Automated testing ROI: fact or fiction?

A customer's perspective: What real QA organizations have found.

White paper

Table of contents

About the author: Paul Grossman is a senior staff consultant at Technisource, the IT staffing division of Spherion, one of North America's largest independent providers of IT staffing, solutions, and management services. Paul has almost 20 years experience in testing and quality assurance including nine years designing test automation frameworks. He is an HP Certified Product Specialist, an HP Certified Instructor, and an HP Accredited Software Engineer, HP Quality Center 9.2. Paul has been a popular speaker at HP Software Universe, Mercury World, and The International Conference on Practical Software Quality and Testing (PSQT). His clients have included insurance companies, a medical device manufacturer, and several leading online retailers. He lives in the Chicago area and works with IT quality organizations across the United States.

Automated software testing—expense or investment?

Automated software testing invariably sounds good to IT quality assurance (QA) managers. Intuitively, you see that creating an automated test one time and then running it hundreds or thousands of times will enable you to expand test coverage, find defects earlier, and focus manual test effort where it is really needed. That has to save the company money and reduce business risk, right? But when you submit budget requests for test automation software or services, chief financial officers (CFOs) want more. They evaluate every expenditure based on how much money it will make or save the company, so you must show them what returns the company can expect on the investment.

At Technisource, an HP partner providing QA software and expertise, we have been helping QA organizations implement automated software testing for almost ten years. We have learned that automated software testing can provide an excellent return on investment (ROI). And we have learned some things about how to quantify the ROI to help present an effective business case for automation. That is what this paper is about—quantifying the financial returns of automated testing.

I will first tell you a bit about automated testing—what it does, but also what it does not do. Then we will look at the kind of financial benefits most of our clients derive from automated testing. Much of the financial return accrues to IT, but the biggest returns often accrue to the business when the expanded test coverage enabled by automated testing prevents defects from being released into critical production systems. Quantifying IT and business value requires examination of several key business factors. I will show you what those factors are and how to apply them, and I will pass on some of our experiences as examples you should be able to apply to your situation.

Why automated functional testing?

In functional testing, testers execute applications following test suites designed to exercise and test application functionality. They collect results, compare to what is expected, and report the outcome. Once the test suite passes successfully, you must rerun the test suite every time the software changes—regression testing. In some cases, you have to repeat it for various combinations of operating systems, system configurations, and even different application component integrations. You don't have to do much of that before you realize that much of that testing could have been done easier and faster by a computer.

Automation provides tools that enable testers to create automated test suites that can be run, and the results collected and reported, by a computer. There is more up-front effort required to set up and maintain automated test suites. Tools like HP QuickTest Professional software have a record/playback mode that lets you execute application functions and automatically generate a test script. That's a simple solution for simple problems (and a great way to see how the scripting language works), but more complex tests require that testers develop test scripts, create or import test data,

and maintain the tests as the application changes. Once you have created the test suite, however, it can be rerun hundreds or even thousands of times with little effort.

The best applications for automated testing, then, are those where a lot of repetition and retest is needed. Most test teams can easily identify projects fitting this description, and they target them for early implementation (and quick payback). More mature automation efforts can be an ideal complement to iterative development methodologies where development teams will strive to produce a new, completely functional version of the software every three or four weeks with each iteration introducing new functionality. Automated test suites can quickly test the functionality of the previous iteration. Then testers can incrementally enhance the test suite to test the new functionality introduced in the current iteration. Without this kind of approach, you could end up with a four-week test cycle for every three-week development cycle.

Automated testing does not eliminate the need for manual testing; some things can only be done manually. For example, testing that requires visual verification must usually be done manually. (Does that new logo appear correctly on each screen?) And exploratory testing, where the results of one test prompt the tester to explore related areas, requires human observation and decision making. It is not usually cost effective to automate a test if it will be executed only a few times, and a manual test is the easiest way to quickly verify new functionality.

How much testing can be automated? Each situation is different. A test team might initially target 10 to 15 percent of regression tests for automation. That is a good way to master the new skills and processes required by automation. As proficiency grows, most teams find that 40 percent to 60 percent of tests can be effectively automated.

Automation is not usually a headcount reducer. If you automate 50 percent of tests, that still leaves 50 percent that must be done manually. And automation itself requires some effort. On a recent project with a ten-person test team, we assigned five testers to manual testing, two testers did exploratory testing, and three developed and maintained automated test suites. That is a fairly typical distribution of labor that covers the manual testing still required, and sets QA up for the benefits of automated testing. But if automation doesn't reduce labor cost, what benefits does it produce, and how do we express them in financial terms?

