The Business Argument for Investing in Test Automation

Edward Adams
Testing Evangelist

TP329 01/03

Rational®
the software development company

INTRODUCTION 1
TESTING AS AN INVESTMENT 1
THE WRONG APPROACH 1
TALKING ROI 2
CALCULATING COSTS AND ROI 2
ESTIMATING NET QUANTIFIABLE BENEFIT 3
ROI FOR MANUAL TESTING 4
ROI FOR AUTOMATED TESTING 5
LONG-TERM PAYBACK 6
WHAT SHOULD YOU AUTOMATE? 6
TALK BUSINESS TO BUSINESS PEOPLE 7
REFERENCES 8
ABOUT THE AUTHOR 8

Introduction

Eighty percent. By now you have probably heard about this grim statistic from industry analysts like Gartner, The Standish Group, and others. It refers to the percentage of software projects that are late, over budget, lacking functionality, or never completed. These statistics are particularly shocking if you think of every software project as an investment — or a series of investments. Knowing that four out of every five of these investments fail, as an industry we want to have as much confidence as possible that each new investment we make or propose to management is well thought out, and even proven with some metrics or a business case.

Test automation is one investment that has produced tremendous results for many software development organizations; yet it is still misunderstood or misapplied by many others. Test automation is neither a panacea for all testing problems nor a replacement for skilled testers. And it cannot do all the work for your testing team. Instead, it is a long-term investment that, when used in conjunction with software engineering best practices, pays dividends over and over again. The critical fact to remember here is that test automation is both a long-term investment and a software development project in itself. As such, when creating your test automation code, you should employ the same best practices you would apply to any software project: logical, modular designs; configuration and change management; well chosen documentation; and comprehensive testing — for example, using code coverage to estimate test effectiveness before and after automation— and so on.

Testing as an Investment

Testing is the only software quality activity that directly observes and reports on how the system actually performs. It can increase — or decrease — confidence in all the things a system is supposed to do; in fact, that is testing's greatest contribution to a project. Testing gives the project team the ability to say, "Yes, this product is ready to go," or "No, it isn't ready, and here's how bad it will be."

Essentially, testing is a defect discovery activity; however, *assessing the results* of that activity has a direct effect on our confidence level. The effect, of course, could be either an increase or decrease in confidence regarding product readiness. So through testing we find bugs, but that is not why testing will get funded. The processes of software testing and software development are fundamentally different. Development produces an artifact that automates certain functions — for example, to validate credit card numbers for an order processing system. The activity gets funded because it produces a tangible asset for the company, and it is easy to see the potential value of that asset.

Testing, on the other hand, doesn't really produce anything. Sure, you can generate reports and use metrics like fix-find rate, and so forth. But there is no development and therefore no persistent tangible asset. Testing "only" provides information *about* the artifact development produces. It gets funded because we learn something valuable about the system under test. Testing is enlightenment. And the more effectively we can communicate our enhanced knowledge of the system, the more obvious the value of testing becomes.

Any investment decision involves a set of key stakeholders: development managers, directors, the CIO, and others. To convince this group, you must be able to communicate the benefits of an investment in testing to an audience that is entirely unlike your coworkers on the test team. The question is, How?

The Wrong Approach

Let's begin with what you *shouldn't* do. I used to be a tester, so unfortunately, I learned this the hard way. Whenever I was trying to convince a manager or executive that we needed to do more testing, I would take one of two approaches. The first was to take the moral high road: "We need more testing because it's the right thing to do…we haven't thoroughly exercised the functionality as called out in the test plan, etc." This tactic, a favorite of test teams worldwide, isn't very effective. You never have enough time to test all you want, and the manager whose budget is supporting the test team is likely to be more motivated by ship dates and dollar amounts.

The second approach was to quote testing metrics: "Well, we've got three test engineers working sixty hour a week. At last count we had 1,352 open defects. Blah, blah, blah…" Just watch. The manager's eyes will glaze over within three minutes — because you're discussing low-level, technical details that he or she won't care about.

Neither of these approaches ever did me any good. Inevitably, the executive would tell me, "Ship it. No more testing. We can't afford it." It was discouraging. Although *I* knew we couldn't afford *not* to do more testing, I failed to convince the decision-maker because I was talking in tester terms, not his terms.

