

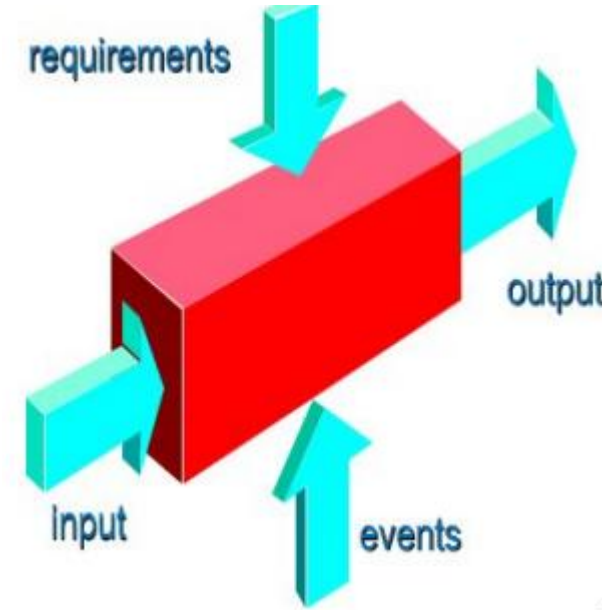
Software Testing

Functional (Black Box) Testing

May 29, 2020

Black Box Testing

- In black box testing items are treated as black whose logic is unknown
- All that is known is what's goes in and what comes out. the input and the output



Focuses on the functional requirement of the software also called behavioral testing

Black Box Testing

- Does not need any knowledge of internal design or code
- It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
- Black-box testing is not an alternative to white-box techniques but it is complementary approach.
- Tester is needed to be thorough with the requirement specifications of the system.
- User should know how the system should behave in response to the particular action

Black Box Testing

Black-box testing attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external data base access.
- Behavior or performance errors
- Initialization and termination errors.

Commonly used Black Box methods :

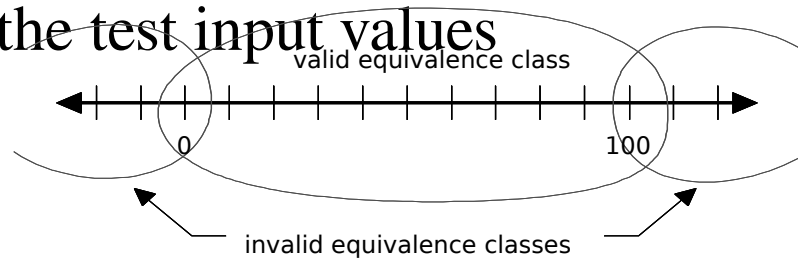
1. Equivalence partitioning
2. Boundary-value analysis
3. Error guessing

Equivalence Partitioning

Equivalence testing is a black-box testing method that divides the space of all possible inputs into equivalence groups such that the program “behaves the same” on each group

Two steps:

1. partitioning the values of input parameters into equivalence groups
2. choosing the test input values



Equivalence Partitioning

1. *“If an input condition for the software-under-test is specified as a **range of values**, select one valid equivalence class that covers the allowed range and two invalid equivalence classes, one outside each end of the range.”*

For example, suppose the specification for a module says that an input, the length of a Wire in millimeters, lies in the range 1–499; then select

One valid equivalence class that includes all values from 1 to 499.

Two invalid classes

That consists of all values less than 1, and

That consists of all values greater than 499.

Equivalence Partitioning

2. *“If an input condition for the software-under-test is specified as a **number of values**, then **select one valid equivalence class that includes the allowed number of values and two invalid equivalence classes that are outside each end of the allowed number.**”*

- For example, if the specification for a real estate-related module says that a house can have one to four owners, then we select
One valid equivalence class that includes all the valid number of owners
Two invalid equivalence classes ◦ Less than one owner and More than four owners.

Equivalence Partitioning

3. *“If an input condition for the software-under-test is specified as a **set of valid input values**, then **select one valid equivalence class that contains all the members of the set and one invalid equivalence class for any value***

- *outside the set.”*
- For example, if the specification for a paint module states that the colors RED, BLUE, GREEN and YELLOW are allowed
 - as inputs, then select
 - One valid equivalence class** that includes the set RED,BLUE, GREEN and YELLOW, and
 - One invalid equivalence class** for all other inputs.

