CHAPTER 9: FUNCTIONAL TESTING

FUNCTIONAL TESTING

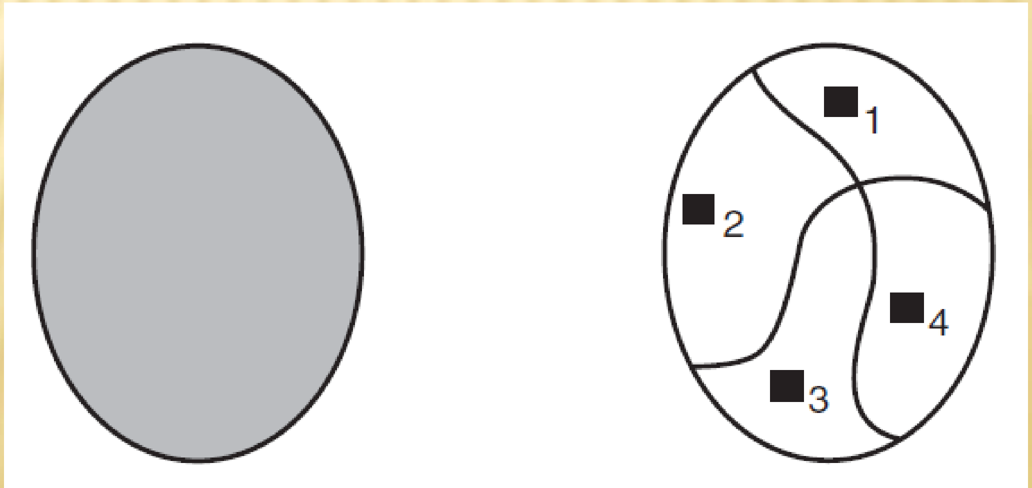
- Precisely identify the domain of each input and each output variable.
- Select values from the data domain of each variable having *important* properties.
- Consider combinations of special values from different input domains to design test cases.
- Consider input values such that the program under test produces special values from the domains of the output variables.

EQUIVALENCE CLASS PARTITIONING

- An input domain may be too large for all its elements to be used as test input.
- However, the input domain can be partitioned into a finite number of subdomains for selecting test inputs.
- Each subdomain is known as an equivalence class (EC), and it serves as a source of at least one test input.
- The objective of equivalence partitioning is to divide the input domain of the system under test into classes, or groups, of inputs.
- All the inputs in the same class have a similar effect on the system under test

EQUIVALENCE CLASS PARTITIONING

- An EC is a set of inputs that the system treats identically when the system is tested. It represents certain conditions, or predicates, on the input domain.
- An input condition on the input domain is a predicate over the values of the input domain.
- A valid input to a system is an element of the input domain that is expected to return a nonerror value.
- An invalid input is an input that is expected to return an error value.



IDENTIFICATION OF TEST CASES FROM EC

- Test cases for each EC can be identified by the following:
  - Step 1: Assign a unique number to each EC.
  - Step 2: For each EC with valid input that has not been covered by test cases yet, write a new test case covering as many uncovered ECs as possible.
  - Step 3: For each EC with invalid input that has not been covered by test cases, write a new test case that covers one and only one of the uncovered ECs.

IDENTIFICATION OF TEST CASES FROM EC

- **Example:** Adjusted Gross Income. Consider a software system that computes income tax based on adjusted gross income (AGI) according to the following rules:
  - If AGI is between $1 and $29,500, the tax due is 22% of AGI.
  - If AGI is between $29,501 and $58,500, the tax due is 27% of AGI.
  - If AGI is between $58,501 and $100 billion, the tax due is 36% of AGI.

IDENTIFICATION OF TEST CASES FROM EC

- Equivalence partitioning

BOUNDARY VALUE ANALYSIS

- The central idea in boundary value analysis (BVA) is to select test data near the boundary of a data domain so that data both within and outside an EC are selected.
- It produces test inputs near the boundaries to find failures caused by incorrect implementation of the boundaries.
- Boundary conditions are predicates that apply directly on and around the boundaries of input ECs and output ECs.

BOUNDARY VALUE ANALYSIS

- In practice, designers and programmers tend to overlook boundary conditions.
- Consequently, defects tend to be concentrated near the boundaries between ECs.
- Therefore, test data are selected on or near a boundary.
- In that sense, the BVA technique is an extension and refinement of the EC partitioning technique

BOUNDARY VALUE ANALYSIS (BVA)

- **Guidelines for BVA**
  - *The EC specifies a range: If an EC specifies a range of values, then* construct test cases by considering the boundary points of the range and points just beyond the boundaries of the range. For example, let an EC specify the range of −10 *≤ X ≤ 10. This would result in test data*
  - {−11, *−10,−9} and {9, 10, 11}.*

BOUNDARY VALUE ANALYSIS (BVA)

- **Guidelines for BVA**
  - The EC specifies a number of values: If an EC specifies a number of values, then construct test cases for the minimum and the maximum value of the number. In addition, select a value smaller than the minimum and a value larger than the maximum value. For example, let the EC specification of a student dormitory specify that a housing unit can be shared by one to four students; test cases that include 1, 4, 0, and 5 students would be developed.

BOUNDARY VALUE ANALYSIS (BVA)

- **Guidelines for BVA**
  - The EC specifies an ordered set : If the EC specifies an ordered set, such as a linear list, table, or sequential file, then focus attention on the first and last elements of the set.

BOUNDARY VALUE ANALYSIS (BVA)

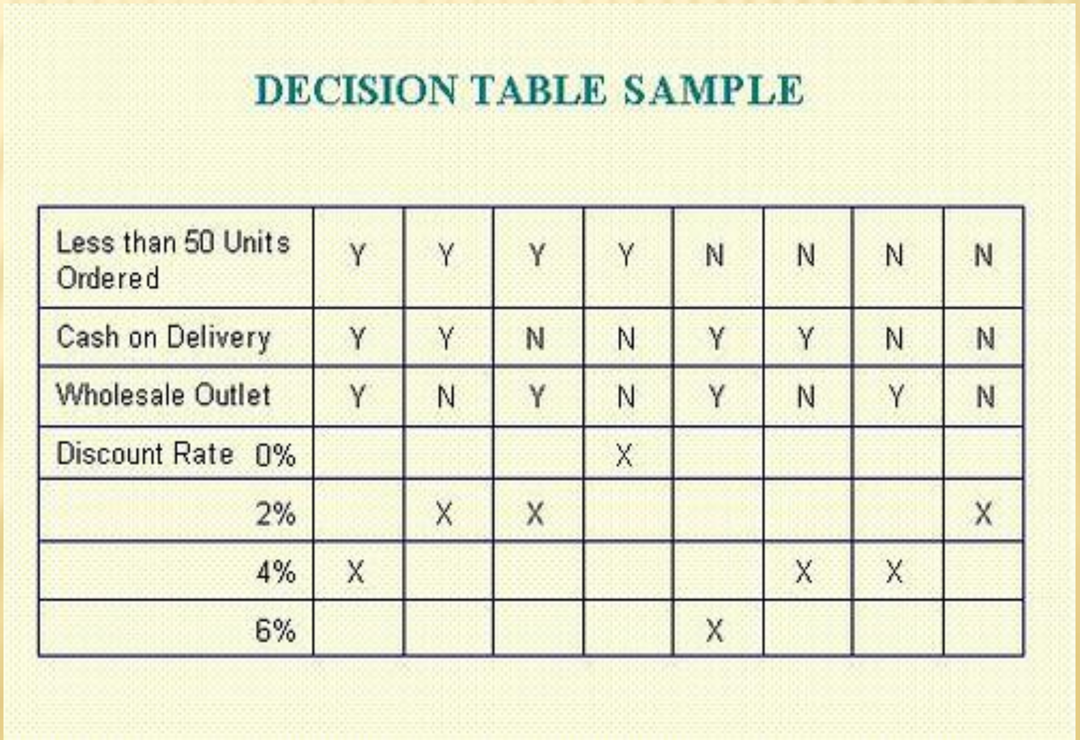
- EC1: \$1 ≤ AGI ≤ \$29,500; This would result in values of \$0, \$1, \$2, and \$29,499, \$29,500, \$29,501.
- EC2: AGI < 1; This would result in values of \$0, \$1, \$2
- EC3: \$29,501 ≤ AGI ≤ \$58,500; This would result in values of \$29,500, \$29,501, \$29,502, \$58,499, \$58,500, \$58,501.
- EC4: \$58,501 ≤ AGI ≤ \$100 billion; This would result in values of \$58,500, \$58,501, \$58,502, \$99 billion, \$100 billion, \$101 billion.
- EC5: AGI > \$100 billion; This would result in \$99 billion, \$100 billion, \$101 billion.

DECISION TABLES

- Different combinations of equivalent classes can be tried by using a new technique based on the decision table to handle multiple inputs.
- Decision tables have been used for many years as a useful tool to model software requirements and design decisions.
- It comprises a set of conditions (or causes) and a set of effects (or results) arranged in the form of a column on the left of the table.

DECISION TABLES

Company X sells merchandise to wholesale and retail outlets. Wholesale customers receive a two percent discount on all orders. The company also encourages both wholesale and retail customers to pay cash on delivery by offering a two percent discount for this method of payment. Another two percent discount is given on orders of 50 or more units. Each column represents a certain type of order.



DECISION TABLE SAMPLE

Less than 50 Units Ordered	Y	Y	Y	Y	N	N	N	N
Cash on Delivery	Y	Y	N	N	Y	Y	N	N
Wholesale Outlet	Y	N	Y	N	Y	N	Y	N
Discount Rate 0%				X				
2%		X	X					X
4%	X					X	X	
6%					X			

ERROR GUESSING

- Error guessing is a test case design technique where a test engineer uses experience to

(i) guess the types and probable locations of defects and

(ii) design tests specifically to reveal the defects.

ERROR GUESSING

- The following are the critical areas of the code where defects are most likely to be found:
  - Different portions of the code have different complexity. One can measure code complexity by means of cyclomatic complexity.
  - Portions of the code with a high cyclomatic complexity are likely to have defects. Therefore, it is productive to concentrate more efforts on those portions of code.
  - The code that has been recently added or modified can potentially contain defects. The probability of inadvertently introducing defects with addition and modification of code is high.
  - Portions of code with prior defect history are likely to be prone to errors. Such code blocks are likely to be defective, because of clustering tendency of the defects, despite the efforts to remove the defects by rewriting the code.

ERROR GUESSING

- Parts of a system where new, unproven technology has been used is likely to contain defects. For example, if the code has been automatically generated from a formal specification of the system, then there is higher possibility of defects embedded into the code.
- Portions of the code for which the functional specification has been loosely defined can be more defective.
- Code blocks that have been produced by novice developers can be defective. If some developers have not been careful during coding in the past, then any code written by these developers should be examined in greater detail.
- Code segments for which a developer may have a low confidence level should receive more attention. The developers know the internal details of the system better than anyone else. Therefore, they should be quizzed on their comfort levels and more test effort be put on those areas where a developer feels less confident about their work.

ERROR GUESSING

- Areas where the quality practices have been poor should receive additional attention. An example of poor quality practice is not adequately testing a module at the unit level. Another example of poor quality practice is not performing code review for a critical part of a module.
- A module that involved many developers should receive more test effort.If several developers worked on a particular part of the code, there is a possibility of misunderstanding among different developers and, therefore, there is a good possibility of errors in these parts of the code.