Testing w/full coverage is not enough

March 2018

1 Testing and full coverage

Coverage is a useful metric of testing, but, high coverage is *not* the goal. Code can have 100% coverage and still have bugs. For example, if some code is *missing* and thus software doesn't do what it's supposed to do.

1.1 100% covered code with a bug

In a module that had 100% coverage, both line and branch ones, there was a pretty serious bug.

The module deals with timer lists, implemented as (doubly) linked lists. Each element carries info on "how much ticks should pass after the previous node expires until we are to expire". For example, a list might look like this at some point (showing only forward pointers):

```
T1(100) -> T2(20) -> T3(38) -> T4(2)
```

which represnts that we have 4 active timers which will expire:

- T1 in 100 ticks
- T2 in 120 ticks
- T3 in 158 ticks
- T4 in 160 ticks

This makes handling timer lists pretty efficient. "Tick" is a unit of time measure as used in the application (ms is common).

The bug was simple - if we're cancelling a timer, thus removing it from the list, and it is the very *first* timer in the list (the next to expire), it was handled badly. The time for this timer to expire was *not* added to the next timer.

To illustrate, suppose we have two timers, both started to 100 ticks, but, 1 tick "apart". So, after the start of the second timer, list looked like this:

```
T1(99) -> T2(1)
```

Let's say 40 ticks passed, list would be:

```
T1(59) -> T2(1)
```

Then T1 got cancelled. List *should* be:

```
T2(60)
```

But, because of the bug, the list was:

```
T2(1)
```

So, T2 will expire after 41 tick, instead of after 100 ticks.

There was a branch of code that handled this case. But, the code to add time "time to expire" was missing. It was C, and this line was missing:

```
list->timeout_left_ms += to_remove->timeout_left_ms;
```

1.2 Why did the tests not "catch" this?

The tests did test the removal of first element. But, since the `timeout_left_ms` was internal to the module, they didn't check it. They also didn't check "what if some time passes after the removal". In our example, what if, say, 40 ticks pass - T2 should *not* expire (but, it did).

Obviously, this simple scenario was not envisioned in testing, but each feature (function) was tested independently. Test for removal only tested removal. Test for "time passing by" (and timers expiring - or not) only tested time. Bugs are often hidden in scenarios that combine different features.

1.3 Conclusion

To test with good quality, one needs to consider the usage scenarios and test them (all, if possible). One should *not* consider the coverage metrics. Even more - the coverage metrics should *not* be the goal. We don't want to cover the code, we want to test it. Coverage metric is just an indication - if coverage is poor, it's not very likely that the tests are very good.