Financial benefits of test automation

We said earlier that automation has the most value in situations where you can develop an automated test suite and then execute it many times. You can dramatically expand test coverage while maintaining the same labor cost. And expanding test coverage translates directly into benefits the CFO will understand.

Expand test coverage at the same cost.

With manual testing each tester works an eight-hour day. When he or she goes home, testing and test systems stop. Automated testing, however, can run continuously to greatly extend the test time available. Further, automation runs many more tests per hour than manual testers. This means we are getting more testing for our test dollar. Let's do the math.

The cost of testing labor varies by company and by region, but a loaded cost of $50 per hour for an entry-level tester is a pretty good average, and a senior test engineer costs about $75 per hour. (If you use contract services, it's even more.) With manual testing, you get eight hours of effort from an entry-level tester for $400. With automated testing, a test engineer might spend all day creating automated tests at a cost of $600, but the tests can then run an additional 16 hours a day (more with additional test systems) for no additional cost.

More significantly, automation runs more tests in the same period. In a man-to-machine faceoff we performed on one project (yes, it was a slow day), we found that automation could run four to five times more tests per hour than a heads-down manual tester. Considering a reasonable work schedule for the man, automation had at least a five-to-one performance advantage. How do we turn this into something the CFO understands?

Let's consider our ten-person test team. Assuming the team is comprised of five entry-level and five senior testers, it would cost about $105,000 per month to keep that team on the payroll (loaded cost for 168 hours per month). For that, we would get about 1,350 hours of testing at a cost of almost $78.00 per test hour. (We have assumed that each tester will average 135 hours per month of actual testing. The rest is lost to breaks, holidays, vacation, sick leave, training, and

other overhead.) If we automate testing, the labor cost per month will remain the same, but for the effort of the three test automation engineers, we get 16 hours a day of testing, and we will run five times more tests during those hours. That totals 5,040 hours per month of (manual equivalent) testing created by the three test automation engineers. Adding 945 hours of manual test and exploratory testing for the seven remaining testers, we now have 5,985 test hours at a cost of $17.54 per test hour ($105,000 divided by 5,985).

Now we're speaking in terms the CFO understands. We have reduced cost per test hour from $78 to $17.54. Or, we have increased testing from 1,350 hours to 5,985 equivalent hours and gained $315,000 worth of testing monthly for the same cost (5,040 times the average hourly cost of a tester). It is the increased test coverage that really pays the benefits. Let's look at what that can mean in most IT organizations.

Reduce the cost to fix defects.

It costs more to fix defects that are found later in the development cycle. A landmark study on the cost of inadequate software testing commissioned by the U.S. National Institute of Standards and Technology (NIST) in 2002[1] reports the cost of fixing defects according to the stage at which they are introduced and the stage at which they are found. This data is used in the "Hours to fix" rows of Table 1. (In this case, the data was developed by surveying companies in the financial services sector developing electronic funds transfer applications. We have applied $75 per hour to derive the cost of fixing defects in each stage. You are welcome to adjust that to represent your actual developer cost.)

The conclusion is: The earlier defects are found, the cheaper they are to fix. It costs almost five times more to fix a coding defect once the system is released that it does to fix it in unit testing. When you provide more test coverage—and you can when some testers are now working 24 hours a day—you find more defects before the release goes into beta or into production. That means real financial savings for the company.

[1] Planning Report 02-3, "The Economic Impacts of Inadequate Infrastructure for Software Testing," Prepared by RTI for National Institute of Standards and Technology, May 2002, p 7-12.

Table 1. Cost to fix coding defects in each development stage

Defect introduced in coding/unit test phase	**Stage found**			
	Coding/unit testing	**Integration**	**Beta testing**	**Post-release**
Hours to fix	3.2	9.7	12.2	14.8
Cost to fix (USD)	240	728	915	1,110

Find "hidden fruit."

Manual testing lets you find low-hanging fruit. In testing terms, that means finding the few critical defects that are easy to locate. Automated testing lets you go beyond that to find hidden fruit—subtle defects that you are unlikely to find through manual testing alone.

Resource leaks, such as memory leaks, are a prime example. Memory leaks manifest themselves over time. They occur when the same code, such as a search, is executed hundreds or even thousands of times, but fails to release all the memory it reserves. Performing that much testing is almost impossible when doing it manually, so that kind of defect often slips through test into production. Similar situations arise with other resources. Applications may consume escalating amounts of central processing unit (CPU), disk, or table space as transaction and data volume grows. Key sorts or table loads that seem instantaneous for a few hundred values can leave business users staring at hour glasses when production volumes grow into the tens or hundreds of thousands. Load testing may find some of these problems, but that is usually later in the test cycle when defects are more difficult and more expensive to fix. Automation allows you to exercise applications with a level of vigor and completeness that is difficult to achieve manually. Here are some examples we have encountered.

