

# Testers Do Not Grow on Trees!

A whitepaper that explores the continued reliance on manual testing in software quality and asks if there is a way to make manual testing less... er, manual?



## Abstract

QA departments around the world continue to rely on manual processes for 80% or more of their software testing. Indeed in many places no automation is carried out at all.

Why is this? Testing is almost always perceived as the main bottleneck in any application delivery timeline, and it is universally agreed that manual testing can be time consuming, laborious, repetitive, and costly.

With all this negativity seemingly aimed at manual testing, why is it still the mainstay of application development quality assurance?

This whitepaper looks in more detail at manual testing, and attempts to understand why it is still so prevalent in such an automated age. Finally, as we all know testers do not grow on trees, we will ask if the business can tackle the issue of manual testing in a different way other than the traditional response of throwing more resources at it.

And as it happens, it can.

## What is Manual Testing?

It will not surprise you to know that manual testing is the oldest form of software testing. It may surprise you however, that despite the rise of software test automation solutions, manual testing still accounts for at least 80% of all testing carried out today.

Manual testing requires the tester to perform manual test operations on the test application without the help of test automation software. Manual testing can be a laborious activity that requires the tester to possess a certain set of qualities; to be patient, observant, speculative, creative, innovative, open-minded, resourceful, unopinionated, and skilful.

Manual testing is carried out by a variety of people, from developers and QA through to Business Analysts and end users and helps discover defects related to usability testing and user interface testing areas. While performing manual tests the software application can be validated as to whether it meets the various standards defined for effective and efficient usage and accessibility. For example, the standard location of the OK button on a screen might be on the left and the CANCEL button on the right. During manual testing you might discover that on some screens, this is not the case. This is a new defect related to the usability of the screen. In addition, there could be many cases where the user interface is not displayed correctly on screen and the basic functionality of the program is not correct.

There are a number of advantages to manually testing everything: The entire surface of the product can be covered (albeit superficially); when something unexpected happens it is easily followed up; there is little planning needed and technology issues and there are no lengthy set up issues or ongoing maintenance required to keep the test cases up to date with changes in the application.

However, all is not necessarily rosy in the manual testing garden...

Repetitive manual testing can be difficult to perform on large software applications or with large datasets. It just gets too complicated. Testing in subsets of the whole application can help ease this burden, but it will still be complicated and arduous.

A manual tester would ideally perform the following steps for manual test:

1. Understand the business/functional requirement
2. Prepare the test environment
3. Execute test case(s) manually
4. Verify results
5. Record the result (pass/fail) & Record any new defects uncovered during the test execution
6. Make a summary report of the pass/fail test case
7. Publish the report

These eight steps are valid if the tester is conducting pre-designated tests of an application. But what would it mean if the tester simply stumbled across a fault while doing some ad-hoc testing? Perhaps the steps would look a little more like this:

1. Stumble over an error
2. Go back a few steps, trying to remember the exact steps that brought about the error in the first place
3. Attempt to recreate the error from memory to ascertain that it was a true bug in the system
4. Attempt the recreation again because memory isn't what it used to be!
5. Succeed in recreating the steps, and have ascertained it is actually a bug in the application
6. Ring up developer and tell them about the bug
7. Recreate the error again, this time printing each screen as you go in an attempt to record what you are doing.
8. Walk to developers' desk and hand him pile of screenshots and hand written notes about the error. Or, if the development team is not in your office, email, scan or fax the print-outs over to them.

When there are hundreds of bugs to be found in a single cycle, no wonder testing is looked upon as such a bottleneck in application development.

There are other issues too. Unless a tester spends as long again documenting exactly what they have been doing, traceability and audit compliance will be extremely difficult. Then there is the issue of re-testing, and of visibility and sharing the knowledge between the tester and other interested third parties which may be geographically dispersed.

Then there is the issue of personnel. The more manual testing you do, the more people you need to do it. Testers do not grow on trees, they are not cheap to hire, and they need constant training. But we are not talking just about testers here; end users and business analysts may cost the business even more money and have limited availability. No department head is going to want their people spending hours or even days testing IT applications when they should be working on their day-to-day activities.

The other option here is to allow a third party offshore test agency to do your testing for you. High agency fees plus a lack of control could make this a difficult decision.

Figure 1 below offers a glimpse into the financial reality of manual testing. To be able to illustrate such costs we have had to make a number of assumptions:

- A total test team of 5 developers and 2 QA. With 5 Business Analysts and Users also being utilized for UAT
- Hourly rates of \$40 for QA and Developers and \$60 for Business Analysts and Users
- Each person involved works 8 hours a day, 220 days a year
- Throughout the year, each group of people spend a certain % of their time testing. Which obviously varies per group (see Figure 1 for details).
- 80% of the total amount of testing is carried out manually.

