

The State of Functional Software Testing: Continuing to Solve the Wrong Automation Problem

The major problem with functional software testing today is that we continue to solve the wrong problem. We spend millions of dollars worrying about whether we built the test code the right way. Instead, we should be asking: Does the application under test (AUT) work the way it is supposed to work? Does it solve the business problems that the software was developed to solve and generate revenue?

Let's take a typical example. The developer builds an application to buy a book on a website. Here's the way the application might be tested today.

Step One: The application code is delivered to the QA group for automated testing. 90% of the time, the functional test tool that this group uses for software testing is not set up correctly because the test tool manufacturer typically offers a "one size fits all" configuration. The company knows that the test scripts it will generate – via record and playback -- to test the application will not re-run and will fail because of timing issues, object recognition errors, object repository errors, etc.

Step Two: Rather than solve the set up problem, the industry solution has been to code around it. Instead of replicating the actions taken to buy a book online (type in author's name, click on name, etc.), the company replicates the code used to create each action. This involves re-creating thousands of lines of code. Many times, the effort to code around the problem is much more labor intensive than manual testing, so to save time and money, some companies choose not to automate software testing at all.

Step Three: As the cost of retaining test engineers to write code (descriptive programming) becomes more expensive, and requires specialized programming skills, companies that do a lot of automated software testing have sought to save money by outsourcing their testing to highly skilled, but less expensive, offshore resources.

Step Four: These highly skilled technical experts build complex coding architectures (testing frameworks or engines) to mirror the coding of the application itself, instead of testing the end user business processes that should be tested. These frameworks are coded to run hundreds or thousands of scripts and replicate what the developers do. Not surprisingly, this work requires many people to upkeep code, because a change in the application code requires a mirror change in the test code, as well. The AUT often goes into production with

errors anyway, because (1) the business personnel do not know how to use the testing frameworks, (2) the offshore programmers do not fully understand the business processes that the AUT is supposed to handle, and (3) the test focus was on coding the AUT, instead of how the AUT was supposed to function.

Step Five: The offshore programmers are paid for their expertise in descriptive programming. Companies have to teach their offshore contractors what their software does and what business problems it solves. This is also expensive and time consuming. With time, the work becomes voluminous, so that many companies recognize that the offshore model is no less expensive than doing the work in-house. There has been a movement to bring software testing back in-house to address this concern.

All this effort creates and works around problems that do not have to exist, if the company just addressed the problem in Step One: Fix the architecture underlying the set up of the functional test tool. If the functional test tool worked the way it should, Steps Two, Three, Four, and Five are completely unnecessary. As coding gets more and more complex, the industry continues to spend more and more money and time developing solutions to the wrong problem. We soon will have Steps Six, Seven, and Eight. We continue to ask the wrong question: Did we build the testing code the right way? We should be asking: Does the application work the way it is intended? Does it solve the business problem the software is intended to solve?

Most business analysts (BA's) just want to hit "replay" and have their test script run. They don't want to "throw it over the wall" to engineering to code. BA's get frustrated when they cannot test their software sufficiently to catch errors. Management gets frustrated when software goes into production and complaints come back that the software doesn't work the way it should.

TAAS, a software testing solution from Telesis, LLC, solves the right problem. TAAS, which stands for Telesis Advanced Automation Solution, allows companies to fix the architectural foundation underlying the set up of their functional test tool to make it work as intended.

A software tool within TAAS, called BT3, takes the "one size fits all" approach of the leading functional test tool vendors and actually makes it work with each individual AUT. First, BT3 modulates the functional test tool to the speed of the AUT, so the functional test tool does not time out if the AUT is faster or slower than the "one size fits all" timing. Second, the software prevents object recognition errors that programmers traditionally code around by personalizing the functional test tool configuration to read into the AUT. Third, BT3 aligns developer design details with end user AUT utilization. With BT3, there is no longer a need for descriptive programming or complex frameworks.

TAAS offers other benefits, as well.

1. **Ease of Use.** TAAS's BT3 is a functional testing solution that was intended for use by both your business people and your technical people: BA's, QA's, and developers. The technical aspects of software testing become a subset of the application's required business functioning.
2. **Priority Grids.** BT3's priority grids let you begin building test scripts, at the same time (in parallel) that you begin coding. Developers, IT, QA, and business resources can use the same testing criteria, all of which are available as the application pieces are coded. No need to wait to build a testing framework over months and months. This applies to all phases of testing: unit, systems, integration, and acceptance testing. Everyone knows at each phase of the project what needs to work correctly for the application to be production ready. BT3 lets you know what failed, and why.
3. **Speed to Market.** TAAS becomes part of the software development process because you do the software testing in parallel with the software development process. Build test scripts, objects, and screens in parallel with coding. The testing architecture ties directly to the business processes that the application is supposed to perform.
4. **Cut software testing and maintenance costs by 35% to 50%.** Time is money. No more need for expensive or outsourced testing personnel.