Achieving business benefits through automated software testing

By Dr. Mike Bartley, Founder and CEO, TVS (mike@testandverification.com)

1 Introduction
During my experience of test automation I have seen a variety business benefits.
- Improved quality through both the regular running of the automated tests and having more time for running manual tests.
- Reduced costs as automation reduces the manual effort and thus the cost of testing.
- Reduced costs and improved time to market by finding bugs earlier. The earlier a bug is found, the quicker (and cheaper) it is to fix. The ability to regularly run an automated suite should ensure some\(^1\) bugs are found sooner.
- Improving employee motivation by reducing the amount of tedious manual testing.
- Improved time to market by getting test results sooner as the results from an automated test suite should be available sooner than the manual equivalent.
- Stabilising the software sooner and being able to access the stability of the software through the regular execution of an automated regression test suite. The automated test suite should provide a history of failure rates and can also be a means to generate structural coverage information.

In my experience a test team can make a significant contribution to the business through efficient and effective testing, and automation can increase that contribution if done well (or actually decrease if done badly!). In this paper I will describe my experiences of test automation in various organizations over the past 20 years. However, one particular case study will be used and this is described in detail below.

2 Case study
The software under test in this case study is a commercial product that is a mixture of
- IDE (an Integrated Development Environment) which included a compiler for an extended C language.
- Run-time libraries running on a host server (under various favors of Linux and Windows) driving an external processor (to accelerate the host application).
- Code (which was developed using the IDE) running on the external processor
- Host applications running on the host using the peripheral for support functions.

The external hardware consisted of a board and chip (also developed within the company) connected to the host computer using a PCI connection. This is shown in Figure 1 “Case study software under test”.

\(^1\) It depends on what tests are automated as to what sort of bugs are found. It is unlikely that every possible bug will be found via an automated test suite.
study software under test”. The product (software + hardware) provides a math coprocessor offering accelerated math functions through multiple processors on the chip. The run-time libraries running on the host intercepted calls to the math libraries on the host computer and directed them to the external hardware in those situations where the software decided that the hardware would accelerate the math function.

One major challenge that the test team faced was the diversity of the customer environments:

- Multiple operating systems were supported (32 and 64-bit versions of RedHat and SLES, Windows XP 32 bit, Windows Server 32 bit, Windows Compute Cluster 32 bit, Windows Server 64 bit (thunking))
- Numerous host computer platforms (various Intel and AMD chipsets in computers from a variety of manufacturers) connected to the external hardware using PCIx and PCIe.
- Different versions of the external hardware.

The above variables generated a huge multi-dimensional matrix of potential combinations and there were obviously too many to test them all. We therefore applied an n-way test analysis to reduce the number of systems to test. However, this still left us with a large number of tests and a large number of systems to run them on, so we had to consider automation.

3 Deciding to automate – it goes to the top!

As the software test manager the business case for automation was an obvious one but we needed to make our business case to senior management (as you would expect in any well run organization). We found that the following points needed to be considered.

- **Quality**: We judged that through automation we could increase the amount of tests we could run and the number of environments we ran the test. This would increase the likelihood of finding any bugs.

- **Time to market considerations**: We estimated the time it would take for running automated tests against the time currently taken for running them manually.

- **Estimating future needs**: We estimated the number of tests that we expected to need in the future based on the product roadmap and our existing view of the holes in the current test suite. We then extrapolated the time to market saving that automation would have.

- **Costs**: We could easily estimate the cost of non-automation given our current levels of manual testing and the estimated future needs. We could also reasonably estimate how much effort it would take to develop the automation capability and write the tests. However, estimating the cost of test maintenance is hard and in my experience this is where most of the effort goes. For example, if there are changes in the software then automated tests will need to change; so you need to estimate the likelihood of change and the amount of effort to change tests to match it.

As mentioned, the software test team already believed the case for automation. But going through the business case had three major advantages: we better understood the challenges (e.g. writing maintainable test would be a key goal); we got agreement and buy-in at a high level of the organization; we set agreed improvement targets for the automation which we could track (time to market, quality,…).
4 How to automate: buy vs. build?

We had already developed an in-house software version control and build system and it was a fairly easy decision to extend that to include test automation. The major advantage we expected to (and did) achieve was to more easily tailor the automation tool to our needs. For example, we were able to integrate the tool with our build process which meant that we only ran those tests which tested changed source code as our build rippled up through the source code dependency tree.

We had one full-time person who developed and maintained the in-house version control and build system and this person’s role was extended to include the development and maintenance of the automated test system. The initial work to add in the basics was about six weeks. This allowed us to add in a test scripts at a node in the code dependency tree and to add a test machine into test machine pool.

- The test scripts defined various aspects of the test as our analysis showed that this would improve maintainability of the test. This is discussed later in the paper.
- The test machine was a particular combination of OS, machine and board from our n-way analysis. This information was held in a file which allowed us to easily add additional test machines.
- We made it easy to remove tests from the test suite which we used when a test needed updating to reflect changes in the software. This proved to be a major benefit as this meant that running the test suite was not delayed by test maintenance (although of course we had to consider the reduced test coverage).

