Bloor

White Paper

White Paper by Bloor
Author **Philip Howard**
Publish date **September 2015**

Automated testing: coping with change

“The truth is that you can skimp on automated testing and deploy manual testers because you think it is cheaper (it isn’t) or because it is easier to get operational rather than capital budget, but this is simply short-sighted.”

Author **Philip Howard**

Automated testing: coping with change

Change is a constant in both development and testing. In development environments this has, over the years, resulted in the use of agile approaches and, more recently, what has come to be known as "cloud first". That is, the idea that you aim to have multiple small releases and enhancements to your applications rather than larger, more occasional releases. Actually, we would argue that this really originated with open source projects but, in any case, it is a subset of "continuous delivery". Whatever it is called, the idea is that by focusing on incremental improvements to an application you are less at the mercy of changes to requirements. Of course, this is not a panacea: it only applies to upgrades and improvements, not to green-field developments, though agile development can reasonably be regarded as supporting continuous delivery.

There is a distinction between "cloud first" and "continuous delivery" in that the former emphasises development whereas the latter refers to the whole software development lifecycle, including testing, provisioning, and so on, as well as development. And not forgetting that there are a myriad of tools that you might want to use that need to be linked automatically and without the need for manually scripted integration. However, while this is the broader context, in this paper we are going to discuss the impact of change on testing within the context of continuous delivery. This is an issue that has not historically been well addressed.

Much testing continues to be manual and **Figure 1** illustrates some of the costs associated with that practice. Moreover, it should be obvious that manual processes are going to be equally deficient when it comes to managing change. Nevertheless, there are automated testing frameworks available on the market and here we want to discuss how these cope with new and amended requirements. In fact, one of the arguments against using automated testing frameworks has historically been precisely that they haven't been good at managing change. We are going to argue that, with the right tools, it is actually possible to automate the change process as a part of the testing environment and thus to enable continuous delivery.

In practice there are various issues to consider:

- The need for **test automation frameworks** to be able to respond to constant user demands. We can call this "responsive automation".
- **Reusability.** By building up a library of reusable test assets functionality can be tested more rapidly by selecting components from this library.
- **Traceability.** In order to automate change you need all the data, expected results and test scripts to be automatically updated through traceability back to requirements.
- **Impact analysis.** Simply implementing a change is one thing, but you need to understand how this might impact on other parts of the system, because the former can break the latter.
- **Speed of delivery.** To keep up with the competition, companies need to get new applications and upgrades to market faster. Testing cannot be a roadblock on this path.

We will discuss each of these issues in turn.

Figure 1: The Price of Failure

OVER **60%** OF IT PROJECTS FAIL
Standish Group's Chaos Manifesto 2013

ONLY **69%** OF FUNCTIONALITY IS DELIVERED
Standish Group's Chaos Manifesto 2013

$46bn SPENT ANNUALLY FIXING DEFECTS
IIBA, 2013

56% OF DEFECTS STEM FROM AMBIGUOUS REQUIREMENTS
Bender RBT, 2009

CAUSING **80%** OF DEFECT COSTS
Bender RBT, 2009

UP TO **50%** OF TIME SPENT LOOKING FOR TEST DATA
Grid-Tools' (acquired by CA Technologies) experience working on site

70% OF ALL TESTING STILL MANUAL
Bloor Research, 2014

WHAT DIFFERENCE HAS CA AGILE REQUIREMENTS DESIGNER MADE?

95% REDUCTION IN DEFECTS
Grid-Tools' (acquired by CA Technologies) Press Release

30% REDUCTION IN TEST CYCLES
Grid-Tools' (acquired by CA Technologies) experience working on site / performing audits of test cases

100% COVERAGE FROM 12 TEST CASES GENERATED
Audit at a large financial services company

Responsive automation

Responsive automation

All testing environments have to be able to react to changing user demands. Moreover, it is typical that change requests are both frequent and never ending. The issue arises as to how you react to these requests in a timely and efficient manner. The short answer is that you need to reduce manual testing, increase automation and do so in a way that allows you to be more responsive to change. However, it is easy to say this, much more difficult to realise in practice. The question is: how can automation enable responsiveness?

To turn this around: what are you actually looking to achieve? From a testing perspective on application change requests, what you would like in an ideal world is automated derivation of all the test cases you need to ensure adequate coverage, generation of the relevant test scripts, and the automated provision of appropriate data to run against those tests. In fact, if we really want to be idealistic, you would like this to be a one-click process. And this isn't entirely blue sky thinking. It is not difficult to imagine artificial intelligence and machine learning capabilities being built into test automation frameworks that start to move testing in this direction.

However, we are not there yet and, in the meantime, at least some degree of manual intervention is going to be required. The question is, therefore, how to minimise this requirement? And the first part of any answer to this conundrum must be that requests for change, and the details thereof, are captured in some sort of formal manner. There are actually two (perhaps three) considerations here. Firstly, the definition of the change requirements should be directly usable at the start of the automation process. Secondly, the process of capturing these requirements needs to be understandable not just to developers and testers, but also to the business users that are commissioning the changes. If this is not the case then there is too great a risk that what the developers are creating will be different from what the user wants. Thirdly, preferably, this whole process should be easy to use and not require detailed training.

“The short answer is that you need to reduce manual testing, increase automation and do so in a way that allows you to be more responsive to change. However, it is easy to say this, much more difficult to realise in practice. The question is: how can automation enable responsiveness?”

The key is the first point: changes are formally captured. Software should then identify what test cases are required to validate a change made to an application and search the existing library of test cases to see if suitable test cases already exist and, if not, to generate new test cases to be stored in the library for future use. Notice that this implies some sort of test case management software. If suitable existing test cases exist then they should have test scripts already associated with them, along with profiles of the data required to run those tests and links to where that data resides. If those test cases don't exist, then you want the software to generate the test scripts and data profiles at the same time as you generate the test scripts.

Put all that together and you genuinely have the ability to be responsive to change.

Automating reusability

Testing is all too frequently treated as a series of unrelated processes: you have some code to test, you design test cases, write test scripts, define the profile of the data needed for your tests, identify where that data is, and describe the expected results. If the data is not easily available you may have to use the facilities of a service virtualisation tool in order to capture and/or simulate appropriate data.

In any case, these steps are typically considered as a part of a single process that is isolated from other such processes. Needless to say, test cases and their associated test components are typically stored for potential reuse but how much reuse really goes on? Of course, this has been a bugbear in development circles for decades: everyone recognises the theoretical benefits of reuse but making it happen is another matter entirely. However, it is potentially easier to implement in testing than it is in development. This is because test cases can be generated directly from requirements whilst that is not generally the case for application software.

The key point to supporting reusability in a testing environment is software that will identify what test cases (along with the scripts, data and expected results) are relevant to the particular software being developed and which can scan an existing library of test cases to identify if a test case already exists and, if not, will create and store it for future use. In other words, reusability needs to be automated: simply creating a library of potentially reusable test components will not be sufficient because we know that human nature means that it will not be properly utilised. Worse, you end up with more and more test cases, which makes the identification of reusable components even more difficult, meaning less and less reuse. So, test case management needs to be automated.

However, it isn't simply a question of reusability for new test components; you also need to cater to the fact that there will typically be (tens of) thousands of existing assets. These will need to be scanned by the test case management software so that you can identify both duplicates and out-of-date test components that are no longer valid. It would probably be sensible if you could identify where test cases were simply versions of an underlying, more fundamental test case. In any case you need software to help you perform governance against your existing test assets. If you were running this in stand-alone mode you would then want the ability to compare any new test case with what already exists. However, in a truly automated environment you would want the software that captured your requirements to automatically look for relevant test cases in your repository, only generating new test components if these were not already available.

In practice, the total automation described is not available, but this is the direction in which the market is, and should be (in our opinion), moving.

Traceability

Change is a constant and that creates problems for testing environments. In particular, there is a problem with test components and particularly test scripts, especially if these are written and maintained manually. This is because the cost of manually maintaining scripts can be prohibitive. In fact, and we can generalise here – not just to testing but to any sort of development process – maintenance, especially manual maintenance, is to be avoided if at all possible. For example, we spoke not so long ago with a company that had so many ETL (extract, transform and load) scripts – tens of thousands – that the department literally had no time to do anything other than to maintain those scripts.

The question is therefore how to avoid manual maintenance or, at least, to minimise it (even in an automated environment you will need some sort of manual oversight)? The short answer, beyond saying simply "automation", is that you need traceability from requirements, through test cases and test scripts, to the data and your expected results. And it is only if you have this traceability right through the environment that you can successfully expect to implement automation that will take away many of those expensive manual processes.

What does that mean in practice? It means that when a requirement is changed then relevant amendments are automatically generated (or retrieved if you have appropriate test cases in your library) for all the subsequent steps in the testing process: the test cases, the scripts, the data that needs to be run through the tests, and the results that you expect from those tests.

Achieving this is not as simple as stating it. In reality you are going to need an integrated suite of tools that starts with requirements capture and test case and test script generation, combined with test data capabilities. In this latter case you will want test data management for in-house data but will need to integrate with service virtualisation for third party data or other data that is not easily accessible. This suggests that point products will not be suitable as these will only resolve, at best, a part of the problem. As an aside, and taking a broader perspective – from requirements through development to testing and provisioning – then we are talking about an integrated suite of products that combine to provide continuous delivery.

Going back to the testing environment, if we are assuming that traceability is implemented throughout, then everything depends on the original requirements, or changes thereto. This in turn means that requirements need to be captured in a formal manner in some sort of model (where the word "model" is used

"Change is a constant and that creates problems for testing environments. In particular, there is a problem with test components and particularly test scripts, especially if these are written and maintained manually."

…fewer, smaller changes are less likely to disrupt an existing system. However, regardless of the approach taken you would like to be able to know what impact any particular change might have on the rest of the application.

here in its most abstract sense) so that when you make a change to the model then everything else is automatically updated by virtue of the traceability back to the model. Note that this doesn't necessarily mean generating new test cases, it may mean recognising that there is an existing test case that can be reused to support the current scenario.

What is required is joined-up thinking or, more accurately, joined-up product suites. This should include test case management (managing reusable testing assets) as well as the other capabilities discussed.

Impact analysis

Supporting changes through a test automation framework is one thing but it's not the whole story. Changes in themselves can have implications beyond their obvious scope. It is entirely easy to make what seems like an innocuous little change only to find that the whole application breaks. The risk of this happening tends to be proportional to the complexity of the application you are changing – the more complex the application, the more likely it is to collapse – the last straw on the camel's back.

This is one good reason to adopt a style of application upgrades that focuses on incremental upgrades rather than major releases: fewer, smaller changes are less likely to disrupt an existing system. However, regardless of the approach taken you would like to be able to know what impact any particular change might have on the rest of the application.

In principle, the knock-on effects of making a change should be captured and handled by the developers of the application in question but, in practice, this will often be left to testers. However, how do testers know what impact any particular change might have elsewhere? In practice, the simple answer is that they don't. In reality, it is more or less impossible to catch unintended consequences if you are using manual testing methods because you won't be able to see linkages across the application. There are many documented cases of companies implementing new systems where these have failed precisely because of unforeseen consequences. The most well-known are those that bring down company web sites for days or weeks, costing not just revenues but loss of prestige and, in some cases, fines.

Conversely, an automated test framework should be able to identify any implications of a change, provided that it has been used to capture the entire application with all of its requirements. Then, when a change is made to those requirements you should be able to perform impact and dependency analyses to see how these changes will impact on the rest of the system. If you are going to assess these manually then ideally they should be available in graphical format (we would recommend actually using a graph) as well as in a more tabular manner, to suit different users' preferences. However, better yet, what you would like the software to do is to identify all the relationships and dependencies that are altered because of this change, and then generate (or retrieve from a library) all relevant test cases, scripts and so on. This should mean that not just the direct effects of a change are tested but also its indirect effects.

Of course there is a coverage issue here. Typically not everything gets tested. But this is because of the time and manpower required for testing, especially manual testing. Automation offers the promise of exhaustive testing. If you test everything then you'll know that everything works. Test less than everything and you won't.

Speed of delivery

Consider Uber. Its service is challenging traditional markets for taxis all over the world. Love it or hate it, it is disruptive. And similar things are happening across industry sectors. In particular, customer facing applications are rapidly evolving, with companies adopting cloud-first (or, more broadly, continuous delivery) development cycles whereby new releases come out every quarter. They don't have lots of new features in each release – they are incremental – but they rapidly accumulate new features and functionality. This is the world you live in and the old "we'll outsource development because it is cheap" model no longer works except perhaps for some back-office applications. Thus, for many applications, time to market and speed of delivery is crucial. However, that can't be at the expense of bugs and functions that don't work so proper testing still needs to be done, but it needs to be done in such a way that does not slow down release cycles.

