Minimum Testing Requirements

James Wilcox and Kevin Zatloukal

March 2024

Our first rules supersede all others:

- **Exhaustive Testing**: for functions with at most 20 *allowed* inputs, test every input
- **Disallowed inputs**, either by the types or the specification, do not require testing
- Test each function individually. Assume the other functions it calls work properly

If a function has more than 20 allowed inputs, then we split them up into "subdomains" (subsets) using rules described below and then choose multiple test inputs for each subdomain, as described in the next section.

Testing a Subdomain

The following rules must be followed when choosing test inputs for an individual subdomain:

- Every subdomain containing at least 2 inputs must have **at least 2 test cases**.[1]
- Every **boundary case** must be tested along with at least one non-boundary case.[2]

Splitting Into Subdomains

For the simplest code, containing no loops or recursive calls chains[3], we use the following rule:

- Separate inputs that take **different paths** through the code into separate subdomains.

For functions with recursive calls, we split inputs based on the **maximum depth** of recursion made during the call. Specifically, maximum depth 0, 1, and 2+ are split into their own subdomains. We then split those inputs into smaller subdomains as follows:

- For inputs that make no recursive calls, inputs that take different paths are in separate subdomains.
- For inputs that make 1 recursive call, inputs that take different paths are in separate subdomains.[4]
- For inputs that make 2+ recursive calls, we focus on the outermost two calls in the call sequence. Inputs that take different paths in the outermost call or the first recursive call are in separate subdomains.

[1] One input subdomains require only one test.

[2] In general, if the subdomain has $n \geq 1$ boundaries, then the subdomain requires $n + 1$ test cases.

[3] Any sequence of calls that has a call to our function while still executing an existing call to our function counts as recursion, even if the function does not call itself directly.

[4] Note that the path includes the code executed in both the outer call and the inner call.

Loops

Loops are largely analogous to recursive calls, but for the sake of clarity, we will spell this out in detail.

For functions with loops, we split inputs based on the number of iterations made during the call. Specifically, inputs that make 0, 1, and 2+ iterations are in their own subdomains. We then split those inputs into smaller subdomains as follows:

- For inputs that make no iterations calls, inputs that take different paths are in separate subdomains.[5]
- For inputs that make 1 iteration, inputs that take different paths are in separate subdomains.[6]
- For inputs that make 2+ iterations, we focus on the *first* two iterations. Inputs that take different paths in the either of those iterations are in separate subdomains.

Rather than including the code after the loop as part of the paths considered above, we conceptually treat it as a helper function called when the loop exits. We apply the rules to that code on its own, considering what subdomains and cases are necessary. It is okay to skip a case that would be a duplicate of one used to test the loop itself, but the case must still be documented / explained.

If a function has two (or more) loops in sequence, we likewise conceptually split the code into multiple parts, each containing either a single loop and the code before it or the code after all the loops. Then, as above, we apply the rules to each part to determine subdomains and cases. Again, duplicate cases can be skipped but still must be documented.

Nested loops on the other hand, must be considered together. For example, when we consider all inputs that make exactly one iteration of the *outer loop*, we would need to split inputs that make 0, 1, or 2+ iterations of the *inner loop* into separate subdomains. Likewise, for inputs that make 2+ iterations of the outer loop, inputs that make 0 iterations of the inner loop on their last (or second-to-last) iteration of the outer loop, would be separated from those that make 1 iteration of the inner loop on their last (or second-to-last) iteration.

Escape Hatches

In general, we aim to have around 10-30 tests for a typical function. Fewer than 10 tests is highly suspicious if not embarrassing. There must be a good explanation (in terms for subdomains) for having so few tests.

More than 20 tests is in no way bad; however, it may be time too time consuming to write so many. For that reason, we will include a few "escape hatches" that can be used to reduce the number of tests when the rules above would indicate more than 20 tests. If the first escape catch reduces to 20 or fewer tests, then stop there. If that still leaves the number of tests above 20, then apply the second escape match as well.

- Only consider paths in the outermost call (rather than two calls) of a recursive function.
- Only choose one example from a subdomain.

[5] Here, the path includes only the code executed before (but not after) the loop.

[6] here, that the path includes the code executed before and inside (but not after) the loop.