By making simple calculations we can quite clearly see the substantial annual direct costs of manually testing software applications.

Figure1: The True Cost of Manual Testing

Number of Developers	5	Hourly Rate (\$)	40	% of time spent testing	45%	Cost of all testing (\$)	158,400
Number of QA Staff	2	Hourly Rate (\$)	40	% of time spent testing	75%	Cost of all testing (\$)	105,600
Number of Business Analysts	2	Hourly Rate (\$)	60	% of time spent testing	20%	Cost of all testing (\$)	42,240
Number of Users (for UAT)	3	Hourly Rate (\$)	60	% of time spent testing	10%	Cost of all testing (\$)	31,680
<b>Cost of all testing (\$)</b>	<b>337,920</b>						
<b>% of which is manual</b>	<b>80%</b>						
<b>Cost of manual testing (\$)</b>	<b>270,336</b>						

Ouch. For a relatively “normal” size test team, the annual direct costs of testing applications manually are over \$270,000. Which ever way you dress that up, that is a lot of money. And that is only direct costs of the testers, what about the indirect costs of insufficient testing due to illness or time pressures? And the opportunity costs of the end user testing?

However, despite this there is a valid place for manual testing in many organizations world wide. Some of the reasons are looked at on the next page.

## Manual Testing is Here to Stay!

There is no complete substitute for manual testing. Manual testing is crucial for the thorough testing of software applications. Although ways of automating this process have been available for over 20 years it is often not appropriate or convenient to automate:

### No Alternative on Business Critical / Heavily Tested Software

Many companies revert to manual testing of applications if they are critical to the business. Such applications often undergo change and improvements on a regular basis and as a consequence traditional automation tools struggle to adapt and keep up with such change\*. Because of traditional automation's inability to cope with change very well it often leaves companies with no alternative other than continue to test manually. It is often easier and of less risk to the business.

### New to Testing

Those that are new to testing may not want to dive right into complex automation tools. Sometimes its better to walk or even crawl before you run. But that's another story that is touched on later.

### A Quick First Look

After an application has been built and initial development has been finished it makes more sense to have a quick look at the quality of the work manually, rather than spend time writing automation scripts.

### Don't Believe the Hype

Many people still find that despite investing heavily in script base automation solutions they can only automate 10-20% of their total testing requirements. What happens to the rest? You guessed it.

### Full automation not appropriate.

Test automation tools lack the ability of decision-making and recording any unscripted discrepancies during program execution. It is recommended that one should perform manual testing of the entire product at least a couple of times before actually deciding to automate the more mundane activities of the product.

\*For more information on traditional automation tools failure to cope with changing applications please read "The Great Software Testing Swindle" from Original Software.

## There Must Be a Better Way?

In the case of manual testing, time is definitely money. So the question on all QA Manager's lips must be "How can I manage my finite testing resources, without compromising quality?"

We have all seen the reasons why manual testing will still be around for ever, so automation in its traditional sense is not the answer. What is needed is a specialist helping hand for manual testing; easy to use, with minimal training that can run in the background without interfering with day to day testing, and can cut the time spent documenting and recording the errors that are found.

Sounds too good to be true?

Well, consider TestDrive-Assist, from Original Software

TestDrive-Assist is a totally new concept in testing that has been specifically designed for improving manual testing efficiency. By softly tracking applications in the background, TestDrive-Assist offers testers a unique and powerful way of detecting and reviewing errors, sharing knowledge with other team members, and producing audit quality diagnostics and reporting.

Users of TestDrive-Assist consistently enjoy an average of 31% time savings on their manual testing projects. So, let's revisit what a typical manual test might look like, this time using TestDrive-Assist:

1. Stumble over an error
2. Luckily TestDrive-Assist was working in the background and has captured all the steps (with screen shots, and detailed mouse movements/keystrokes)
3. If there are any comments you want to add to screens, these can be added via TestDrive-Assist's unique Markup function
4. Check the spelling and any links for errors using TestDrive-Assist's spell check and link check function
5. Print the audit quality report or save it as a pdf file
6. email the report to the developers
7. The developers, having all the information they need to fix the error, fix it quickly.

So we can see that using TestDrive-Assist can make it much easier to record, save and distribute information about errors that have been found manually. But how much can it actually save?

