Nature of Development Testing

The following might seem a bit obvious (to such experienced devs) but worth stating anyway…

- When you test, you execute a program
- You "exercise" it using synthetic data
- Checking the results for errors or anomalies

- Aim of testing is to identify presence of defects
- A successful test is one that finds a defect !

"Absence of evidence is not evidence of absence"

Example

Consider a function that,
if given a day, month and year,
would tell you what day of the week it was
Mon, Tues, Weds, Thurs, Fri, Sat, Sun

(Zeller's congruence)

How do we know if it is ***completely*** correct ?

Correctness & Completeness

- We could run program with a few arbitrary dates
- But that probably won't reveal many errors
- It isn't methodical or systematic enough

- How to ensure complete & comprehensive test ?
- What we need is a proper, organised strategy…

Coverage versus Practicallity

- Functions have finite number of input parameters
- Complete black-box testing would try every single possible combination of such input values

- This obviously isn't always practical…
- For the previous "day of the week" function, 100 years would require around 37k combinations !

- We need to identify a ***sample set*** of test cases
- To ensure full coverage, but without having to exhaustively try all combinations

Equivalence Partitioning

A technique to help systematic selection of test cases

An equivalence partition is:

"A cluster of input values for which a program should behave in the same way"

An example partition might be set of all +ve numbers (up to max allowable in the programming language)

The number 54 is ***likely*** to have a ***similar*** effect to 55 !

Selecting Test Data

For each partition, select upper & lower boundary
(Boundary values can be overlooked by coders)

Also choose a data value from middle of partition
Should be representative of main body of values

In this way, we end up testing each partition, rather than every single possible value or combination

Example Equivalence Partitions

Imagine we had a function to convert a grade (as a percentage) into a degree classification

(1st, 2.1, 2.2. 3rd, Fail)

The equivalence partitions might be as follows:

The test case values might then be:

-10, -1, 0, 1, 26, 39, 40, 43, 49, 50, 56, 59, 60, 63, 69, 70, 80, 100, 101