

TESTING (PART B)

Chapter 8 Block b

Software Engineering
Computer Science School
DSIC - UPV

DOCENCIA VIRTUAL

Finalidad:
Prestación del servicio Público de educación superior (art. 1 LOU)

Responsable:
Universitat Politècnica de València.

Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento conforme a políticas de privacidad:
<http://www.upv.es/contenidos/DPD/>

Propiedad intelectual:
Uso exclusivo en el entorno de aula virtual.
Queda prohibida la difusión, distribución o divulgación de la grabación de las clases y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes.
La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa o civil

 **UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA**



Black-box testing

- Black-box methods are focused on the functional requirements of software.
- Black box tests try to find errors of the following types:
 - Incorrect or non-existent functions.
 - Interface related errors.
 - Errors related to data structures or in external databases.
 - Performance related errors.
 - Initialization and termination errors.

Equivalence Partitioning

- Range conditions: normally have
 - 1 correct class
 - 2 incorrect class

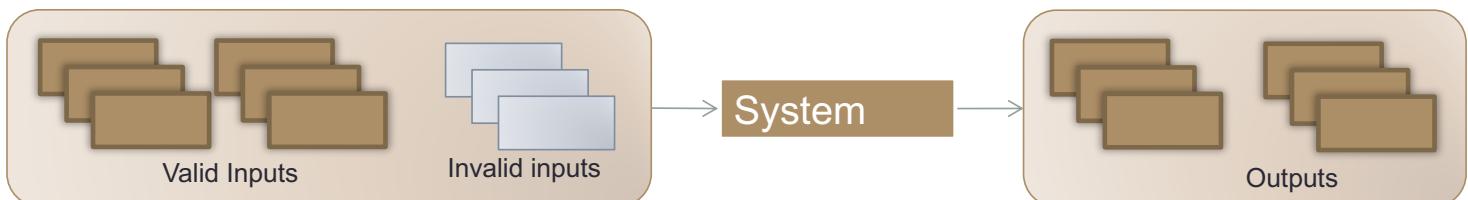
- Input conditions** of the Software under test:

format or content restrictions of the input data

- foo (int x)*
- Valid data $\longrightarrow C_1 = \{x \in \text{int} \mid x \in [20, 30]\}$ *range condition*
 - Invalid data $\longrightarrow C_2 = \{x \in \text{int} \mid x < 20\}$ $C_3 = \{x \in \text{int} \mid x > 30\}$

- Equivalence classes:**

- A test with a representative value of a class assumes that the obtained results (failure or success) will be the same for any other arbitrary member of the category



Equivalence Partitioning

- **Technique for the identification of test cases:**
 1. For each input condition of the software under test, we will identify its equivalence classes using heuristics.
 2. A unique number will be assigned to each identified equivalence class.
 3. Until all valid equivalence classes are covered by some test case. A test case covering as many as possible valid classes will be designed.
 4. Until all invalid equivalence classes have been covered. There will be a separate test case per invalid class to be covered.

The previous process may also be applied to the output conditions of the software under test

Equivalence Partitioning

- **Identification Heuristics of equivalence classes:**

- a) If a **range of values** is specified for the input data, a **valid class** and two **invalid classes** will be created *! Check the type of value*
- b) If a **finite value** is specified a **valid class** and two **invalid classes** will be created
- c) If a condition of type **«it must be»** or a **boolean condition** is specified (e.g. “ the first character must be a character from the alphabet», a **valid class** «it is an alphabet character» and another **invalid class** «it is not an alphabet character» will be created
- d) if a **set of valid accepted values** and the program treats them in a different way, then a **valid class** for each **value** and another **invalid class** must be created
- e) If it is suspected that **some concrete elements** of a class are handled in a different way, the class must be **partitioned into reduced classes**

Exercise: Equivalent Partitioning

- Banking application. Input data (text file):
 - **Area code:** number with 3 digits not starting by 0 nor 1
 - **Operation Name:** 6 characters
 - **Valid Commands:** “check”, “deposit”, “bill payment”, “withdrawal”

Exercise: Equivalence Partitioning

Area Code:

Boolean:

1 valid class: it is a number

Range:

1 valid class: $200 < \text{area code} < 999$

2 invalid classes: $\text{area code} < 200$; $\text{area code} > 999$

1 invalid class: not a number

Operation Name:

Finite Value:

1 valid class: 6 characters

2 invalid classes: > 6 characters; < 6 characters

Valid commands:

Set of values:

4 valid classes: 4 valid commands

1 invalid class: invalid command

Exercise: Equivalence Partitioning

Input data	Valid classes	Invalid classes
Area code	(1) $200 \leq \text{code} \leq 999$	(2) $\text{code} < 200$ (3) $\text{code} > 999$ (4) not a number
Operation name	(5) 6 characters	(6) < 6 characters (7) > 6 characters
Command	(8) “check” (9) “deposit” (10) “bill payment” (11) “withdrawal”	(12) invalid command

Exercise: Equivalence Partitioning Valid Test Cases

Code	Operation Name	Command	Covered classes
300	Nómina	“Deposit”	(1) ^C (5) ^C (9) ^C
400	Viajes	“Check”	(1) (5) (8) ^C
500	Coches	“Bill payment”	(1) (5) (10) ^C
600	Comida	“Withdrawal”	(1) (5) (11) ^C

Exercise: Equivalence partitioning Invalid Test cases

Code	Operation Name	Command	Covered Classes
180	Viajes	“Bill payment”	(2) ^C (5) (10)
1032	Nómina	“Deposit”	(3) ^C (5) (9)
XY	Compra	“Withdrawal”	(4) ^C (5) (11)
350	A	“Deposit”	(1) (6) ^C (9)
450	Regalos	“Check”	(1) (7) ^C (8)
550	Casita	&%4	(1) (5) (12) ^C



Analysis of Boundary values

Analise the values of the boundaries for the correct cases and the values closest to the boundaries but out of it for the incorrect ones

- This technique selects as values for the test cases those that are boundary values (errors often occur at the boundaries).
- It complements the equivalent partitioning technique. Instead of selecting an arbitrary value from an equivalence class we choose a boundary value.
- Test cases may be derived for both input conditions and output conditions.

Analysis of Boundary values

- Heuristics:
 - If an input condition specifies a range of values limited by a and b, test cases must be defined using the a and b values and other values just below and above them.
 - If an input condition specifies a number of values, test cases must be define to consider the minimum and maximum values and also those just below and above them.

Analysis of Boundary Values

- Heuristics:
 - Apply the previous heuristics also to output conditions. Define test cases that produce values in the boundaries.
 - If we use data structures with boundaries (e.g., an array of 10 elements), a test case must be defined so that the data structure is tested at its boundaries.

Other types of tests

- Walkthroughs.
- Robustness testing
- Stress testing
- Performance testing
- Conformance testing
- Interoperability testing

Automatic Testing tools

- **Static Analyzers.** These systems allow the testing of those statements considered as weak within a program.
- **Code auditing.** Filters defined to verify that the code complies with different quality criteria (usually strongly dependent on the programming language).
- **Test files generators.** These tools automatically generate files with data that will serve as input for programs.
- **Test Data Generators.** These systems generate concrete input data to drive the program to a concrete behavior.
- **Test controllers.** Generate and feed the input data and simulate the behavior of other modules to restrict the scope of the test.

Automatic Testing tools

- Static Analyzers:
 - Data bugs: variables being used before they are initialized, variables defined but never used, variables not used between to assignment statements, violations of the size of an array, etc.
 - Control bugs: code fragments that are never reached.
 - Input/Output bugs: an output variable is returned without modifying its value.
 - Interface bugs: types or incorrect number of parameters, not using the result value of a function, never called functions.
 - Bugs related to pointers management.

Equivalence Partitioning: Exercise

A program takes as input a file with the following record format (fields):

- Num-employee is a field of positive integers less than 1000 and excluding 0.
- Name-employee is an alphanumeric field with 10 characters.
- Months-Work is a field indicating the number of months an employee is working; it is a positive integer less than 1000 and including 0.
- Manager is a field with just one character that may be «+» to indicate that an employee is a manager and «-» otherwise.

The program assigns a bonus to each employee and prints the list of assigned bonus following these rules:

- B1 for manager with at least 12 months of work experience
- B2 for non-managers with at least 12 months of experience
- B3 for manager with less than 12 months of experience
- B4 for non managers with less than 12 months of working experience

Obtain:

- a) A table of numbered equivalence classes with the following columns: 1 Input Condition under consideration; 2 Valid Classes and 3 Invalid classes 4 Heuristic rule that is applied
- b) The associated test cases

Basic Path \Rightarrow white box

Equivalence Partitioning \Rightarrow Black box

Input Condition
under evaluation

Valid classes

Invalid classes

Heuristic rule that is applied

Num-employee

(1) $0 < \text{num-employee} < 1000$

(2) $\text{num-employee} \geq 1000$
(3) $\text{num-employee} \leq 0$

Range

Name_employee

(4) $\text{name-employee.length} \leq 10$

(5) < 10 characters
(6) > 10 characters
(7) non-alphanumeric

Finite value

Month-Work

[0-11]

[12-999]

< 0
 > 999
Nota numbr

Range
Reduced classes
Break

Manager

+

-

Another character

Set of values

If we are
told that the
data is obtained
from file

↓ ALL

CHECK TYPES

If data is passed
as parameter
↓
NO NEED

T.C. Valid C.	Valid Classes	Input	Output
1 - 5 - 8 - 13	123, gumersindo, 9, +	B4	
1 - 5 - 9 - 14	456, sebastiano, 13, -	B2	

- File must be present (Valid Category: file present, invalid category: file not present)

- Name-Employee must be alphanumeric (Valid Category: is alphanumeric, invalid category: is not alphanumeric)

Test Cases	Invalid Cl.	Input	Outp.
Invalid Classes	2 - 5 - 9 - 13	0, gumersindo, 14, +	
	3 - 5 - 9 - 14	1024, minotauros, 16, -	
	4 - 5 - 8 - 13	abc, sebastiano, 8, +	
	1 - 6 - 8 - 13	123, cobos, 6, +	
	1 - 7 - 8 - 13	123, torceballos, 3, +	
	1 - 5 - 10 - 13	123, margaritos, -1, +	
	1 - 5 - 11 - 14	123, margaritos, 1024, -	
	1 - 5 - 12 - 14	123, margaritos, abc, -	
	1 - 5 - 9 - 15	123, margaritos, 13, *	

