ortask

Why is Software Testing So Complex?

And Why Common Test Management Tools Don't Help.

An Ortask White Paper

Test automation and test management by itself do not increase testing efficiency, yet virtually all software testers subscribe to this misconception. This paper explains why such approaches don't work and illustrates what is really needed to manage the testing complexity.

Contents

The Myth of Automation

The urge and fads to "*automate ruthlessly*" have led many testers to the mistaken belief that their automation efforts will *automatically* decrease both testing costs and time. For an example, see items 4 and 5 in **this link**[1].

This might be the case for small, simple systems, but this assumption collapses with systems and projects in the mid- to enterprise-size range, where the manifold kinds of pre- and post-conditions introduce dependencies that are not typically accounted for by tests.

Let us explore a fictitious perspective that will better illustrate the dilemma and will, in particular, illustrate the main problem: the problem of *complexity in software testing*.

Suppose that it is the year 2136 and you have developed the next best widget, but the only way to sell it is via direct, face-to-face contact with potential customers. Suppose your sales force consists of a single unit of the latest super-realistic "*I Can't Believe It's Not Human Salesrobot-3000*" and you have the ability to automate the selling routine: you automate the many greetings, the opening pitch, the extended presentation and the many closing approaches.

You then successfully program and send your super-realistic robot on its way to visit several cities for the day. You don't care how your robot gets to the customers; you just want to see how many sales it made when it comes back.

This story illustrates many analogies between typical (i.e. *naïve*) testing and smart, *optimal* testing. Here's why.

In the story, the robot will likely not come back at the end of the business day. It may not even come back until several days later.

One reason for this is because you opted not to by the license for the "*Hamiltonian Cycle*" feature, so your robot visited all cities in the worst possible sequence. That is, your robot could have returned to you sooner and gone to more customers in subsequent days, but it didn't because it was inefficient in the way it visited customers. To put simply: even though the selling was automated, there was *waste* in the way your robot made its sales travels.

With testing, you want to see *results* – you may want to see the green bar more often than the red bar, but you want to get at least *some* feedback regardless. Your tests might run to completion but the *order* in which they run will likely be *sub-optimal*.

[1] For those who want to know what the link is before clicking: http://agilebooknote.blogspot.com/2010/06/agile-is-ruthless-automated-testing.html

In the simplest case, this means that a dependency that was satisfied by a previous test that could have been exploited by a subsequent test is destroyed. The typical case is more complex and involves destroying various degrees or layers of dependencies that could have been leveraged.

So even though the verification of state or behavior of the SUT is automated, there is *waste* in the way the tests are executed that leads to inefficient and *slow* testing.

Tests that deal with mid- to enterprise-size systems usually contain emergent inter-dependencies that are inherently difficult to visualize completely and, thus, to leverage. Let's take a simple case to illustrate this point and develop some intuition on the complexity.

Show Me The Testing Complexity

Suppose that each test within your suite contains exactly one pre-condition and exactly one post-condition. An upper complexity bound in this case would be for each test to share a dependency with all other tests, meaning that any test can be run before any other but also that the same test can be run after any other.

Using graph theory, let's represent the *dependencies* as edges (links) and the *tests* as vertices (nodes). Visually, your test bucket would look like Figure 1.

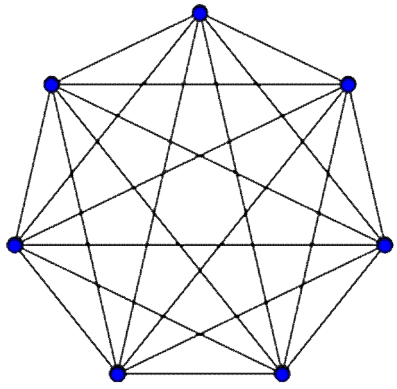

Figure 1: visualizing your test bucket

The quantity of dependencies between tests will equal twice the number of edges in a complete graph, namely

$$\frac{2n(n-1)}{2}$$

which becomes

$$n^2 - n$$

Now, assume a more common case where each test contains more than one distinct pre-condition and more than one distinct post-condition. The good news is that how many pre- and post-conditions exist for each test will simply affect the constant factor of the complexity bound.