Thorough testing of input fields has always been a challenge. What if the user leaves a field blank, enters zeros or negatives, letters rather than numbers, special characters, invalid dates? Manual testers will try to test boundary conditions like these, but the number of conditions to test on every input field becomes overwhelming. We developed an automated routine that tests dozens of conditions on every input field. It is easily included in test scripts, and it can work 24 hours a day enabling us to find defects that would likely go unnoticed until found by production users. Checking for structured query language (SQL) injection vulnerabilities on input fields poses a similar problem. A manual tester can try a few cases, but an automated routine can test many more cases and find defects that manual testers might miss. Once created, such a routine can be easily included in many test suites.

There are other cases that are difficult to test manually. An application we tested had an internal counter set to an initial value of one. Every time the system rebooted, the counter was incremented and then checked to ensure its value was one or greater. Unfortunately, after 64 reboots, a coding error reset the counter to zero, and the system would fail to boot up properly. This appeared to the manual testers to be an infrequent and seemingly spurious application problem that could not be duplicated. With a universal power supply that would auto-boot our test system whenever it powered down, rebooting the system 64 times was no problem for an automated test script. That ultimately allowed our developers to pinpoint the problem. Manual testing has other limits. A manual tester may not immediately notice that a browser has been redirected from the test environment URL to the live site during a test, or that a Web connection is not a secure HTTPS protocol. However, a robust automation framework can continuously watch for that kind of defect or test error.

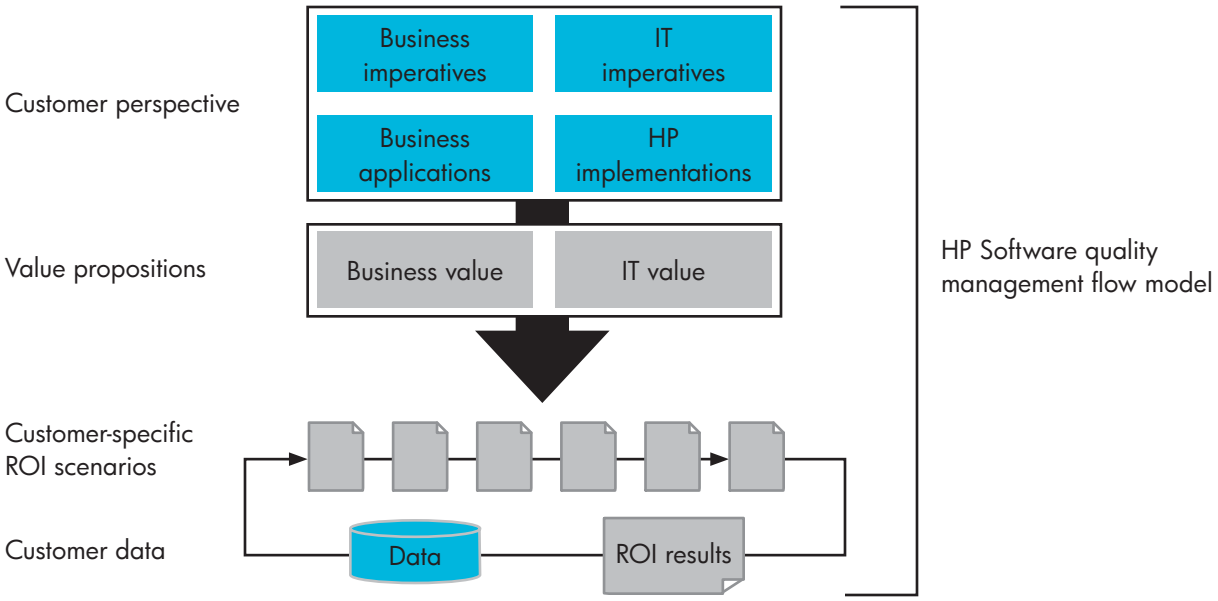
Reduce business risk.

We have seen that the cost to fix defects discovered in beta or in production can be many times the cost to fix them earlier. But the IT cost to actually fix defects that are discovered in production is often the smallest part of the problem. Failures in production software can leave business users sitting idle resulting in lost productivity, or they may result in lost revenue, lost customers, and legal or compliance liabilities.

One of my clients, an industrial product supplier, has calculated the actual cost of downtime on one of their key systems to be $41,000 per minute. That is consistent with other surveys and analyst reports that put the cost between $15,000 and $50,000 per minute. That is for a complete outage. The cost of defects in production would be highly variable and more difficult to estimate. The NIST study referenced above, however, did estimate the total impact of software defects on the U.S. economy at up to $59.5 billion in 2002. That was 0.6 percent of the U.S. gross domestic product. Applying that factor to a corporation's revenue suggests the average cost of software defects in a company with $1 billion revenue is $6 million. According to NIST, more than half these costs are incurred by software users rather than IT. Now we're getting to something the CFO understands!

In your discussion with the CFO, figures like those above will help set the stage, but you must find ways to apply them to your business situation. What are the most critical applications? What is the defect history? What outages and lost productivity have been documented? What might be the impact on the business of defects in production? How many defects might be caught earlier if automation could increase your test coverage by, say, 1,000 percent? These unknowns prevent our applying some simple formula to the question of automation ROI. While we cannot do that, we can supply a framework for thinking about and communicating ROI in software quality.

Figure 1. HP software quality ROI model



The HP software quality ROI model

One thing should be clear from the discussion above: Automated testing can deliver two kinds of value to the organization—business value and IT value. Business value is achieved through avoiding cost to software users and preventing loss of revenue, liability, and other business risks. IT value is achieved through the reduction of IT cost or through increased IT efficiency. I have found HP software quality ROI model[2] (Figure 1) a useful way to help our clients zero in on the financial benefits of automation.

Essentially, the model requires us to examine the business imperatives that prompted the creation of a software application as a way to begin to quantify the value (and therefore the ROI) of a software quality initiative. For example, if the application is an online ordering application, then the business imperative prompting it is sale of product, and its business value is increased revenue. That suggests impact on revenue might be a useful way to quantify the cost of defects in production software. If the software application is an asset management system designed to reduce the cost of asset ownership or to improve the efficiency of business users who manage assets, then the business imperative driving it may have been business efficiency. In this case, the impact of defects might best be expressed in terms of lost man hours or in hours of rework. For financial applications, demonstration of compliance is a key value, and reduction of liability that could be incurred through defects might get management's attention.

The IT value side is similar. If a driving force behind the development effort is time-to-market, then shortening test cycles and reducing the time required to fix defects becomes a key value of automated testing. If better software quality and attendant customer satisfaction is a business imperative of the project, then reducing the number of defects found in production may be a key value and finding hidden fruit a quantifiable benefit.

[2] "Measuring the total value of HP quality management solutions," White paper, March 2008

Applying the ROI model

The ROI model requires that we identify both the business and IT value of an automated test initiative, then apply that to the specific scenarios using the specific data found in your business. Table 2 shows how the financial benefits I have presented above might be applied in various cases.

To see how to apply this, let's look at a few examples. If efficiency is an IT imperative that applies to the development project, you can propose that automated testing will help your QA team find defects earlier in the development process. If you make some reasonable assumptions about how many additional defects might be found in the test stage rather than in beta or in production, you can apply the cost-to-fix information in Table 1 to posit a financial benefit. If your development organization has an initiative to improve the quality of released software, you can propose a reduction in the number or severity of defects released to production, and you might use the NIST cost data to put a rough dollar figure on it. (i.e., if your company's revenue is $500 million, then your share of the U.S. cost of software defects is $3 million annually—$500 million multiplied by 0.6 percent. If you reduce defects by 25 percent, then you can save the company $750,000 per year.) If 25 percent of the $500 million revenue depends upon an application being available, then reducing application downtime by 1 percent could protect $1.25 million in revenue ($500 million x .25 x .01).

Table 2. Applying automation benefits to quantify returns

Scenario	Automation benefit	How to quantify
Common business imperatives		
Revenue	Fewer defects in production	Estimate revenue lost in downtime.
Business efficiency	Fewer defects in production	Estimate man-hours in lost productivity.
Customer intimacy	Fewer defects in production	Estimate number of times customers might experience application downtime or errors.
Business agility	Shortened test cycles	Estimate test effort that can be moved after hours.
Governance and compliance	Fewer defects in production	Estimate increase in number of defects found before release.
Common IT imperatives		
IT efficiency	Find defects earlier.	Estimate improvement. Apply cost factors to show reduced development cost.
	Find more defects with the same test cost.	Estimate value of expanded test effort compared to manual testing.
Schedule	Find defects earlier. Reduce development time.	Estimate improvement. Apply hours to fix bugs to show reduced development time.
Software quality	Fewer defects in production	Estimate improvement. Use defect count and severity or apply NIST factor 0.6% of corporate revenue to quantify.

