What's my Testing ROI?

An *m*Verify White Paper

mVerify Corporation

A Million Users in a Box ®

What's My Testing ROI?

Document Number 004-001.

*m*Verify Corporation
350 N. Orleans St. Suite 950
Chicago, Illinois, 60654

www.mverify.com

Abstract

An automated software testing system is a solution to a common business problem: how to produce high quality software with ever-more complex technology under increasing competitive pressure. Automated software testing provides a cost-effective solution to this problem.

This white paper identifies the costs and benefits of software testing and explains the added value of automation. A case study shows how to estimate the return on investment (ROI) of automated testing for your company or department using the *m*Verify ROI model. The calculations are done in a companion Excel spreadsheet, allowing readers to easily estimate the ROI for their unique situation.

Contents

Executive Summary

An automated software testing system is a solution to a common business problem: how to produce high quality software with ever-more complex technology under increasing competitive pressure. Automated software testing provides a cost-effective solution to this problem.

This white paper explains how to estimate your return on investment (ROI) in an automated testing system. Automated software testing improves business results in three ways: increased test efficiency, increased test effectiveness, and shorter time to market.

- Testing efficiency is the average number of tests you can run for an hour of tester time. Higher testing efficiency drives down product development time and costs, improving your bottom line.

- Testing effectiveness is the rate at which your testing technology reveals bugs *before* your systems are released. Increased effectiveness reduces costs through a better product, improving your bottom line now and your top line later by building customer satisfaction and loyalty.

- If your company sells software products that must be tested before release, reducing your time to market can yield enormous benefits. Replacing manual testing with automated testing can cut weeks or months from elapsed testing time, improving your top line.

This white paper presents the *m*Verify ROI model, which also is available in an Excel spreadsheet. With this model, you can enter your own parameters to estimate your ROI. All of the assertions made in this document and embedded in the model are backed up by many studies, which are listed under *Sources*.

We'll first look at the overall costs and benefits of software testing. Then, we'll show you how to estimate the ROI of automated testing for your company or department.

The Business Case for Automated Testing

Around $15 billion a year is spent on automated testing for software and network systems. To understand why this is, we need to consider what software testing does and what automated testing can do for your business.

How Does Software Testing Create Value?

Software testing creates value by revealing bugs in a software system before it is released to production or shipped to customers. If bugs aren't found this way, they are said to "escape." Sooner or later (usually sooner) a customer or end user will do something that triggers an escaped bug. On average, it costs 10 to 1,000 times more to fix an escaped bug than fixing before release. Besides the immediate impact of the failure, escaped bugs usually have other negative consequences with significant costs.

- Increased customer support expense: headcount, payroll, overhead, communications, equipment, etc.
- Unexpected expense to prepare and distribute a patch, service pack, or bug-fix release.
- Loss of revenue and increased costs due to disruption of or outages in revenue-generating production systems.
- Loss of customers to competitors due to outages and/or bad customer experience, leading to lost revenue.
- Loss of customers to competitors due to negative publicity and/or word-of-mouth about failures, leading to lost revenue.
- Disruption or outage of a process or equipment that depends on a failed system, leading to spoilage losses, opportunity costs, or higher indirect and overhead costs.
- Lawsuits for injury or damage resulting from a system failure.
- Increased interest or insurance costs resulting from unfavorable risk assessment due to low operational reliability.
- Financial losses due to missing, incorrect, or late information.
- Criminal prosecution (Sarbanes-Oxley) due to missing, incorrect, or late information.

The spreadsheet provided with this document will help you estimate these costs. These costs can quickly add up, which is why many people look at software testing as a kind of insurance. Automated testing goes beyond basic preventative measures to achieve significant competitive advantage and enhanced financial results.

How Does Automated Software Testing Create Additional Value?

Software can be tested manually. After designing or improvising a test case, a tester types at a key pad or touch screen, and then observes the response. If the system hangs, takes a long time to respond, or its output looks incorrect, this test may have revealed a bug. In the world of testing this is a good thing! The found bug can now be fixed and will not escape to customers or end users.

