



JOHNS HOPKINS
UNIVERSITY

EN.601.422 / EN.601.622

Software Testing & Debugging

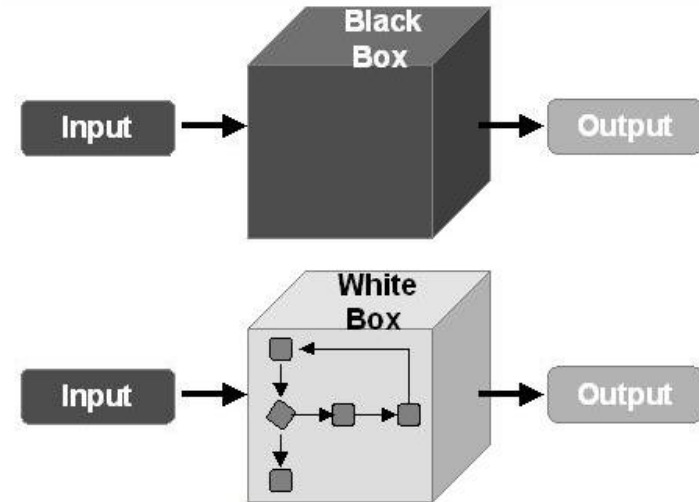
The material in this video is subject to the copyright of the owners of the material and is being provided for educational purposes under rules of fair use for registered students in this course only. No additional copies of the copyrighted work may be made or distributed.

Plan for today

- ▶ Blackbox vs Whitebox testing
- ▶ Blackbox testing techniques:
 - ❖ Partitioning of input/output space into equivalence classes
 - ❖ Boundary Analysis
 - ❖ Error Guessing

Blackbox and Whitebox Testing

- ▶ Blackbox testing views the software as a black box. The goal is to concentrate on the “software specifications”
 - ❖ also known as data-driven, io-driven, or specification-based testing
- ▶ Whitebox testing is concerned with the degree to which test cases cover the source code of the software
 - ❖ also known as Glassbox or logic-driven testing



Greybox Testing

- ▶ When there is only partial access/understanding of the internal structure of the software under test (SUT)
 - ❖ you know the algorithm, but not the exact implementation
 - ❖ you know the design or structure of the code, but not the exact implementation
 - ❖ Etc.
- ▶ *We do not examine this further in the class*