Talking ROI

I have found that talking in terms of Return on Investment — ROI — is perhaps the single most effective way to garner the interest of decision makers. If you have compelling numbers backed by solid facts, and if you can show that every dollar spent on test automation will return two dollars down the road, for example, then you're practically guaranteed funding.

As a quick review of ROI, here is the common definition:

$$\text{ROI} = \frac{\text{Return}}{\text{Investment}} = \frac{\text{What you get out}}{\text{What you put in}} = \frac{\text{Benefits}}{\text{Cost}}$$

As a function of time, the ROI for automation is sometimes defined as:

$$\text{ROI}(t) = \frac{\Delta(\text{Benefits from automation over manual})}{\Delta(\text{Cost of automation over manual})}$$

The definition I prefer, though, is:

$$\text{ROI} = \frac{\text{Net Quantifiable Benefit}}{\text{Net Cost}}$$

Expressed in words, ROI is the net quantifiable benefit — or the "good" — you get from the use of resources, expressed as a percentage of the cost of those resources. Net Quantifiable Benefit (NQB) is the difference between the achieved benefit through implementation and the opportunity costs of *not* implementing the benefit. Net Cost (NC) is simply your total cost. Net cost is fairly easy to quantify in terms of hours, person-power, or other investments. Quantifying the benefits is not as straightforward, but it can be done. In the examples below, I'll explore both numeric and "intrinsic" value calculations, providing guidelines on how to put some numbers around those intrinsic benefits.

Calculating Costs and ROI

We can define the cost of quality as the sum of various subcosts, including:

- Appraisal costs
- Prevention/adherence costs
- Cost of failure

Appraisal costs include test reviews, inspection, and other recurring assessments such as install smoke tests. Prevention/adherence costs are things like training, ensuring compliance to standards, and so on. For example, the new project you're working on might be aimed at government agencies. That requires that you test for Section 508 accessibility compliance, so the test team would need to come up to speed on Section 508, accessibility testing, and perhaps additional tools required for standards-based testing. Basically, these costs include anything that is not normally part of your test cycle but is necessary for a given project. Failure costs have both internal and external components. Internally, there are costs associated with additional regression testing and development iterations to fix regressions, plus costs for extra tester time.

But the external costs are where the real trouble starts. Service releases or patches consume a lot of resources — especially if you have a multi-component product that requires heavy integration testing. In addition, there is a significant impact on revenue and customer satisfaction if you miss performance tolerances. All of these components need to be considered in your cost calculations. Also note that these costs are above and beyond existing fixed costs, such as salary, software licenses, and capital equipment. Your calculations should include these fixed costs as well as variable costs — investments and the cost of outsourcing, for example.

With automation there are additional costs to consider: fixed costs such as hardware (additional or upgrades to existing), software licenses, tool training, and so forth, as well as variable costs such as test case automation design and execution, script maintenance, after-hours testing by systems (not people), and so on.

Keep in mind, however, that test automation adds an interesting twist to ROI calculations. Investment in automation tools (plus training, maintenance, etc.) is a cost; yet, at the same time, automation is a mechanism through which you can trim costs. How? Consider the base benefits of what automation lets you do:
Broader testing. By automating redundant or manually-intensive tasks, the test engineer can apply himself to more challenging activities that require him to *think* and attack the application in a strategically considered way. For example, he can run passed test cases with some run-time fault injection conditions such as low memory availability.
Resource reallocation. Not only do team members have more time from the alleviation of manually-intensive work, but they can also run automation scripts in batch or unattended mode and shift their attention to other projects.

You will need to consider these benefits when you estimate Net Quantifiable Benefit, which we'll discuss below. For now, however, the task is to document and sum your costs, which gives you a baseline to measure against. Then, you can do a regular assessment about once every two weeks to account for adjustments to the test plan, priorities, and other factors.

The most common way of expressing investment costs and returns is in monetary terms.
However, using metrics for factors such as time or effort — especially when framed as opportunity against other activities — is also completely valid. Use dollars, weeks, labor, or any other units you deem appropriate.

Estimating Net Quantifiable Benefit

Now that we have a clearer picture of costs, let's take a closer look at Net Quantifiable Benefit (NQB) before examining some example ROI calculations. I look at NQB as an opportunity to add significant benefits to an organization by changing the way things are currently done. Some examples of this — taken from my involvement in many projects over the years — include:

- *Automating pieces of the testing process.* This can include automating not just regression tests or GUI tests, but also activities directly related to test, such as requirements and change management. For example, replacing a defect tracking system based on email threads and Word documents, with a tool designed specifically for that purpose. Also there are the base benefits of broader testing capability and resource reallocation that we noted above.

- *Increasing the scope of testing.* For example, if an organization is not performing unit tests, you can identify the value that unit testing would bring to the organization, and include that in the NQB. Other examples include introducing component testing, or performance testing, where it was not being done before.

- *Increasing the depth of testing.* Look for areas in which more rigorous testing would add substantial value. For example, if more thorough acceptance tests would find a significant number of defects earlier in the development lifecycle, then use that to build NQB.

All of these are ways to identify potential quantifiable benefits, which are needed to produce an accurate ROI calculation. Note that each of these approaches introduces a little bit of cost up front, but offers a potentially tremendous cost-savings later on. For example, in many projects that I have been involved with, the system testing phase requires a great deal of integration testing. By introducing unit testing earlier on in these projects, issues that would have taken testers a long time to track down during system tests were easily located and fixed because unit tests were able to localize the defect — in the OS, the kernel or a specific component of the application, for example. That saves a lot of effort and time, which translates to a lot of money saved.

When calculating ROI, I tend to keep the Net Cost figures as tangible and as accurate as possible, and keep the more intangible estimates in the NQB. To continue the unit testing example, the license and manpower costs of performing unit tests would be added to Net Cost, while the cost reduction in system testing — or potential savings — resulting from the unit testing would be added to Net Quantifiable Benefit. There are really only two variables in the ROI equation: Net Cost and NQB.I prefer to keep one variable, Net Cost, as free from speculation as possible, rather than opening up both variables to uncertainty.

The distinction between NQB and Net Cost provides testers with an opportunity to help upper management more clearly understand challenges faced by test managers. As an example, I was recently involved in a project in which we determined that it would be valuable to perform pre-integration testing to ensure the reliability and functionality of the system’s components. The responses from the development teams were interesting. One team was enthusiastic; they immediately did some basic functionality and boundary condition testing on the code and reported the results to the other development and test teams. The second team was so frightened at the thought of testers checking their code before the integration phase that they also stepped up the testing efforts themselves. They, too, did some basic unit testing and reported results into the defect tracker — something they hadn’t been doing before. The third team didn’t want to have anything to do with unit testing and said so pretty clearly. We accepted that stance but then adjusted our testing costs for that component and our integration testing. When the development manager saw that our estimates for testing that team’s component was five times higher than costs for testing other components, he asked why. And the next week, that third team started doing more formal pre-integration testing, too. This was a great event: The test team was leading change in another part of the organization for the overall good of the company.

ROI for Manual Testing

Let’s look at the sample ROI calculation for manual testing shown in Figure 1.

Sample ROI Calculation 1: Manual Testing

Assumptions: 6 people @ $65,000/year or $1,250/wk.; benefits 20% of salary
NQB = opportunity cost of lost sales in 1 year
= ($3,000/license - 50 customers/mo. - 12 mo.)
= $1,800,000 in incremental revenue gained
NC = salaries + benefits + hardware + software
= [(6 - $65,000) + (0.2(6 - $65,000)) + $250,000 + $175,000
= $390,000 + $78,000 + $250,000 + $175,000
= $893,000 investment
ROI = savings from testing / costs of testing - 100%
= NQB/NC - 100%
= $1,800,000/$893,000 - 100%
= 202% (or by a factor of 2.02)
Each $1 of the $893,000 investment yielded a return of $2.02

Figure 1: Sample ROI Calculation for Manual Testing

In this example, the person performing the ROI analysis has determined that the Net Quantifiable Benefit is equal to the opportunity cost of *not* testing certain functionality and the resulting potential negative impact on revenue. He estimates that the company would lose fifty customer orders per month, each one averaging $3,000. To arrive at these estimates, he obtained information from the operations center — such as the value of the average order, and monthly volumes — and worked with the customer service team to understand the probable impact of delivering a system that lacks specific features. The lost revenue amounted to about $1.8 million — and that’s the NQB.

These costs were based on the test cases and strategies that he determined would be required to test appropriately. He calculated that the project would require six test engineers earning $65,000 a year each, plus an additional 20 percent in benefits. He then added in the cost of hardware and software required for the manual tests, which translates to a little less than $.9 million for the year. That brings the ROI in this example to about 200 percent, or approximately two dollars for every dollar spent.