But that's only if the tester is clever or lucky. Many studies have shown that for every thousand new lines of a software program, we can expect an average of five bugs. At least 5,000 bugs are probably lurking in an untested business system with just a million lines of software. With so many bugs, you'd think it would be easy to find them. But it isn't. Even in simple systems, the number of possible combinations of input values and sequences is astronomical. Only a few of these will trigger an observable failure. Software testers have to guess which inputs will do that.

Increased Productivity and Effectiveness

It would help if your testers could run many more tests, *much* more quickly, working 24 hours a day, seven days a week. Of course, that isn't possible. Human testers average about one completed test per hour, taking account of all of the time they need to design, set up, input, run, observe, evaluate, and document one test case. Although manual testing has a role to play, it just isn't fast enough for today's large, complex, multi-platform systems.

This is where automated testing comes in. Automated test systems use a "test script" to submit input and observe results. This is a special kind of computer program that runs another program (the program being tested) and checks the results. *m*Verify's MTS uses object-oriented technology, so we call our test programs "test objects." Once a test object is produced, it can be executed time and again with minimal human effort.

Besides just running tests faster, automatic repeatability has other benefits. For example, suppose a tester finds a bug, which a programmer then corrects. On average, 7% of bug fixes create new bugs, so it's a good idea to run the same test again and repeat previous tests on *other* parts of the system. But this almost never happens when testing is done manually. The testers are too busy testing new features. Even if they had the time and could tolerate the boredom, it is very hard for a person to exactly repeat a test run.

So, automated testing amplifies the basic value of testing in several ways.

- *More testing gets done faster*. This increases the odds of revealing bugs, thereby avoiding the costs and consequences of their escape.

- *Tests can be repeated exactly*. Automatic retesting allows existing test scenarios to be run many times, with identical input and timing on each run. The same test cases are used as systems change, providing consistent results. Any tester can run a test object developed by another tester and get consistent results.

- *Testers focus on quality, not key-pressing*. Your testers can spend their time thinking of more clever ways to find bugs, instead of typing test inputs and checking output.

- *Tests can be reused and rerun at will.* Test objects can be saved and rerun any time. The incremental cost to rerun an automated test is insignificant, so the unchanged parts of a system can be quickly and consistently rechecked. This greatly increases your chance of revealing new bugs that are side-effects of new or changed software. You can have greater confidence that something hasn't "slipped through the cracks" as your systems change.

- *Realistic stress and stability testing becomes feasible*. Many bugs are triggered only when a system is tested with a high input rate or run continuously for many hours. It's usually not possible to achieve these conditions with manual testing.

Reduced Time to Market

In today's highly competitive information technology markets, few firms would dare to release an untested product. Although testing is necessary, the time it adds to the product development cycle is both a problem and an opportunity. A McKinsey study found that manufacturing companies lose an average of 33% of after-tax profits when they ship products six months late, compared to 3.5% when they overspend 50% on product development. Reducing testing time even a few weeks can have significant top-line and bottom-line improvements.

- Higher margins are possible in the early life of a product if your competition cannot challenge your price with a comparable product.

- Earlier entry means you can capture a greater share of the market. Breakeven is reached sooner, so subsequent sales achieve an even greater margin.

- Higher market share usually translates to lower unit cost, compared with a competitor's lower volume, higher unit-cost product. Higher volume usually means greater quantity discounts from suppliers and better operating efficiency. This is especially important if you make a hardware product or product that requires factory integration.

- With more time to sell your product, increased total revenue will achieve a higher rate of return on the initial product investment and other investment ratios.

- You accelerate the investment recovery for the product. Revenue is realized sooner, reaching breakeven and your product's target ROI sooner.

Once a test scenario has been automated, it can be repeated, on average, in 1/20th the time it takes to re-enter it by hand. If the same tests must be repeated on many different devices or computers, an automated system can be set up to do all that work concurrently. This two-way speedup is the main reason that automated testing reduces time to market. The ROI spreadsheet provided with this document will help you estimate time-to-market benefits for your company or product line.

Is There Any Downside?

Automated testing has both startup and maintenance costs. Migrating from a mostly manual test process to a mostly automated test process is comparable with automating a mostly manual business process. After you've established an automated testing process, there are costs to add and revise test objects. As application user interfaces and features change, you can expect a proportional cost to change corresponding test objects. For highly volatile applications, it may make sense to defer automated testing until your systems stabilize. The ROI model uses the Test Scenario Volatility and Scope Growth parameters to estimate these ongoing costs.

Some kinds of application systems are not well-suited to all-out automated testing. For example, highly interactive video games require subjective evaluation that isn't practical to automate.

Automated testing isn't a silver bullet and will not achieve high quality software by itself. Proven software engineering techniques can greatly improve software before you start any kind of testing and can be more cost-effective at finding or preventing certain kinds of bugs. These techniques amplify testing effectiveness. Testing can then focus on insidious bugs that can't be prevented or found in other ways. As with so many things, the weakest link limits overall results.

How to Estimate Your Test Automation ROI

The mVerify ROI Model

The *m*Verify ROI model estimates incremental cash flows, ROI, net present value, and months to breakeven. The estimates reflect typical test automation costs and benefits

- Benefits of increased testing efficiency.
- Benefits of increased test effectiveness.
- Benefits of reduced time-to-market.
- Cost to acquire, convert, and use an automated testing system.

You enter parameters to characterize your testing requirements, development cycle, and tooling costs. Parameter values for average time to develop, run, and maintain test objects are built-in to the model. You can also enter your own values. The model uses all these parameters to estimate costs and benefits over two years. Individual effects are tabulated by quarter. You can change any parameter in the spreadsheet and instantly see the impact it has on estimated ROI.

We'll use the *CMD* case study to show you how to develop your test automation ROI estimate.

Case Study: Construction Mobile Data, Inc.

Background

Construction Mobile Data, Inc. (CMD) develops and markets the Job Site Management System, also known as *JobMan.*[†] With JobMan, contractors, supervisors, engineers, suppliers, and architects access work orders from a central server using a handheld or laptop computer. A wireless data link allows JobMan to be used at any place in a construction site, anywhere in the world. Field information can be shared and updated in real time without being tethered to a desk.

JobMan is offered as a turnkey service and as an application software product. The JobMan client software runs on PDAs using Microsoft's Windows Mobile platform, including the Compaq iPaq and Asus. The Symbol Technologies ruggedized handheld with extended life batteries is offered for field users. JobMan automatically selects the most economic available wireless datalink to the central server. The connection may be via high-speed cellular data, cellular data, WiFi, wired Ethernet, or a satellite datalink.

[†] Except for trademarked product names used for illustrative purposes, all other names and circumstances in the CMD case study are fictitious.

JobMan 1.0 and 2.0 were successful, generating about $15 million in total revenue. Then, competitive products came "out of nowhere." In response, CMD developed JobMan 3.0. With 3.0, users could attach pictures taken with a Smartphone camera to work orders, use new multi-mode handsets, and query the server database with a web browser.

The CMD quality assurance team designed and documented about 500 test scenarios for the 1.0 and 2.0 releases. Each test scenario was based on a typical interaction (enter a new work order, look up subcontractor details, etc.) To run a test, a tester entered scenario inputs and then noted results in a log file. Test inputs, results, and log entries often varied depending on who designed the test scenario and who ran the test.

Although the same JobMan client software is installed on all mobile devices, CMD's quality assurance department found that bugs "followed" certain devices. For example, the JobMan software would work fine on the iPaq, but intermittent failures occurred on the Symbol handset. About 50 scenarios were developed to isolate handset-specific bugs. After release 1.0, the testers began repeating each test scenario on each supported mobile device.

JobMan 3.0 supported new handset and airlink configurations, as well as new features. The quality assurance team had to devise tests for new capabilities, cover new combinations, and decide how much of the existing system to retest. The resulting plan called for 18 weeks of testing, up from 6 weeks for JobMan 2.0. Although CDM's management knew scope had increased, they were deeply concerned that competitors were gaining at CDM's expense. Every day the 3.0 release was delayed surely meant lost sales. Testing time was therefore set at 12 weeks and new testers were hired. To meet this deadline, the quality assurance team decided to not repeat existing tests for unchanged features and to skip testing some low-risk configurations.

The testers found new bugs at roughly the same rate as they'd seen when testing JobMan 2.0. They concluded that rerunning existing tests could be skipped without a big risk. Instead, they concentrated on testing new features and handsets. In all, about 100 new test scenarios were identified, about a third of the existing test scenarios changed, and 50 obsolete scenarios were dropped. Despite many long days and the increased testing schedule, the release deadline slipped. JobMan 3.0 finally shipped after 14 weeks of testing.

Within the first week of deployment, field incident reports jumped. A second wave followed after JobMan was in operation for over 127 hours – a situation that wasn't tested. When the largest turnkey customer had more than 39 active users, the CMD server crashed and wiped out all user inputs from the preceding twelve hours. Angry letters and cancellations followed.

Under pressure from key customers and investors, CMD's management identified several root causes. Newly hired test engineers hadn't performed documented manual test procedures the same way experienced testers did. Many critical test scenarios were not documented at all, and terse directives from the senior testers were often misunderstood. Time was too short and testing resources too thin to repeat all the manual tests. As programmers rapidly added new features, few if any tests were run to find interaction problems with unchanged features.

After this assessment, CMD decided to evaluate an automated testing solution, using the *m*Verify ROI model. The following shows how CMD's assumptions were entered as parameters. The CMD case study data is provided in the read-only Excel file AutoTest-ROI-Calculator-CMD.xls. Open this file with Excel, then click on the **Parameters** tab to see the same inputs.

Test Investment Parameters

The ROI model needs basic data points to estimate effects and costs. CMD decided to use its actual experience with JobMan 3.0 for these assumptions.

- CMD decided to equip the both the programming and quality assurance staff (25 persons in total) with an automated testing system.
- Starting with 500 test scenarios, 100 new test scenarios were added, and 50 were dropped, for 550 total test scenarios.
- Test scenarios had to be repeated on four differently configured platforms to test: the iPaq, Symbol, Asus, and the newer Smartphone.
- Manual testing of 3.0 took 14 weeks, which is 98 elapsed days.
- The same scope growth (10% = 550-500 / 500) seen for the 2.0 to 3.0 release was assumed.
- The one-third (33%) test scenario volatility seen for the 2.0 to 3.0 release was assumed.
- CDM's accounting department recommended using a 15% rate for cost of capital.

Number of Testers	25	persons	Number of developers and testers to support
Hourly cost, each tester	$ 50.00	dollars	Average fully burdened hourly cost per tester
Existing Manual Test Procedures	550	procedures	Number of existing manual test procedures
Unique configurations to be tested	4	platforms	Number of different platforms on which tests are repeated
Number of releases per year	2.0	releases	Number of releases per year
Existing manual test duration	98	days	Average manual testing duration
Scope Growth	10%	rate	Average net add/drop application scope
Test Scenario Volatility	33%	rate	Average percent of test scenarios changed per release
Cost of Capital	15%	percent	Average cost of capital for discounted cash flow

Test Productivity Parameters

The productivity parameters define the average work hours to create, run, and maintain automated tests. The model's default numbers are as reported in the Linz and Daigl study.[†] Because CMD hadn't actually measured their own productivity, they decided to use the default values.

Test Scenario Design work	11.6	hours	Average time to design a new test scenario
Test Procedure Conversion work	4.0	hours	Average time to convert a procedure to a test object
Test Object Coding work	7.9	hours	Average work to code a test scenario in a test object
Test Object Run work	0.2	hours	Average time to run a test object
Test Procedure Run work	3.9	hours	Average time to manually run a test scenario
Test Procedure Maintenance work	2.0	hours	Average work to change a manual test procedure
Test Object Maintenance work	4.0	hours	Average work to change a test object

Test Effectiveness Parameters

The effectiveness parameters define assumptions about how many bugs will be found and the average cost of escaped bugs.

