digits, 5 digits

14

Example:

Function Calculating Average

15

Function Calculating Average

```
// The minimum value is excluded from average.
float Average (float scores [ ], int length)
{
    float min = 99999;
    float total = 0;
    for (int i = 0; i < length; i++)
    {
        if (scores[i] < min)
            min = scores[i];

        total += scores[i];
    }
    total = total - min;
    return total / (length - 1);
}
```

16

Test Cases (Basis: **Array Length**)

Test case (input)	Basis: Array length				Expected output	Notes
	Empty	One	Small	Large		
()	x				0.0	crashes!
(87.3)		x			87.3	
(90.95,85)			x		92.5	
(80.81,82.83,84.85,86.87,88.89,90.91)				x	86.0	

17

Test Cases (Basis: **Position of Minimum**)

Test case (input)	Basis: Position of minimum			Expected output	Notes
	First	Middle	Last		
(80.87,88.89)	x			88.0	
(87.88,80.89)		x		88.0	
(99.98,0.97,96)			x	97.5	
(87.88,89,80)			x	88.0	

18

Test Cases (Basis: Number of Minima)

Test case (input)	Basis: Number of minima			Expected output	Notes
	One	Several	All		
(80,87,88,89)	x			88.0	
(87,86,86,88)		x		87.0	
(99,98,0,97,0)		x		73.5	
(88,88,88,88)			x	88.0	

19

Example:

Function Normalizing an Array

20

Function Normalizing an Array

- ⇒ The following C function is intended to normalize an array. (It can be used to get black/white of an image.)
- ⇒ Normalization Algorithm:
 - First, the array of N elements is searched for the smallest and largest non-negative elements.
 - Secondly, the range is calculated as max - min.
 - Finally, all non-negative elements that are bigger than the range are normalized to 1, or else to 0.
- ⇒ There are no syntax errors, but there may be logic errors in the following implementation.

21

C function

```

1. int normalize (int A[], int N)
2. {
3.     int range, max, min, i, valid;
4.     range = 0;
5.     max = -1;
6.     min = -1;
7.     valid = 0;
8.     for (i = 0; i < N; i++)
9.     {
10.        if (A[i] >= 0)
11.        {
12.            if (A[i] > max)
13.                max = A[i];
14.            else if (A[i] < min)
15.                min = A[i];
16.            valid++;
17.        }
18.    }
19.     range = max - min;
20.     for (i = 0; i < N; i++)
21.     {
22.         if (A[i] >= 0)
23.             if (A[i] >= range)
24.                 A[i] = 1;
25.             else A[i] = 0;
26.     }
27.     return valid;
28. }

```

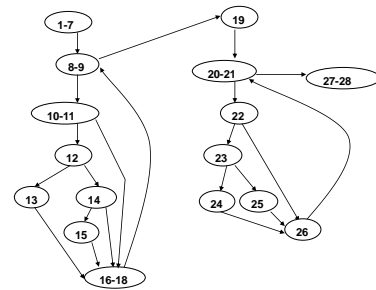
22

Tasks

- ⇒ Draw the corresponding flow graph and calculate the cyclomatic complexity V(g).
- ⇒ Find linearly independent paths for basis path testing.
- ⇒ Give test cases for boundary value analysis.

23

Flow Graph



24

Cyclomatic Complexity and Test Paths

Number of edges = $E = 21$
 Number of nodes = $N = 16$
 Cyclomatic complexity = $V(g) = E - N + 2 = 21 - 16 + 2 = 7$

Path1: 1-7, 8-9, 10-11, 12, 13, 16-18, 19, 20-21, 22, 23, 24, 26, 27-28
 Path2: 1-7, 8-9, 10-11, 12, 14, 15, 16-18, 19, 20-21, 22, 23, 24, 26, 27-28
 Path3: 1-7, 8-9, 10-11, 12, 14, 16-18, 19, 20-21, 22, 23, 24, 26, 27-28
 Path4: 1-7, 8-9, 19, 20-21, 22, 23, 24, 26, 27-28
 Path5: 1-7, 8-9, 10-11, 12, 13, 16-18, 19, 20-21, 22, 23, 25, 26, 27-28
 Path6: 1-7, 8-9, 10-11, 12, 13, 16-18, 19, 20-21, 22, 26, 27-28
 Path7: 1-7, 8-9, 10-11, 12, 13, 16-18, 19, 20-21, 27-28

25

Test cases for boundary value analysis

Test Case-1)

Input Condition: $N=0$, $A[] = \{10, 20, 30, 40, 50\}$
 Expected Output: Valid = 0 (Due to invalid N value)

Test Case-2)

Input Condition: $N=4$, $A[] = \{10, 20, 30, 40, 50\}$
 Expected Output: Valid = 4 $A[] = \{0, 0, 1, 1, 50\}$

Test Case-3)

Input Condition: $N=5$, $A[] = \{10, 20, 30, 40, 50\}$
 Expected Output: Valid = 5, $A[] = \{0, 0, 0, 1, 1\}$

26

Test cases for boundary value analysis

Test Case-4)

Input Condition: $N=5$, $A[] = \{-10, -20, -30, -40, -50\}$
 Expected Output: Valid = 0