How do you achieve this? One answer would be to hire more testers. A lot more. An alternative would be to make existing testers more productive. We recommend the latter but how is this to be accomplished? This isn't a complex question – the answer to making workers more productive has been the same for more than 200 years – you make workers more productive by giving them tools that help them work more efficiently. Specifically, it is tools that help to automate some or all manual processes – whether it's the Spinning Jenny or the production line – that enable improved productivity. In the case of testing: test automation frameworks.

In effect, testers should be the operators of test automation tools: leaving the routine tasks of identifying what test cases need to be run, the generation of the relevant test scripts and so forth, to be handled by the automation software. Testers then become like DBAs: they are managing the testing environment, resolving any issues that arise, focusing on high level problems where genuine expertise is required, liaising with developers and users, and so on. This is how we envision the future, but we are not there yet. While test automation framework vendors are now truly attempting to grasp the automation nettle in a holistic way, fully functional, fully integrated product suites are not available yet so there will be a gradual evolution for testers, which will give them time to adapt and to learn new skills. We do, however, believe that the day of the traditional tester is numbered: not just because the technology is emerging that can automate many testing tasks but because the market requires application delivery in timescales that simply can't be met through traditional manual testing.

"...the answer to making workers more productive has been the same for more than 200 years – you make workers more productive by giving them tools that help them work more efficiently. Specifically, it is tools that help to automate some or all manual processes."

Conclusion

Automated testing frameworks are not what they were even a couple of years ago. Then we were looking at disparate, poorly integrated point solutions that addressed a bit of the testing environment but were not in any sense holistic. In that environment you could see some sense in the argument that maybe a manual approach, at least in some areas, had benefits over adopting automation. In our opinion that point of view is no longer valid and, within three to five years, it will be completely discredited. We expect to see a considerable leap forward as more and more testing becomes automated.

The truth is that you can skimp on automated testing and deploy manual testers because you think it is cheaper (it isn't) or because it is easier to get operational than capital budget, but this is simply short-sighted. You might get lucky and never have the sort of outages that some organisations are infamous for, but you probably won't. Your competitors that adopt automated testing frameworks will get to market faster than you can and with applications of higher quality. The truth is that you need to get that competitive advantage before they do.

FURTHER INFORMATION

Further information about this subject is available from
www.bloorresearch.com/update/2289

About the author

PHILIP HOWARD
Research Director / Information Management

Philip started in the computer industry way back in 1973 and has variously worked as a systems analyst, programmer and salesperson, as well as in marketing and product management, for a variety of companies including GEC Marconi, GPT, Philips Data Systems, Raytheon and NCR.

After a quarter of a century of not being his own boss Philip set up his own company in 1992 and his first client was Bloor Research (then ButlerBloor), with Philip working for the company as an associate analyst. His relationship with Bloor Research has continued since that time and he is now Research Director focused on Data Management.

Data management refers to the management, movement, governance and storage of data and involves diverse technologies that include (but are not limited to) databases and data warehousing, data integration (including ETL, data migration and data federation), data quality, master data management, metadata management and log and event management. Philip also tracks spreadsheet management and complex event processing.

In addition to the numerous reports Philip has written on behalf of Bloor Research, Philip also contributes regularly to ***IT-Director.com*** and ***IT- Analysis.com*** and was previously editor of both ***Application Development News*** and ***Operating System News*** on behalf of Cambridge Market Intelligence (CMI). He has also contributed to various magazines and written a number of reports published by companies such as CMI and The Financial Times.
Philip speaks regularly at conferences and other events throughout Europe and North America.

Away from work, Philip's primary leisure activities are canal boats, skiing, playing Bridge (at which he is a Life Master), dining out and foreign travel.

Bloor overview

Bloor Research is one of Europe's leading IT research, analysis and consultancy organisations, and in 2014 celebrated its 25th anniversary. We explain how to bring greater Agility to corporate IT systems through the effective governance, management and leverage of Information. We have built a reputation for 'telling the right story' with independent, intelligent, well-articulated communications content and publications on all aspects of the ICT industry. We believe the objective of telling the right story is to:

- Describe the technology in context to its business value and the other systems and processes it interacts with.
- Understand how new and innovative technologies fit in with existing ICT investments.
- Look at the whole market and explain all the solutions available and how they can be more effectively evaluated.
- Filter 'noise' and make it easier to find the additional information or news that supports both investment and implementation.
- Ensure all our content is available through the most appropriate channels.

Founded in 1989, we have spent 25 years distributing research and analysis to IT user and vendor organisations throughout the world via online subscriptions, tailored research services, events and consultancy projects. We are committed to turning our knowledge into business value for you.