- When manual tests are automated, *m*Verify typically sees bugs reduced by at least a factor of two, and as much a factor of ten. CMD decided to use the low end of this scale, assuming they'd find twice as many bugs with automated testing.
- CMD's analysis of past bug reports found a bad fix rate of 11% (i.e., about one in ten bug fixes had to be fixed again.) This value was used instead of the industry average of 7%.
- The server crash took nearly two days to resolve. Annualized, this equated to 90 hours of unscheduled downtime. CMD's application service operation generated an average of $15,000 per hour.
- After reviewing all the costs caused by field bugs, CMD determined the average cost per bug was $7,500.

Test effectiveness improvement	2	factor	Expected improvement in bug removal rate
Baseline bad fix ratio	11%	percent	Observed percent of fixes that cause new bugs
Baseline annual outage hours	90	hours	Observed outage hours, annualized
Average outage cost, $ per hour	$ 15,000	dollars/hour	Average cost/loss of one hour of unscheduled downtime
Baseline annual incident reports	120	incidents	Observed production/field incidents, annualized
Average Incident cost, $ each	$ 7,500	dollars each	Average cost/loss of one production or field bug report

[†] This study was funded by the European Software and Systems Initiative (ESSI) of the European Commission.

Time-To-Market Parameters

The Time-to-Market parameters make sense for companies that offer software or software-enabled products, like CMD. If your software system is used internally, this benefit probably doesn't make sense for your ROI. For CMD, the highest cost of the JobMan 3.0 debacle was the loss of customers and revenue to competitors. They estimated the relevant parameters as follows.

Total market for product	$ 5,000,000	dollars	Addressable total market for your product, per release
Maximum market share	45.0%	percent	Market share, earliest possible release
Factory overhead rate	$ 5,325	dollars	Average daily overhead cost

Tooling Costs

CMD decided to use *m*Verify's MTS system because of its unique support for mobile application testing. CMD chose MTS Team Edition to maximize sharing of test objects and development productivity. To achieve realistic stress and load testing, they ordered 500 "virtual users." This would allow CMD to run stress tests at twice the peak rate seen in JobMan 3.0

The startup cost for this configuration was $325,000. Annual support of $65,000 was budgeted as a recuring annual expense. No equipment cost to host the automated testing system was assumed, because the *m*Verify system runs on available desktop computers and CMD has enough handset devices to support the anticipated testing.

	Unit Cost	Unit	Quantity	Cost	Frequency
Base License	$ 25,000	Site	1	$ 25,000	Once
Additional Seats	$ 2,500	Person	20	$ 50,000	Once
Additional Virtual Users	$ 500	Virtual User	500	$ 250,000	Once
Other	$ -		0	$ -	Once
Total Test Automation Software				$ 325,000	

Training	$ 2,500	Session	1	$ 2,500	Once
Equipment	$ -		0	$ -	Once
Other	$ -		0	$ -	Once

Estimated Total ROI

ROI is *(Benefits – Costs) / Costs*, in percent. The *m*Verify ROI model also computes months to breakeven and net present value.[†] Based on the entered parameters, the model indicates CMD should go ahead with the test automation investment.

- Breakeven payback is estimated at seven months after the improvement project starts.
- CMDs' estimated test automation ROI is 1,468% over two years.
- The net present value is positive.

In ROI model spreadsheet, open the **Total ROI** tab to see the summary of the ROI analysis. Open the **Quarterly ROI** tab to see quarterly costs and benefits for two years.

Effectiveness Improvement Estimate

The model estimates the effects of improved bug finding, which leads to improved production/field reliability. Using CMD's 2x assumption, significant cost reductions are forecast.