Test Case-5)

Input Condition: $N=5$, $A[] = \{0, 0, 0, 0, 0\}$
 Expected Output: Valid = 5, $A[] = \{1, 1, 1, 1, 1\}$

Test Case-6)

Input Condition: $N=5$, $A[] = \{10, 10, 10, 10, 10\}$
 Expected Output: Valid = 5, $A[] = \{1, 1, 1, 1, 1\}$

27

1. Implementation/Programming Guidelines
2. Software Testing Concepts
3. Unit Testing
4. Module Integration and Testing Strategies
5. Testing Approaches
 1. White-Box Testing
 2. Black-Box Testing
6. Other Types of Testing

Other Types of Testing

9.6

Special Tests

- Regression testing
- Alpha test and beta test
- Performance testing
- Volume testing
- Stress testing
- Security testing
- Usability testing
- Recovery testing
- Testing web-based systems

29

System Testing

1. Functional Testing
2. Performance Testing
3. Acceptance Testing
4. Installation Testing

30

Functional Testing

- ✎ An alternative form of black-box testing for classical software
 - We base the test data on the functionality of the code artifacts
- ✎ Each item of functionality or function is identified
- ✎ Test data are devised to test each (lower-level) function separately
- ✎ Then, higher-level functions composed of these lower-level functions are tested

Integration Testing

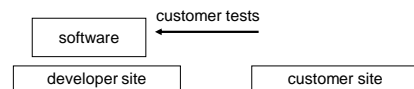
- ✎ The testing of each new code artifact when it is added to what has already been tested
- ✎ Special issues can arise when testing graphical user interfaces — see next slide

Regression Testing

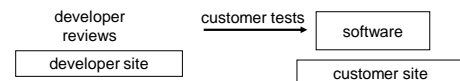
- Each time a new module is added as part of integration testing, the software changes and needs to re-tested.
- Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
- In broader context, regression testing ensures that changes (enhancement or correction) do not introduce unintended new errors.
- When used?
 1. Integration testing
 2. Testing after module modification

Alpha Test & Beta Test

Alpha Test



Beta Test



Performance Testing

- ✎ Measure the system's performance
 - Running times of various tasks
 - Memory usage, including memory leaks
 - Network usage (Does it consume too much bandwidth? Does it open too many connections?)
 - Disk usage (Does it clean up temporary files properly?)
 - Process/thread priorities (Does it run well with other applications, or does it block the whole machine?)

Volume Testing

- ✎ Volume testing: System behavior when handling large amounts of data (e.g. large files)
 - Test what happens if large amounts of data are handled
- ✎ Load testing: System behavior under increasing loads (e.g. number of users)
- ✎ Test the system at the limits of normal use
 - Test every limit on the program's behavior defined in the requirements
 - Maximum number of concurrent users or connections
 - Maximum number of open files
 - Maximum request or file size

Stress Testing

- ⇒ Stress testing: System behavior when it is overloaded
- ⇒ Test the limits of system (maximum # of users, peak demands, extended operation hours)
- ⇒ Test the system under extreme conditions
 - Create test cases that demand resources in abnormal quantity, frequency, or volume
 - Low memory
 - Disk faults (read/write failures, full disk, file corruption, etc.)
 - Network faults
 - Power failure
 - Unusually high number of requests
 - Unusually large requests or files
 - Unusually high data rates
- ⇒ Even if the system doesn't need to work in such extreme conditions, stress testing is an excellent way to find bugs

37

Security Testing

- ⇒ Try to violate security requirements
- ⇒ Any system that manages sensitive information or performs sensitive functions may become a target for illegal intrusion
- ⇒ How easy is it to break into the system?
- ⇒ Password usage
- ⇒ Security against unauthorized access
- ⇒ Protection of data
- ⇒ Encryption

38

Testing Web-based Systems

Concerns of Web-based Systems:

- Browser compatibility
- Functional correctness
- Usability
- Security
- Reliability
- Performance
- Recoverability

39

Acceptance Testing

- ⇒ The client determines whether the product satisfies its specifications
- ⇒ Acceptance testing is performed by
 - The client organization, or
 - The SQA team in the presence of client representatives, or
 - An independent SQA team hired by the client
- ⇒ The key difference between product testing and acceptance testing is
 - Acceptance testing is performed on actual data
 - Product testing is performed on test data, which can never be real, by definition

Examples of Test Automation Tools

Vendor	Tool
Hewlett Packard	QuickTest
IBM	Rational Functional Tester
Selenium	Selenium
Microfocus	SilkTest
AutomatedQA	TestComplete

41

Wrap-up


This week we present

- ⇒ Good Software Implementation Principles
 - Variable naming, commenting, debugging, etc.
- ⇒ Low Level Testing
 - Applying Unit Testing
 - The concept of drivers and stubs
 - How to integrate the system at overall
- ⇒ Testing Strategies and Approaches
 - White-box testing and black-box testing
 - Input domain partitioning
 - Higher Level Testing

Introduction & UML

1.42

Next Week



☞ We will be covering *Software and Design Quality Concepts!!!*

Introduction & LMC

1.43