ROI for Automated Testing

Now let’s look at the example ROI calculation for automated testing in Figure 2.

Sample ROI Calculation 2: Automated Testing

Assumption: $65,000/year or $1,250/wk.; with benefits 20% of salary @ $1,500/wk.
NQB = manual testing labor costs – automated testing labor costs
= (3 people - 7 wk. - 4x/yr.) – (2 people - 2.5 wk. - 4x/yr.)
= (3 - $1,500 - 7 - 4) – (2 - $1,500 - 2.5 - 4)
= $126,000 – $30,000
= $96,000 in savings from test automation (point release)
NC = licenses + maintenance + training = $30,000 investment
ROI = savings from automation / costs of automation - 100
= NQB/NC - 100
= $96,000/$30,000 - 100
= 320% (or by a factor of 3.2)

Each $1 of the $30,000 investment yielded a return of $3.20

Figure 2: Example ROI Calculation for Automated Testing

In this example, the test manager calculates the amount of time (converted to dollars, based on the same compensation assumptions for a test engineer) spent for the same amount of testing as in the manual example. He estimates that, for the four test cycles in the project, three people could do the job manually in seven-week cycles. If the company invests in automation, they can use one less person and get the job done in two-and-a-half weeks, as opposed to seven weeks on average. This project happened to have a lot of repetitive graphical user interface (GUI) testing, which is much easier to automate. The savings is $96,000. Taking into account the $30,000 investment in licenses, training, and maintenance, that yields a return of three dollars and twenty cents for every dollar spent.

A few other aspects of this example are worth noting. First, although you need fewer testers to do automated testing, you do not necessarily need to reduce head-count; you can deploy the extra people to do more thorough testing on other projects. Second, remember that the two people who do the testing on the proposed project will actually spend less time on it than the larger, manual testing team would have done; so they can spend more of their time on the toughest, most challenging aspects of testing, instead of on tedious, repetitive tasks.

Long-Term Payback

As mentioned earlier, test automation is a long-term investment. You cannot expect immediate returns. Consider the final example shown in Figure 3, which calculates ROI twice, at twelve-month intervals.

Sample ROI Calculation 3: GUI Testing – Manual vs. Automated

Assumptions:
-A new product with all new tests
-5 people-years developing manual tests, 15 people-years for automated tests
-1 person maintenance after 1st year for automated tests
-10 people full time running manual tests, 1 person for automated
-Fixed costs for automated tests of $90,000 with useful life of 3 years
-Periods of time (***t***) selected: 12 mos. (250 days) and 24 mos. (500 days)
-People cost $78,000 per year = $312 per day = $39 per hour

NQB (***t***) = S (improvement in fixed costs of automated testing) +
S (variable costs of running manual tests during time ***t***) –
S (variable costs of running automated tests during time ***t***)

NQB (12 months) = 0 + (10 people * $78,000) – (1 person * $78,000)
= $702,000
NQB (24 months) = 0 + (10 people*$156,000) – (1 person *$156,000)
= $1,404,000
NQB (12 mo) = $702,000 NQB(24 mo) = $1,404,000

NC(***t***) = S (increased fixed costs of automated testing times) +
S (variable costs of creating automated tests) –
S (variable costs of creating manual tests) +
S (variable costs of maintaining automated tests during time ***t***)

NC (12 mo) = ($90,000 * (1/3)) + (15 * $78,000) – (5 * $78,000) + $0
= $30,000 + $1,170,000 – $390,000 = $810,000

NC (24 mo) = ($90,000 * (2/3)) + (15 * $78,000) – (5 * $78,000) + $78,000
= $60,000 + $1,170,000 – $390,000 + 100,000 = $940,000

ROI (12 mo) = NQB/NC = $702,000 / $810,000 = 0.867 [small loss]

ROI (24 mo) = NQB/NC = $1,404,0000 / $940,000 = 1.494 [49% return]

Figure 3: ROI Calculations at Twelve-Month Intervals

By now, the benefit and cost calculations are fairly transparent. But notice that the ROI at twelve months actually shows a small loss, whereas the ROI at twenty-four months indicates a 49 percent return. In the second year, the savings in tester salaries more than makes up for the additional costs of automated testing. Of course, your calculations will vary. You may already know of additional costs or benefits associated with your project that these examples do not take into account. And you may have alternative sources of information to back up your estimates. Remember, the more reliable your information, the more accurate your ROI calculation, and the more compelling your argument for test automation will be.