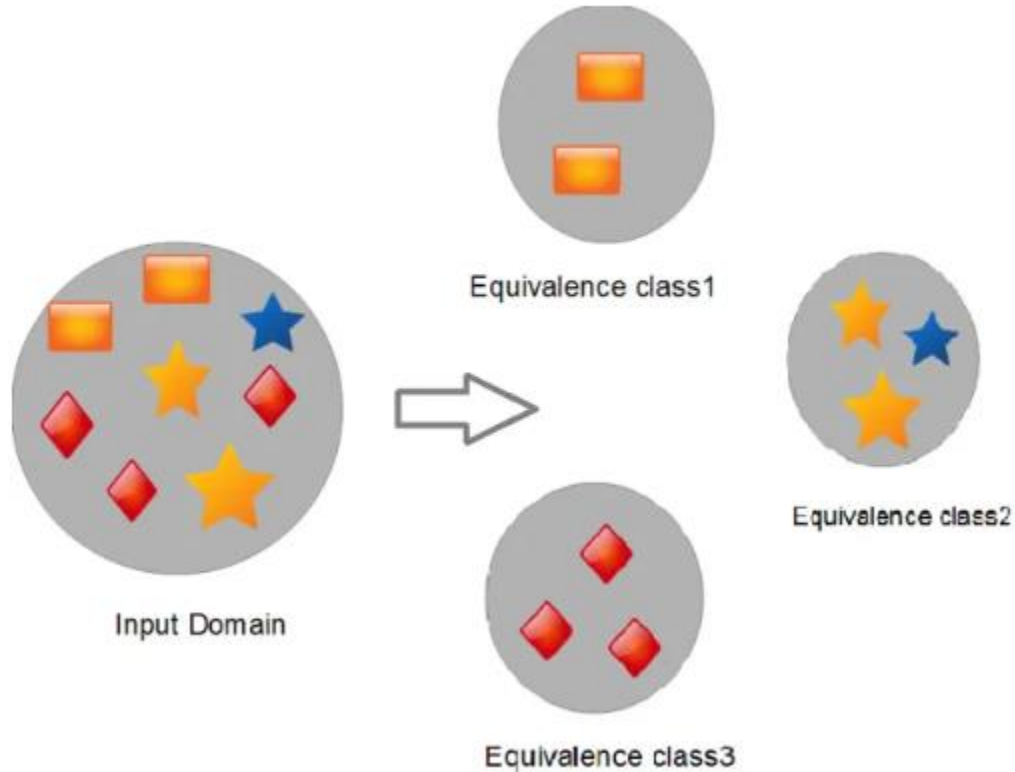
Blackbox and Whitebox Testing

- ▶ Blackbox testing:
 - ❖ test cases drawn solely from the specifications (e.g., formal specifications, API docs, user manual etc.)
 - ❖ exhaustive Blackbox testing is to try all possible inputs
- ▶ Whitebox testing:
 - ❖ test cases drawn by looking at (and manipulating) source code
 - ❖ exhaustive Whitebox testing is to try all execution paths

Equivalence Class

- ▶ A subset of the form $\{x \in X: x R a\}$, where a is an element of X and the notation " $x R y$ " is used to mean that there is an equivalence relation between x and y
- ▶ In other words:
 - ❖ An **equivalence class (or equivalence block)** is the name that we give to the subset of S which includes **all elements that are equivalent to each other**. "Equivalent" is dependent on a specified relationship (i.e., characteristic), called an *equivalence relation or characteristic*. If there's an equivalence relation between any two elements, they're called equivalent.

Equivalence Class



Equivalence Class Examples

- ▶ **Example 1:** X is the set of all cars. \sim is the equivalence relation "*has the same color as*", then equivalence classes consist of cars of different colors. e.g., set of all red cars, set of all blue cars, etc.
- ▶ **Example 2:** I is the set of all integer values. \sim is the equivalence relation "*has the same sign as*", then equivalence classes consist of 1) set of all negative integers, 2) zero, and 3) set of all positive integers
- ▶ **Example 3:** A is the set of all Mattresses. \sim is the equivalence relation "*has the same size*", then the equivalence classes consist of sets of all mattresses of the same size (i.e., Crib, Twin, Twin XL, Full, Queen, King, Cal King)

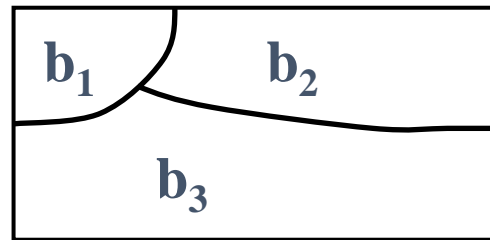
Partitioning domain into equivalence classes

- ▶ 1. Partition the input/output domain into a set of equivalence classes
- ▶ 2. Produce a representative concrete test case for each equivalence class
- ▶ The idea is:
 - ❖ if a test case from an equivalence class detects an error/failure, so does all the other test cases of the same equivalence class
 - ❖ conversely, if a test case from an equivalence class does not detect an error/failure, no other test cases of the same equivalence class does

Equivalence Partitioning

- ▶ Given characteristic i.e., relation C , the partition q defines a set of blocks(i.e., equivalence classes) over Domain D :

$$Bq = b_1, b_2, \dots, b_q$$



- ▶ Two important properties for selecting equivalence classes correctly:

- ❖ **disjointedness:** Blocks (i.e., classes) must be pairwise disjoint; that is no two blocks overlap

$$b_i \cap b_j = \Phi, \forall i \neq j, b_i, b_j \in B_q$$

- ❖ **completeness:** Together the blocks cover entirety of domain D

$$\bigcup_{b \in B_q} b = D$$

Relation (i.e. characteristic)

- ▶ Each partition is built based on a characteristic C :
- ▶ Examples:
 - ❖ Object 'a' is null → two classes namely null and non-null
 - ❖ Input device type → multiple classes namely DVD, CD, VCR ...
 - ❖ Shirt Size → multiple size classes namely xs, s, m, l, xl, xxl ...
 - ❖ etc.