For example, if every test contains exactly 4 distinct pre-conditions and exactly 4 distinct post-conditions, then the quantity of dependencies (edges) in the upper bound between tests (vertices) will equal four times the number of edges in a complete graph, namely

$$\frac{4n(n-1)}{2}$$

which becomes

$$2(n^2 - n)$$

Allowing the number of pre- and post-conditions to vary independently will indeed change the upper bound but only in its constant factor because the graph is complete (recall that we are looking at the worst case where every test has inter-dependencies to every other test).

Thus, the quantity of test inter-dependencies in any given test suite is

$$O(n^2)$$

However, remember that this complexity measure only considers test inter-dependencies. Pile on the actual testing process with many machines, multiple testers, real-life constraints and this nice upper bound immediately becomes a scary *lower* bound of the form

$$\Omega(n^2)$$

That's right. What this means is that the complexity of the testing process will *never* be asymptotically lower than n^2. Most likely, the overall complexity in your own testing process is already exponential.

With this analysis it is easy to see why the testing process is inherently riddled with such a high amount of complexity.

Common Test Management Tool Shortcomings

But do test management tools really help here?

The surprising fact is that every test management tool provides *zero* support for actually managing the testing complexity. Instead, they simply provide a database to store tests, attractive reports to showcase the work that you have already done when you execute tests, and call themselves *management* tools.

By focusing their marketing on the database and the reports, makers of common test management tools excel at hiding the real problems that testers need solved and instead keep test teams unaware of both the nature and the source of the testing complexity; or even that this complexity exists.

This also explains why test teams frequently run over their scheduled testing time and over their budget even though they all use modern, "state-of-the-art" test management tools which were trusted to manage this kind of complexity in the first place.

Common test management tools completely miss the mark for the critical facets of test management that really count. For instance, they do not help testers to visualize, comprehend and leverage the amount of dependencies that exist in their test buckets. This causes testers to end up *brute-forcing* the execution of tests against the SUT without even realizing that their approach is sub-optimal.

In other words, the typical organization executes their testing with high amounts of *waste* that can be avoided by simply *sequencing* tests in a more optimal manner, something that common test management tools are not able to provide even though they have all the necessary information to support this.

Test Management Tool Revolution

In Ortask, we're actively working to change this frustrating state of affairs. With revolutionary algorithms that solve these types of root problems in our upcoming world-wide release, Ortask will blow all test management tools out of the water.

Wouldn't it be nice to get your hands on the *next-generation* test management tool? We believe that we are achieving that in Ortask, and we would like to share.

For instance, with Ortask you can automatically re-order your tests to increase efficiency by at least *25%*[2]. Our algorithms can also speed up your test writing process by intelligently inferring post-conditions and can even go back in time by finding pre-conditions that you forgot to document.

These are exciting advances in the way we do testing, but we need your help. Without your support, test management will continue to be the same antiquated game of storing your tests in a database and showing a report of the work you've done; and will not advance to solve your real problems.

Take part in the Ortask *revolution* and join our small, but quickly growing active community of supporters. You can begin by signing in to the front page (www.ortask.com) to let us know that you're interested in progress.

You can also friend us on Facebook at http://www.facebook.com/ortask. Finally, you can always send Mario, Ortask's founder and chief engineer, an email with your thoughts and support. Unlike many test management tool makers, he would be truly delighted to hear from you and strike a conversation!

It's time us testers demanded *real* test management and stopped accepting simple dashboards that do not manage the true testing complexity. It's time to help *advance* the practice of testing. Will you be a key player?

[2] Based on actual usage at IBM and experimental results.

About Ortask

Ortask provides next-generation predictive and advanced analytics tools that help software test teams truly reduce their testing costs and increase their agility. We provide test management tools on steroids, so to speak. We are a small company with a big mission. Learn more about Ortask or join our active user community at www.ortask.com, and on Facebook at http://www.facebook.com/ortask.