With the basics in place we were able to start adding tests and test machines. We next consider how we managed that process and the on-going maintenance and development of the automation.

5 On-going development and maintenance of the automation tool?

If you develop the automation tool in-house then it should be written by the software development team! Writing the automation tool is basically a software development exercise not a testing exercise. In a previous organization I worked at the test team wrote the tool and maintenance of the tool was very high. The tool was eventually handed over to the software development team who re-wrote it thus improving the functionality and ease of use and reducing the maintenance burden.

If buying in the tool then in my experience installation and maintenance is best done through the IT department who do this all the time with other tools. The installed tool is likely to need configuration to the test team needs and my experience suggests that such configuration is best done by the test team.

6 Developing automation tests

The automation tool was written with a goal to make the writing of new tests and the reuse of old tests as easy as possible. This was achieved by separating different concerns in the writing of the test:

- Test setup: The steps required to set up the tests
- Test execution: The actual test steps
- Expected results: The expected outcome of the test
Comparison method: The method for comparing the actual and expected results. For example, this may allow for rounding errors in numbers.
Test tear down: The steps required to put the machine back into a state that allows the next test to execute.

By separating the above it was easier to reuse tests because the test developer could copy existing tests and just change the parts where the tests differed. However, it was during test development that we also noted a drawback in developing our own automation system. The problem was training internal and external test writers how to write tests that are able to easily be imported into the regression system. However, this is a small price to pay for the flexibility we gained through developing our own automation system.

7 Managing tests and machines in the automation system

Once a test is written then it can be added into the test automation system. It was added into build dependency tree at the highest point possible (i.e. as close to the target as possible). This meant that if nothing it depended upon had changed during a software build then the test would not be executed. This can dramatically reduce the build and cycle time and thus allow a policy of continuous build and test. We did of course keep an option that forced all tests to re-build and run for when we wanted to run a full regression.

Test writers had the ability to add tests into the automation system as they were written. However, the automation system also had a feature that allowed tests to be banned. This is done for tests various reasons:
- It is known that the software has changed and that tests need to be updated.
- A test is known to be giving a false failure (i.e. it wrongly indicates a failure).
- A test is not restoring test machines to a known good state.

Of course, the automation system generated a list of banned tests because the automation results needed to be read in the light of the banned test list. We also have the ability to remove test machines from automation to allow for debug. Often tests fail on a single machine because of some uniqueness in the combination of our software, the machines hardware and operating system etc, or our external hardware. It therefore might be difficult to reproduce that failure and so it might be that the machine was needed to debug the failure.

8 Reviewing results from the automation system

The automation system automatically generated a summary

9 When to automate?

My experience of automation has shown me that one of the key decisions is deciding which tests to automate and which to leave “manual”.
- Complexity of the test environment: The more complex the test environment, the more maintenance the automated tests will require. For example, for the case study

---

2 Even “manual” tests can benefit from some level of automation. For example, scripts to set up the environment before running through a few manual steps.
described above we were required to carefully measure performance improvements gained through use of the product being tested. This required the test environment to be set up very carefully (e.g. to ensure the machine had no background jobs such as virus checkers running), and we found it easier to do the setup manually before running a series of automated performance tests.

Consequently these performance tests were not added to our automation system but were left to be run in a semi-automated way.

- **The level of testing:** My experience in automation is that there are more tools available for supporting unit testing rather than system testing. These tools allow you to more easily build stubs, make calls to the software under test and define the expected results of those function calls.

System tests often require a more tailored setup that is specific to your needs. This was definitely the case with the case study described here. We found that by using our own in-house automation tool we were able to add features that allowed us to create the system test environments generated by our n-way analysis.

We also automated the testing of the graphical user interface (GUI) that came with our IDE testing outside of our in-house tool. My first experience of GUI test automation was from over 20 years ago and at that point the tests were very “fragile”. That is, small changes in the layout of the GUI (e.g. the position of button, the size of a text box) caused tests to fail. However, for the case study we used a commercial GUI testing tool that was very robust in the face of small cosmetic changes in the GUI.

Finally, user acceptance testing is often not a target for automation, because this is often a formal signoff procedure with the customer performed at the customer site. However, the user acceptance tests executed are often very similar to system tests that are first performed in-house to ensure the user acceptance testing goes smoothly. And those system tests can be automated. I have worked on the testing of a large bespoke software development where we mocked up a small version of the larger network that existed at the customer site. We ensured we ran all the user acceptance tests within that environment.

- **How often will the test need to be run?** If a test is going to be run very infrequently then the ROI (return on investment) from automation may not be so obvious.

For example, in the case study described here we needed a suite of tests developed to demonstrate that our floating point library was IEEE compliant. We decided to outsource the development of those tests and we got them delivered as a suite of tests that could be run from a single command. As our floating point code was very stable the tests were usually run as we approached a release and it was felt that the effort to add them into the automated system was not worthwhile.