Example

- ▶ Input: File f
- ▶ Characteristic: Order of file f
 - ❖ b_1 = sorted in ascending order
 - ❖ b_2 = sorted in descending order
 - ❖ b_3 = not sorted in any specific order

Any issues?

Steps to Equivalence Partitioning

- ▶ Design the characteristics to create partition(s) over the input/output domain
 - ❖ ****It is possible to design characteristics based on *output*****
- ▶ Decide on the blocks (i.e., equivalence classes) for each partitioning/characteristic
- ▶ Derive representative values for each block

Triangle Example

```
/**  
 * decides the type of the triangle given the lengths of the three sides  
 * @param a first length  
 * @param b second length  
 * @param c third length  
 * @return an int indicating the type of the triangle: 0 is invalid, 1 is scalene, 2 is isosceles, and 3 is equilateral  
 */  
public static int triangleType(int a, int b, int c)
```

Assume we do a partitioning over the **output** domain using characteristic “Geometric Classification”. From this, we derive four classes: 1) scalene, 2) isosceles, 3) equilateral, and 4) invalid.

Is the above a valid partitioning over
the output domain?

Triangle Example

► Technically, an equilateral is isosceles by definition!

- 1) Scalene
- 2) Isosceles but not equilateral,
- 3) equilateral,
- 4) invalid.

This is better!

Identifying Equivalence Classes

- ▶ Typically produced from specifications
 - ❖ take sentences/phrases about the input/output and apply partitioning based on the specified conditions to produce the equivalence classes
 - ❖ produce both valid and invalid equivalence classes

Example: *“the count should range from 1 to 999 inclusive”*

one valid equivalence class: $1 \leq \text{count} \leq 999 \rightarrow$ test case: 230

two invalid equivalence classes: $\text{count} > 999$ and $\text{count} < 1 \rightarrow$ test case: -1 & 1020

- ▶ If an input specifies a “must-be” situation, produce one valid and one invalid equivalence class

Example: *“the first character of the string must be a digit”*

one valid equivalence class: **the string starts with a digit** \rightarrow “1s2”

one invalid equivalence class: **the string does not start with a digit** \rightarrow “%h”

Boundary Value Analysis

- ▶ Test conditions on bounds between equivalence classes
- ▶ Rationale:
 - ❖ likely source of programmer errors ($<$ vs. \leq , etc.)
 - ❖ Software specifications may be fuzzy/vague about behavior on boundaries
 - ❖ often uncovers internal hidden limits in code
 - Example:

Specs: array must be sized no less than 1 and no larger than 10



- Three equivalence blocks: $\text{size} < 1$, $1 \leq \text{size} \leq 10$, $\text{size} > 10$
- try array sizes 0, 1, 10, and 11
(also, try MAX_INT , $\text{MAX_INT} + 1$, $\text{MIN_INT} - 1$, MIN_INT)

Boundary Value Analysis

- ▶ **Example 1:** input condition specifies the valid domain of an input value is between -1 and 1.0 → write test cases with values -1.0, 1.0, -1.001, and 1.001
- ▶ **Example 2:** input condition specifies an input file can contain 1 - 255 records → write test cases for files with 0, 1, 255, 256 records
- ▶ **Example 3:** output condition specifies payroll software computes the monthly FICA deduction of minimum \$0.00 and the maximum of \$1,165.25 → try to write/invent test cases that might cause a negative deduction or a deduction of more than \$1,165.25

Error Guessing

- ▶ No systematic way
- ▶ Use your intuition/experience trying to cause errors/failures in the system
 - ❖ try different error-prone situations
 - ❖ **Examples:** zero for int values, null for objects, invalid inputs, out of bound inputs, empty sets/lists, sets/lists with one entry, inputs based on holes in the specifications, negative inputs where they are not relevant
- ▶ Complementary to other testing techniques

Relevant Reads

- ▶ Recommended Textbooks:

- ❖ Intro to Software Testing (ch1, ch2)
- ❖ The Art of Software Testing (ch1, ch2, ch4)



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING