

CMSC 735
Assignment 5

November 26 2001
Due December 5, 2001

A major software development organization has decided to use a new unit test technique. The idea is that all code will be submitted to a library along with its specification and tested according to that specification. The tests will be made up using equivalence partitioning, based upon the specification. They will then measure the coverage of the test set to evaluate the quality of the tests.

Part 1 (12 points)

They want to evaluate the technique during the training course and improve it based upon what they have learned. Design an experiment that will allow them to evaluate the process in a classroom situation and learn enough to make improvements in its feasibility and effectiveness.

You should write a GQM evaluation goal (use the GQM template), give a specific, detailed evaluation model that they could implement and learn from (they care about cost and effectiveness of the technique), and give an experimental design, pointing out the major threats to validity of the design.

Part 2 (13 points)

Then they want to study it on a real application. They are open to options, e.g., use it on half of the project but not the other half. They have data from prior projects on the number of faults and failures found during unit test, system test, and acceptance test..

You should write a GQM evaluation goal (use the GQM template), give a specific, detailed evaluation model that they could implement and learn from, and give a study design, pointing out the major threats to the design.

You must limit yourself to a **MAXIMUM** of 4 pages, 12 size times font.

Equivalence Partitioning

Ideal Criterion: Divide the input domain into a set of equivalence classes such that if a test in an equivalence class succeeds, then every test in that class will succeed

Practical Criterion: Put inputs for which the behavior pattern is specified to be different into separate groups and regard these groups as forming equivalence classes.

Test case design by equivalence partitioning involves

- identifying the equivalence classes
- defining the test cases

Identify the equivalence classes

take each input condition and partition it into two or more groups

valid equivalence classes (vec) represent valid inputs to the program

invalid equivalence classes (iec) represent erroneous input values

Record the classes in a table as follows:

External Condition	Valid Equivalence Classes	Invalid Equivalence Classes

Given an input or external condition, identifying the equivalence classes is a heuristic process.

Some guidelines for identifying equivalence classes are:

1. If an input condition specifies a range of values (item count from 1 to 999)
 - identify one vec ($1 < \text{item count} < 999$)
 - identify two iec ($\text{item count} < 1$ and $\text{item count} > 999$)
2. If an input condition specifies a set of input values each handled differently
 - identify one vec for each one
 - identify on iec
3. If an input condition specifies a must condition situation
 - identify one vec containing the must condition
 - identify one iec not containing the must condition
4. If there is any reason to believe the elements of an equivalence class are not handled identically, split the class into smaller classes.

Identifying test cases

1. Assign a unique number to each equivalence class

2. Until all vec have been covered by test cases, write a new test covering as many of the uncovered vec as possible
3. Until all iec have been covered by test cases, write on test case that covers one and only one iec

Boundary Value Analysis

Experience shows that test cases that explore the boundary conditions have a higher payoff than test cases that do not, i.e., finding off by one errors.

Boundary value analysis differs from equivalence partitioning in that each edge of the equivalence class is the subject of a test rather than selecting a representative element from the class
test cases are derived by considering the result (output) space as well as the input space of equivalence classes

Some guidelines for analysis are:

1. If an input condition specifies a range of values (item count from 1 to 999)
 write test cases for the ends of the range, e.g., 1 and 999
 write invalid input test cases for situations just beyond the ends, e.g., 0 and 1000
2. Use the same guideline for each output condition.
3. If the input or output of a program is an ordered list, focus on the first and last elements.
4. Use your ingenuity to search for other boundary conditions based on the problem.