Equivalence Partitioning

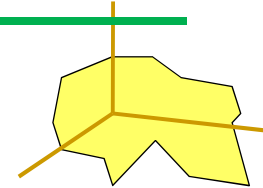
4. *“If an input condition for the software-under-test is specified as a **“must be” condition**, select one valid equivalence class to represent the “must be” condition and one invalid class that does not include the “must be” condition.”*

For example, if the specification for a module states that the first character of a part identifier must be a letter, then select

- **One valid equivalence class** where the first character is a letter and
- **One invalid class** where the first character is not a letter.

Boundary Value Analysis

- “Bugs lurk in corners and congregate at boundaries...”



- The test cases developed based on equivalence class partitioning can be strengthened by use of a technique called boundary value analysis. **Many defects occur directly on, and above and below, the edges of equivalence classes.**

- BVA is a software design technique to determine test cases off-by-one errors

Boundary Value Analysis

*1. If an input condition for the software-under-test is specified as a **range of values**, develop **valid test cases for the ends of the range**, and **invalid test cases for possibilities just above and below the ends of the range**.*

- For example if a specification states that an input value for a module must lie in the range between -1.0 and +1.0, **Valid tests** that include values -1.0, +1.0 for ends of the range, as well as
- **Invalid test cases** for values just above and below the ends 1.1, +1.1, should be included.

Boundary Value Analysis

*2. If an input condition for the software-under-test is specified as **a number of values**, develop **valid test cases for the minimum and maximum numbers** as well as **invalid test cases that include one lesser than minimum and one greater than the maximum***

For example, for the real-estate module mentioned previously that specified a house can have four owners,

Valid Tests that includes tests for minimum and maximum values 1, 4 owners

Invalid tests that include one lesser than minimum and one greater than maximum 0, 5 are developed.

Boundary Value Analysis

3. *If the input or output of the software-under-test is an **ordered set**, such as a table or a linear list, develop tests that focus on the first and last elements of the set*

Example: Loan application

The diagram shows a loan application form with several input fields and checkboxes. Green lines connect specific fields to their boundary value constraints:

- Customer Name**: 2-64 chars.
- Account number**: 6 digits, 1st non-zero
- Loan amount requested**: £500 to £9000
- Term of loan** (checkbox): 1 to 30 years
- Monthly repayment** (checkbox): Minimum £10

Below the input fields, there are labels for calculated values:

- Term:
- Repayment:
- Interest rate:
- Total paid back:

A red circle with the number '44' is located near the 'Total paid back:' label.

Cause–Effect Graphing

Equivalence partitioning and boundary value analysis exercise the unit or system by looking at **each equivalence class in isolation**, but they do not exercise the unit with different combinations of inputs from the equivalence classes.

Cause–effect graphing is a way of doing this while avoiding **the major combinatorial problems that can arise**.

The technique relates the inputs and input conditions as defined by the specification (requirements, high-level design, or unit) such that for each input or set of inputs some output(s) are defined using either Boolean logic or a graph.

Cause–Effect Graphing

The process is as follows:

1. Partition the assertions of the specification to be tested, either by feature (in the case of system testing), by interface (in the case of integration testing), or by structure (in the case of unit testing).

Choose other criteria for partitioning at will.

2. Identify all causes and effects in the specification.
3. Identify each cause and effect by an unique number for each.

Cause–Effect Graphing

4. Draw the cause–effect graph by:

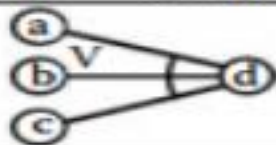
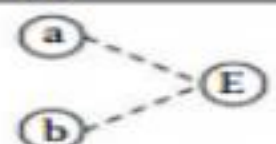
- Listing each cause as a node on the left-hand side of the paper and each effect as a node on the right-hand side of the paper.
- Relating the nodes with a line such that each cause and effect be either true or false, given
- some set of conditions represented by Boolean logic.
- Annotating the graph with constraints describing combinations of causes and/or effects that are impossible for environmental or syntactical reasons.

5. Convert the graph into a decision table by tracing all the causes for each effect. Each column in the table represents a test objective.

6. Write the tests.

Cause-Effect Graph Constructs

Figure below show constructs that can be used in cause-effect graphing.

Construct name	Meaning	Symbol
Identity	a causes b	
Not	a inhibits b	
Or	a, or b, or c cause d	
And	a and b cause d	
Xor	One of a, or b can be causes, not both(Exclusive OR)	

Cause–Effect Graph example

Eg. A database should have each file listed in a 10 -section master index by name and location. To display or list a section in the inquiry mode the operator must enter a D (display) or P (print), followed by a numeric character (0-9) to represent the section number, followed by “return” key

Error handling

If the first character is other than D, or P the system displays

INVALID COMMAND

If the second character is not in a range 0-9 the system displays

INVALID INDEX

Cause–effect identification

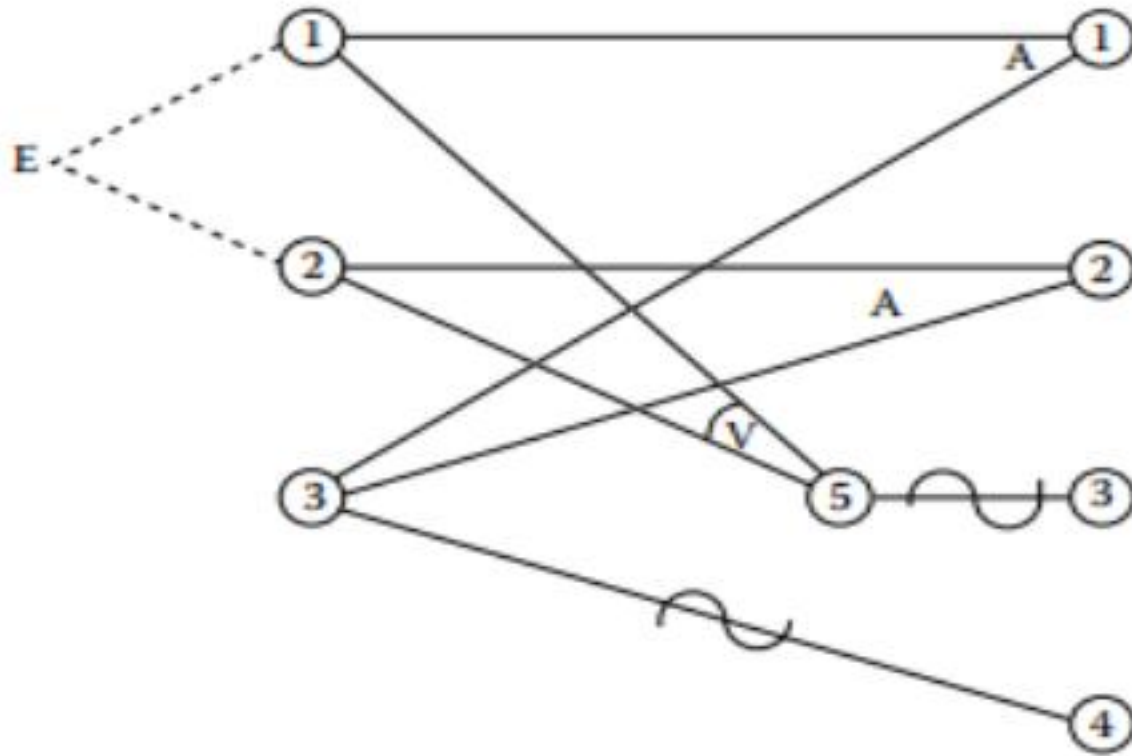
Causes

1. character 1 is a D
2. character 1 is a P
3. character 2 is a digit in the range 0-9

Effects

1. Index section is displayed
2. Index section is printed
3. INVALID COMMAND is displayed
4. INVALID INDEX NUMBER is displayed

A cause-and-effect graph



Example of a cause–effect decision chart

	Test objectives			
causes	1	2	3	4
1 (character 1 is a D)	T	F	F	
2 (character 1 is a P)	F	T	F	
3 (character 2 is a digit between 0-9)	T	T		F
effects				
1 (Index section is displayed)	T			
2 (Index section is printed)		T		
3 (INVALID COMMAND is displayed)			T	
4 (INVALID INDEX NUMBER is displayed)				T