- **Ease of automating the pass/fail criteria:** Automation requires you to predict the result and have a means to automatically check the actual result matches your prediction. This is not always as easy as it sounds. In the case study described here part of our software signoff was to demonstrate that the combination of our software and external hardware accelerated math packages (such as Matlab or Mathematica) which usually performed their calculations on the host computer.

As mentioned above, setting up the environment for these tests was very complex. However, when working with an external tool we needed to consider any changes in
the tool such as added math functions or changes in performance. Measuring the performance improvement was also complex and often involved measuring both CPU and wall-clock time. We found it easier to perform this semi-automatically rather than full automation.

- **Test stability and repeatability:** If the requirements, specification or design under test is not stable, then the lack of stability can mean that the cost of maintaining the test can just be too high. We often found this where we were developing new functionality to the software. We needed to test the new features as they were being developed but it wasn’t always cost-effective to automate the tests until the software started to stabilize.

In my experience deciding what tests to automate and when to automate them is a crucial decision that can make or break your automation. Get it right and you save time. Get it wrong and you will find yourself sinking vast amounts of time and effort into test maintenance and debugging false failures!

10 **What happened to those tests we couldn’t automate?**

If a test was not suitable for the automation tool then there were a number of options open to us:

- **Semi-automation:** For example, with our performance tests we found that we needed to make sure our test machines were very carefully setup to ensure we were really measuring the performance of the software (so no background processes such as virus checkers were running). We found that this excluded adding these tests to the automation systems but we were still able to script the execution of the tests and performance comparisons once the machines had been set up.

- **Offshoring:** We had offshore teams that we used to run tests outside the automation system. This included semi-automated and manual tests. Using an offshore team had a number of advantages including cost, resource availability and independence.

11 **The major features of the automation system**

The major features of the automation framework and how it fitted in the overall test strategy are listed below.

- The automation framework was developed in-house by the same team that had developed the in-house software version control and build system. This meant that the two were integrated which meant we only ran the tests which tested the software that had changed. This saved time and meant we could perform a build, integrate and test every night.

- We automated all levels of testing: unit level, integration of the driver and host application code, system and application testing.

- Our environment allowed us to run all of our automated tests across a wide range of host platforms (both different host CPUs and operating systems) with different versions of our external hardware thus automating our compatibility testing.

- We outsourced nearly all of the execution of tests which were not part of our automation. These tended to be higher level customer focused and performance tests which we found harder to automate for various reasons.
- We also outsourced the development of some of the automated tests, but we did find that it was harder to import the tests into our automation environment as the outsource organisation did not have knowledge of our test automation tool and the tool was not developed with easy import in mind.
- We integrated a static analysis tool into the build system. This found a number of potential issues which we fixed over time and were able to ensure they stayed fixed by regular running of the tool.
- Test maintenance was difficult, and we found that we lost knowledge regarding the older tests as people moved around or left the organisation. This can present significant problems when the test fails. This is not the case with manual (or semi-automated) tests, as somebody is always responsible for maintaining and running the tests. However, ownership and handover of automated tests was not so visible and well managed.

12 Obsolete and old tests!

One major issue I have found in all my test automation projects is that once a test is written it often becomes forgotten about. To a certain extent that is what you want – the objective is for the test automation to run smoothly and not have to worry about the tests. However, it becomes an issue if the test starts to fail (for whatever reason). Suddenly it becomes important to understand the test again so you can assess if the software or the test is broken.

Now, if this was a manual test then by definition somebody will understand what it is doing. However, with an automated test this is not the case. Quite often nobody can remember what the test was for or how the test works. The only way that I have found to avoid this is through good documentation of the tests when they get written. One of the best ways to do this is to make the test self-documenting through the use of comments and a high-level language for writing the test. Also, when the test gets reviewed one of the criteria in the review should be the quality of the documentation and the likelihood that an independent person would be able to understand what the test does.

13 Measuring the business advantage!

The automation used in the above project significantly improved time to market of the software: we were able to shorten release schedules and hit the release dates. The quality of the software (as measured by defects in the field) was also significantly improved mainly through a combination of the automated testing and outsourcing of customer-focused testing.

We found it hard to measure the ROI for the tool. The cost of buying the tool and licensing it is easy to measure. It is also possible to measure the cost-saving in terms of time saved through automatically running tests vs. running them manually. You now have to add in the cost of maintaining the tool and the tests. This can give you a figure for the cost and the cost-saving. However, this does not take into account the advantages you should obtain through your use of automation. For example:

- Given that you now have automation, you might write additional tests that wouldn’t have been viable in a manual environment. This is likely to increase the quality of your software which might be hard to quantify in your ROI.
Finally, what value do you put on improved time to market?

**14 Conclusion**

This article has demonstrated both the advantages and issues surrounding the automation of software testing. It has attempted to provide practical solutions to those issues and demonstrated how the automated testing can improve time to market thus allowing the test team to demonstrate a significant business advantage.

The author, Dr Mike Bartley, gained a PhD in Mathematics at Bristol University and an MSc in SW engineering and MBA with the Open University. He has been involved in both software and hardware development for the past 20 years, including outsourcing. He has recently established his own consultancy to help companies in product QA and offshoring.