What Should You Automate?

In too many organizations, most testing is done in the late phases of a project, after much of the development work has been completed. As testers, our objective is to move testing earlier in the software development lifecycle, to start performing tests and finding defects in the initial phases of development or during design, when they are easier to fix. By doing this, we can lower system testing costs later in the lifecycle. In fact, this is one of the principal benefits of the iterative development approach.

When you take this approach, it is clear that testing cannot be an afterthought, an activity to be performed after everything else is done. Rather, testing is integrated throughout the development process, from the earliest stages on. And so it follows that testing is not the only area in which automation pays dividends — many aspects of software development that relate to testing are also prime candidates for automation. I always advise customers to automate not only GUI and regression testing and component/unit testing, but also change management, configuration management, and requirements management. These areas have a direct impact on the testing process and also yield the highest ROI from automation investments — from a *business* perspective.

As a former Rational employee, I can tell you from experience that Rational's integrated toolset consistently provides exceptional value and high ROIs in these areas. The tools include:

- Rational® Robot for GUI and Regression testing.
- Rational® QualityArchitect and Rational® PurifyPlus for component and unit testing.
- Rational® ClearQuest® and Rational® ClearCase® for change management and configuration management.
- Rational® RequisitePro® for requirements management.

In addition, Rational TestManager® provides an easy way for organizations to move from manual to automated testing, because it manages both types of tests. Plus, testing teams can use Rational TestManager to develop and maintain test plans, using inputs from other Rational tools: requirements in Rational RequisitePro and use cases in Rational Rose®, for example.

Rational Suite® TestStudio®, which includes all of the tools above and more, enhances the overall value of an organization's investment because the tools all work together and are integrated with an iterative development methodology — the Rational Unified Process®. Using a common process and toolset in combination with reusing artifacts throughout the development lifecycle, and maintaining a commitment to early testing can dramatically improve your ROI for test automation.

Talk Business to Business People

ROI is a powerful tool for conveying the value of test automation, but it should by no means be the only one in your bag. As you present your ROI calculations to management, you can strengthen your argument by talking about the less obviously quantifiable business advantages of testing as well. This might include the following:

- Clearly align the testing project with your corporate or divisional goals.
  How does testing impact the big picture? Explain this in as much detail as you can. For example, if ensuring positive customer experience on your Web site is a stated corporate goal, usability and accessibility tests take on added importance and are aligned with those goals.

- Identify testing with the customer and show how it will avoid customer problems.
  One way to approach this is by asking: "What would happen if our system maxed out at ten concurrent users? How many orders would we miss? How much money would we lose with each missed order?"

- Explain how testing will integrate with the current development process.
  This is important. What you do as a tester will affect other groups. Clearly state how your proposal integrates with existing constraints and systems. If instituting your plan would require changes in other systems or processes, don't skim over those realities. They are part of the cost of your solution.

- Generate interest and get buy-in.

To succeed with your plan, you need interest. Not the kind of interest you pay on a loan — you need involvement. To get it, you must make sure others understand the value of the information you will gather through testing by showing them examples and explaining possible outcomes.

Above all, you'll want to emphasize that automated testing provides much more than defect metrics. It brings a unique value to the project: enlightenment. Used in combination with an iterative development approach, it enables all team members to focus on quality assurance throughout the project rather than just at the end. It provides valuable learning they can apply to future projects. And finally, it gives them confidence in their work and helps them produce a better product. When it comes to business investments, it doesn't get much better than that.

References

- Tilo Linz and Matthias Daigl, GUI Testing Made Painless. Implementation and results of the ESSI Project Number 24306
- *The Business Case for Outsourced Testing*. White paper from *The Testers' Network.* www.veritest.com/testersnetwork
- VeriTest's Testers' Network: http://www.veritest.com/Testers'Network/
- Douglas Hoffman, *Cost Benefits Analysis of Test Automation.* http://www.stickyminds.com

About the Author

With more than fifteen years of software experience in his background, Ed Adams has been a tester, product manager, content developer, and technical marketing manager. A frequent speaker on quality and test automation topics, he has published articles in dozens of periodicals around the world and conducted training sessions for thousands of people; he also serves on advisory boards for several universities and corporations. His formal training is in mechanical engineering and non-lethal weapons systems. More recently, Ed has served in leadership roles at both Rational Software and VeriTest, the testing division of Lionbridge Technologies, Inc.