Studies carried out by Original Software have suggested that TestDrive-Assist can save on average 31% of manual testing time. Based on the following breakdown of testing activities:

Tester activities (broken down as a % of total testing activities):

- o Test planning 10%
- o Visual layer testing 50% (TestDrive-Assist can save 10%, so 5% net)
- o Database testing 10% (TestDrive-Assist can save 90%, so 9% net)
- o Defect reporting 20% (TestDrive-Assist can save 60%, so 12% net)
- o Test documentation 10% (TestDrive-Assist can save 50%, so 5% net)

This gives us our overall saving of 31%. For developers, not all of these areas apply, so we tend to see 15-17% savings.

So what does all this mean for the total cost of manual testing? Simple. Test Drive-Assist saves you money. Lots of money.

Fig3. TestDrive-Assist Saves Money, Lots of Money.

<b>Cost of manual testing (\$)</b>	270,336
<b>Cost with TestDrive-Assist (\$)</b>	204,272

By calculating the savings (i.e. 17% less cost for the development team, 31% less cost for each of the other teams) we can see that the same development/test team we looked at earlier can reduce over \$66,000 off of their manual testing bill. Nice!

So there you have it. With a cost of \$3000 for a single license it seems that to any organization that is doing any amount of manual testing, TestDrive-Assist makes sound business sense. The savings can be considerable.

Lets quantify that another way. Using TestDrive-Assist the same group we have analysed here could cut a 10 week test cycle down to 7.30 weeks. That means the group would be able (if they wanted) to work on another 2 projects over the course of a year. In a business environment where businesses are increasingly turning to their IT departments to provide a competitive edge, this could have large positive ramifications for the business In general.

## TestDrive-Assist: A Fast Start to Full Automation.

Previously there has been no sensible alternative to full automation of your software testing processes. It has been really an “all or nothing” decision. That is up until now.

By implementing automation when it is right for you, not just when it suits your vendor, you only purchase what you need, and will be in a position to fully take advantage of its many benefits, without breaking the bank or having expensive shelf ware.

We call this approach Crawl, Walk, Run.

The manual testing scenario that we have looked at above, coupled with an effective test planning tools (like TestPlan) is what it deemed the “Crawl” phase. Effective planning and more efficient communications, along with the dramatic savings in manual testing will allow you to have more time to continue process improvements which in turn will allow you even more success and time/cost savings.

The “Walk” phase concerns itself with your Test Data Management (TDM) which is fundamental to the success of any testing strategy and is an area where technologies can often delivery fast payback. Effective test data extraction from live databases will allow you to do meaningful tests on real data without compromising live information.

Our TestBench solutions have unique database and server integration capabilities. This enables central storage of all test scripts, results and data on Microsoft SQL Server, Oracle and IBM DB2 databases. It means that testing resources can be accessed and shared by the entire QA or user testing team - and your server becomes a repository of your team's testing 'knowledge'.

Such control of test data ensures that every test starts with a consistent data state; and with strong data scrambling, sampling, manipulation and archiving capabilities you will benefit from more accurate testing with reduced disk space requirements.

The final step (“Run”) in our methodology is the transition to full automation. Effectively deployed next generation software automation solutions (such as TestDrive) can have a significantly positive effect on functional and regression testing timescales. We deliver a unique solution that solves the twin issues of complexity and maintenance - areas where other vendors have failed. Our totally code free, state of the art user interface empowers the subject matter experts, enabling them to define and execute sophisticated tests unhampered by complex programming languages. Our ground breaking self healing technology enables you to re-use your testing investment for new releases and upgrades of your application - allowing you to benefit from minimum maintenance and maximum reuse.

## About Original Software

Original Software offers next generation automated software testing and quality assurance solutions that deliver tangible benefits across a wide range of IT and application environments. As a recognized innovator, Original Software's goal is to reduce business risk and improve application time to market for IT departments through the development of class leading automated solutions.

Over the last 10 years, more than 400 organizations operating in 25 countries have come to depend on Original Software for their software testing solutions. Current users range from small software development shops to major multinationals, including: Cargill Global Financial Solutions, Circuit City Stores, Pfizer Pharmaceutical, BP, DHL, Coca-Cola, Skandia and hundreds of others.

Original Software operates central offices near Chicago, and London. Their solutions can be obtained through these offices or through a network of qualified and knowledgeable business partners throughout Europe, the Middle East, Australasia and the Americas.

**[solutions@origsoft.com](mailto:solutions@origsoft.com)**  
**[www.origsoft.com](http://www.origsoft.com)**



**Original Software**