SELECTING THREE KEY TEST CASE DESIGN METHODS FOR UNIT TESTING

When conducting unit testing in software engineering, selecting effective test case design methods is crucial to ensuring individual components or functions behave as intended. If limited to only three methods, I would choose **equivalence partitioning**, **boundary value analysis**, and **statement coverage**. These methods collectively provide a robust testing strategy by combining logical input segmentation, edge case verification, and code execution assurance.

1. Equivalence Partitioning

Equivalence partitioning (EP) divides input data into valid and invalid partitions where test cases are created for each partition. The idea is that if one input from a partition passes, the rest will likely behave similarly, reducing the total number of test cases needed while maintaining adequate test coverage (Pressman & Maxim, 2020).

For example, consider a function that accepts ages between 18 and 60 for a user registration system. EP would create three partitions:

- Valid: 18–60
- Invalid: below 18
- Invalid: above 60

You would test one value from each partition (e.g., 25, 15, and 65) instead of every possible value in the range. This helps detect logical errors and improper input handling without testing all inputs individually.

2. Boundary Value Analysis

Boundary value analysis (BVA) complements equivalence partitioning by focusing on the edges of input ranges, where errors often occur. Off-by-one errors or incorrect relational operators in code are frequently found at boundaries, making BVA particularly useful for catching these issues (Sommerville, 2016).

Using the same age validation function, BVA would suggest testing:

- Lower boundary: 17, 18 (just below and at the minimum)
- Upper boundary: 60, 61 (at and just above the maximum)

BVA is effective because it targets conditions where developers often make mistakes. For instance, if the function uses age > 18 instead of age >= 18, the test case with 18 would fail and reveal the bug.

3. Statement Coverage

Statement coverage is a white box testing method that ensures every line of code in the unit is executed at least once. This method helps identify unreachable code, unhandled conditions, or logic that may not be exercised through other black-box techniques.

Consider a function that calculates shipping fees based on the number of items:

```
def calculate_shipping(items):
    if items == 0:
        return 0
    elif items < 5:
        return 5
    else:
```

```
    return 10
```

To achieve full statement coverage, the test suite must include:

- items = 0 (returns 0)
- items = 3 (returns 5)
- items = 5 (returns 10)

Without covering all branches, portions of the code might never be tested, leading to undetected bugs.

Conclusion

Choosing **equivalence partitioning**, **boundary value analysis**, and **statement coverage** ensures a well-rounded approach to unit testing. EP efficiently reduces test cases while covering key input categories, BVA focuses on error-prone edge cases, and statement coverage guarantees that all logic paths are exercised. When applied together, these methods enhance test quality and confidence in code correctness, especially in time-constrained environments.

References:

Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education. https://www.mheducation.com/highered/product/Software-Engineering-A-Practitioners-Approach-Pressman.html

Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education. https://www.pearson.com/en-us/subject-catalog/p/software-

engineering/P200000003258/9780137503148?srsltid=AfmBOoq9esvBFCOh9ElRe8CMQjss-6wCgHCqFY_3G-Y8c8KVYWtfaXie

Wordcount: 479