T-EMB-800 – Embedded World

Unit test
-
Advices

Unit test briefing

A unit test which satisfies aeronautical norms must contain several test cases which are defined in following categories.

1. Parameters

The function prototype is:

```
//RETURNED_TYPE FUNCTION_NAME(      /* RANGE_RETURNED_VALUE */
//                /*FLOW_PARAM1*/ PARAM1_TYPE PARAM1,/* RANGE_PARAM1 */
//                /*FLOW_PARAM2*/ PARAM2_TYPE PARAM2,/* RANGE_PARAM2 */
//                               ………)
```

Each parameter must be tested at least once, in a significative way, to its minimum and maximum values of the range defined in the header. Then:

- `PARAM1` must be tested to values `MIN[RANGE_PARAM1]` and `MAX[RANGE_PARAM1]`,
- `PARAM2` must be tested to values `MIN[RANGE_PARAM2]` and `MAX[RANGE_PARAM2]`,
- …
- `RETURNED_VALUE` must be computed to `MIN[RANGE_RETURNED_VALUE]` and to `MAX[RANGE_RETURNED_VALUE]`.

If the value range is not specified, the default range is the type range.

Moreover, a parameter has 3 possible flows: input, output, input/output. Then, if a parameter has the flow:

- input: at the end of the execution of the function, the parameter value must be the same than the one given in input,
- output or input/output: at the end of the execution of the function, the parameter value must be different than the one given in input.

2. If

To test an if expression, several test cases must be considered.

a. Basic test

To test a basic if expression, test cases TRUE and FALSE must be done.

We talk about MCDC (Modified Condition / Decision Coverage) to explain that all cases that can occur are tested.

Then, if an if expression combines XOR and/or AND, just testing TRUE or FALSE value is not sufficient.

b. XOR

To test a XOR expression (VAR_X || VAR_Y), the three following test cases must be done:

- VAR_X is TRUE and no specific condition on VAR_Y,
- VAR_X is FALSE and VAR_Y is TRUE,
- VAR_X is FALSE and VAR_Y is FALSE.

c. AND

To test a AND expression (VAR_X && VAR_Y), the three following test cases must be done:

- VAR_X is FALSE and no specific condition on VAR_Y,
- VAR_X is TRUE and VAR_Y is FALSE,
- VAR_X is TRUE and VAR_Y is TRUE.

d. Inflexion point

The inflexion point is the value that allow the expression to change is state. We can distinguish several cases on inflexion point:

<u>Case 1:</u> integer value (int, unsigned int etc…)

- expression VAR_A == K_CONST

The inflexion point is reached when VAR_A == K_CONST value. Then, the following tests must be done:

- `VAR_A = K_CONST – 1` ➔ inflexion_point – 1.
- `VAR_A = K_CONST` ➔ inflexion_point.
- `VAR_A = K_CONST + 1` ➔ inflexion_point + 1.

- Expression VAR_A > K_CONST

The inflexion point is reached when VAR_A == K_CONST value. Then, the following tests must be done:

- `VAR_A = K_CONST + 1` ➔ inflexion_point + 1.
- `VAR_A = K_CONST` ➔ inflexion_point.

- Expression VAR_A >= K_CONST

The inflexion point is reached when VAR_A == K_CONST - 1 value. Then, the following tests must be done:

- `VAR_A = K_CONST – 1` ➔ inflexion_point - 1.
- `VAR_A = K_CONST` ➔ inflexion_point.

- Expression VAR_A < K_CONST

The inflexion point is reached when VAR_A == K_CONST value. Then, the following tests must be done:

- `VAR_A = K_CONST – 1` ➔ inflexion_point.
- `VAR_A = K_CONST` ➔ inflexion_point + 1.

- Expression VAR_A <= K_CONST

The inflexion point is reached when VAR_A == K_CONST + 1 value. Then, the following tests must be done:

- `VAR_A = K_CONST + 1` ➔ inflexion_point.
- `VAR_A = K_CONST` ➔ inflexion_point - 1.

<u>Case 2:</u> expression VAR_A == K_BOOL, VAR_A is a boolean.

The inflexion point is reached when VAR_A == K_BOOL (TRUE or FALSE) value. Then, the following tests must be done:

- `VAR_A = K_BOOL` ➔ inflexion_point.
- `VAR_A = !K_BOOL` ➔ !inflexion_point.

<u>Case 3:</u> double value (double, float etc…)

- expression VAR_A == K_CONST

The inflexion point is reached when VAR_A is close enough to K_CONST value. Then, the following tests must be done:

- `( VAR_A – K_CONST ) < 0.001` ➔ inflexion_point reached.
- `( VAR_A – K_CONST ) > 0.001` ➔ inflexion_point not reached.

- Expression VAR_A > K_CONST

The inflexion point is reached when VAR_A == K_CONST value. Then, the following tests must be done:

- `( VAR_A – 0.001 ) > K_CONST` ➔ inflexion_point reached.
- `( VAR_A > K_CONST )` ➔ inflexion_point not reached.

- Expression VAR_A < K_CONST

The inflexion point is reached when VAR_A == K_CONST value. Then, the following tests must be done:

- `( VAR_A – 0.001 ) < K_CONST` ➔ inflexion_point reached.
- `( VAR_A < K_CONST )` ➔ inflexion_point not reached.

<u>Case 4:</u> expression VAR_A == NULL, VAR_A is a pointer.

The inflexion point is reached when VAR_A == NULL value. Then, the following tests must be done:

- `VAR_A = NULL` ➔ inflexion_point.
- `VAR_A != NULL` ➔ inflexion_point not reached.

e. Switch

Switch condition works the same way as a succession of if/else.

3. Loops

Following conditions must be checked to completely test the for/while instruction:

- 0 loop: entry condition for the loop must never be satisfied,
- 1 loop: entry condition for the loop must be satisfied only once,
- 2/+ loops: entry condition for the loop must be satisfied at least twice.

4. Mocking

When a function is called in the main tested function, its behaviour must be simulated in order to not disrupt the main tested function behaviour.

Then, minimum and maximum value of range must be tested for each parameter (if possible) and for the return value of the stubbed function. Flows must be respected too.