Samuel Harvey
Software Testing Mid Term - Spring 2026
Test Breaker

I was unsure if you wanted us to develop a full broken class to use as an example, but talked myself out of it to keep my Maven and JaCoCo clean, so you could fully examine my working tests and test class! Instead, I have taken excerpts from the code and made changes, which I have highlighted in red, that illustrate faults that would be caught by each method, and to illustrate, as directed, how each different testing method has it's own strengths and weaknesses.

Fault 1 Caught by Equivalence Partitions (Ch. 2)

```
if (kwh > 500 && kwh <= 1500) {
   if (hasSmartDevice && peakOptOut) {
       rebatePercent = 0.15;
   } else if (hasSmartDevice || peakOptOut) {
       rebatePercent = 0.010;
   }
```

For this fault, I modified the second tier "else if" statement to return a rebatePercent = 0.010, instead of the .1 that is outlined in the software specification. This should be caught in Equivalence Partition testing when the second tier partition is tested with one of the two boolean flags set as true. An incorrect return value (in this case, off by a decimal place) seems a good example to show that equivalence partitions can be a good tool for finding these types of return value faults, as they test a representative value from each logic block, which is checked against the specified expected return value.

Fault 2 Missed by EQs, but caught by BVA (Ch. 3)

```
if (kwh > 500 && kwh < 1500) {
   if (hasSmartDevice && peakOptOut) {
       rebatePercent = 0.15;
   } else if (hasSmartDevice || peakOptOut) {
       rebatePercent = 0.10;
   }
```

For this fault, I modified the logic in the initial if statement to be <1500 and not <=1500. By changing this from inclusive to exclusive, I introduced a very common fault "on the boundary", that would be caught by boundary value analysis, but not necessarily by EP alone. Because BVA specifically checks where the logic changes, a successful test suite with BVA would include 1500, which now would not be caught in this tier, and would return the standard 0.0 rebate rate, which would be incorrect.

Samuel Harvey
Software Testing Mid Term - Spring 2026
Test Breaker

Fault 3 Missed by BVA, but caught by Decision Table (Ch. 4)

```
else if (kwh > 1500) {
    if (hasSmartDevice || peakOptOut) {
        rebatePercent = 0.20;
    } else {
        rebatePercent = 0.05;
    }
}
```

For this fault, I made change to the logic && statement that initially required both booleans to be set to true for the rebate percentage to be .20. I changed this to a logical OR, which could conceivably happen in this programs development, as that would mirror the previous tiers logic.This fault might be missed by BVA as it's focus is on the boundaries between logic groups, and does not extensively cover boolean expressions. Decision tables on the other extensively cover TRUE / FALSE logic, and all possible combinations in a program. This would be caught by a Decision Table.

Fault 4 Missed by Decision Tables, but caught by Code Coverage (Ch. 5)

```
if (kwh == 1642) {
    if (hasSmartDevice && peakOptOut) {
        rebatePercent = 1.0;
    }
```

For this fault, I created a special "retired"l if condition to catch a kwh of exactly 1642, to simulate a potential "Columbus day" that gave free electricity for that billing cycle if you happened to use exactly 1642 kwh. This is meant to simulate potential left-over code from previous software specification documents that might have been left in the program. Because this condition is not covered in the specification, there would be no way for it to be covered by any of the previous "black box" testing strategies. Code coverage would flag this as not being tested, and would alert us to it's needed removal, and stop anyone from accidentally getting free electricity when a promotion has expired.

Fault 5 Missed by Code Coverage, but caught by Branch Coverage (Ch. 6)

```
public double calculateRebate(double kwh, boolean hasSmartDevice, boolean
peakOptOut) {
    if (kwh <= 0) {
        throw new IllegalArgumentException("Usage must be positive.");
    }

    double rebatePercent = 0.05;
```

Samuel Harvey
Software Testing Mid Term - Spring 2026
Test Breaker

For this fault, I changed the base rebatePercent to be an incorrect .05, essentially giving everyone a discount. This could be missed by just code coverage, as code coverage doesn't necessarily force coverage of this implicit “else” branch. In code coverage, this might get missed if there are no specific tests written for it, as this number is set at the beginning of the program and then overwritten. So while it technically “runs” every time, and would show as 100% on code coverage, the logic is still very much wrong. Branch coverage would make sure you cover this implicit else branch and hopefully spot this error.