Figure 2. Add up the potential savings.

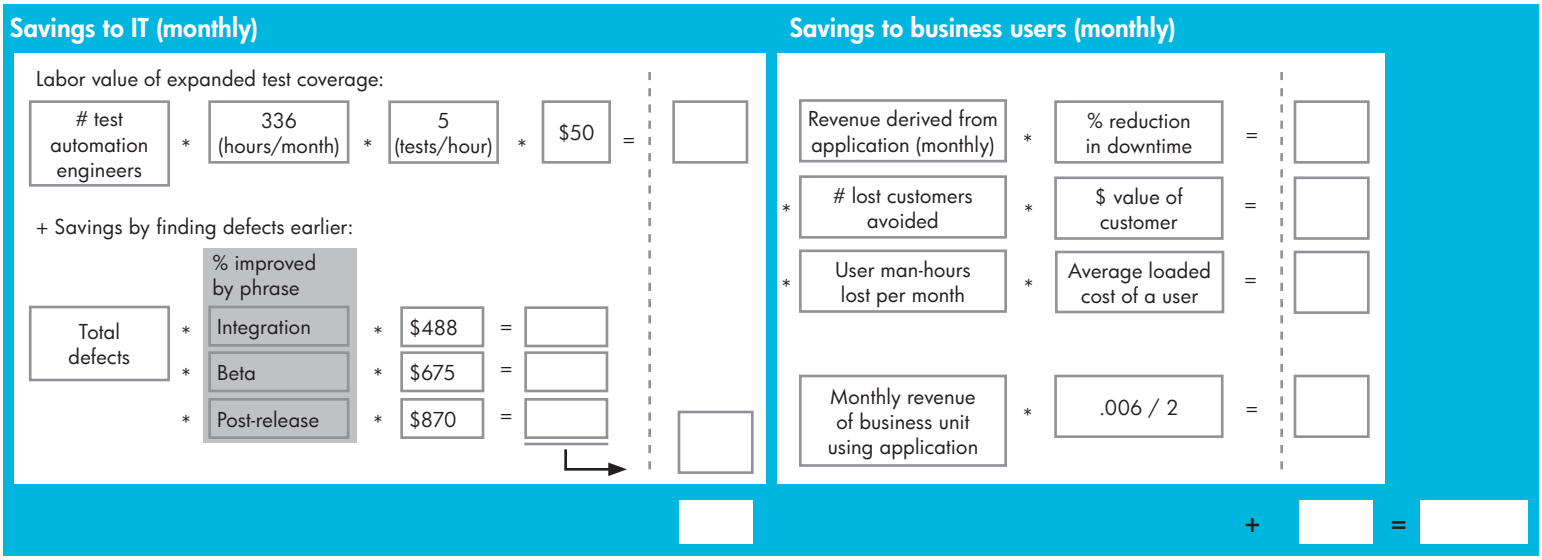


Figure 2 provides a structured approach to identify and quantify the IT value and the business value of test automation. It is based on the financial benefits we have presented above. You will have to determine how they apply in your specific case, establish appropriate assumptions, and estimate the benefits. The more real data you can incorporate from your development and business situation, the more valid the results will be. The intent of this paper is to show you how the benefits can accrue and to give you some ideas on how to value them. Whatever you do, it is almost certain that some ROI data—even with caveats and assumptions—is more useful than no data at all.

Real results

Since QA managers have found they must document the benefits they achieve with test automation, several of our clients have undertaken projects to do just that. One, an insurance company, initiated a program to find defects closer to the point where they were introduced, and documented savings of more than $2 million in development cost. Now, when they submit budget requests for new QA initiatives, their CFO sits up and listens. A medical device manufacturer documented seven defects found through automated testing and valued them at the cost of one customer per error lost to their competition. They coupled this with the cost of a round-the-clock team of manual testers to provide the same coverage to demonstrate a benefit of $2 million. Management green-lighted the test automation project for another year and planned to expand it.

Just as important as estimating the ROI that might occur on an automation project is documenting the benefits that actually do occur. When you do that, your estimates and projections are even more accurate next time, and you have the data to back up your claims.

Learn more.

I have tried to share some of our experience with test automation, show how it has created real financial benefits for our clients and help you communicate to your CFO in language he or she understands. If you believe you may have a good application for automated testing, and you want to begin to reap the benefits available, learn more about HP quality management solutions and about Technisource's services.

For information about HP quality solutions, visit www.hp.com/go/software.

For information about Technisource services, visit www.technisource.com.

Technology for better business outcomes

To learn more, visit www.hp.com/go/quality

4AA2-4175ENW, February 2009