- System downtime would be reduced by 50%.
- The number of reported bugs would be reduced by 50%.
- For each quarter after the next release, hard costs totaling $303,705 in downtime, incident, and bad fix expense would be avoided.

This effect is expected to recur each subsequent quarter. In the ROI model spreadsheet, open the **Effectiveness** tab to see a side-by-side comparison of the estimated costs of manual and automated testing.

[†] Net present value reflects the time value of money on the overall costs and benefits of an investment. The future cash benefits minus the initial and ongoing investment costs are discounted using the firm's cost of capital and then summed. If net present value is positive, an investment should be made (unless an even higher net present value alternative exists). Otherwise it should not.

Productivity Improvement Estimate

With the *m*Verify's MTS, CMD can repeat all test scenarios on each platform with little incremental effort. Due to the decreased time to run test scenarios, CMD expects a significant reduction in the direct labor expense for testing.

- $297,000 reduction in the cost to test JobMan 4.0.
- $408,278 reduction in the cost to test subsequent releases.

In the ROI model spreadsheet, open the **Productivity** tab to see a side-by-side comparison of the estimated costs of manual and automated testing.

Time to Market Reduction Estimate

The ROI model quantified the opportunity cost of delayed entry for CMD's management. Although the losses of JobMan 3.0 could not be undone, the estimated benefits for the 4.0 release were significant.

- A 69% decrease in the testing cycle time, or 31 days sooner.
- The estimated additional sales opportunity is $419,178 in the first quarter of the next release.
- The reduction is expected to produce an additional $295,890 of revenue in each subsequent quarter, compared to later release dates with manual testing.

In the ROI model spreadsheet, open the **Cycle Time** tab to see a side-by-side comparison of the estimated costs of manual and automated testing.

Sensitivity Analysis

After reviewing the ROI estimate, CDM's management was skeptical. Suppose unanticipated problems occur? How robust is this estimate? What is the minimum level of performance that would justify this investment? At what level wouldn't it make sense?

To answer these questions, the quality assurance staff entered several less favorable assumptions into the ROI model to see how the estimate would change.

- *No time to market benefit.* Suppose CDM's system was used internally only, as are most Enterprise IT systems. They entered **0.00** for the Total Market for Product under Test parameter to cancel this effect.

- *No effectiveness improvement.* Suppose automated testing wasn't any better at finding bugs than the manual process. Would there still be any reason to automate? CMD entered **1** for the Test Effectiveness Improvement parameter to indicate that automated testing would not be any better than manual testing.

- *Less test automation speed-up*. Much of the benefit of test automation comes from running more tests faster. What happens to the ROI estimate if the speed-up is lower? What is the breakeven productivity rate between manual and automated testing? After a few experiments, they found that entering **2.8** for the Test Object Run Work parameter drove the net present value under zero.

The following table shows the values from the **Total ROI** worksheet after each change was made.

Reduced Assumption	**Impact on ROI**		
No Cycle Time Benefit	✓	✓	✓
No Effectiveness Benefit		✓	✓
14 times slower test execution (2.8 hours per test, up from 0.2 hours)			✓
Total Costs	$ 392,500	$ 392,500	$ 392,500
Total Benefits	$ 3,344,336	$ 1,656,836	$ 427,036
Net Cash Flow	$ 2,951,836	$ 1,264,336	$ 34,536
Net Present Value	$ 2,404,055	$ 1,024,139	$ (18,873)
Project ROI	752%	322%	9%
Months to Breakeven	7	7	22

- Dropping the time to market benefit reduces ROI by a factor of two. The investment is still economically justified, however, as the net present value is positive with an ROI of 752%. There are still significant benefits from effectiveness and productivity improvements.

- Assuming no better bug-finding and dropping the time to market benefit reduces ROI to 322%. The investment is still economically justified, however, as the net present value is positive. There are still significant benefits from productivity improvements.

- Assuming automated test runs are fourteen times longer than the default average, dropping the time to market benefit, and assuming no better bug-finding ability reduces ROI to 9%. The investment is no longer economically justified, however, as the net present value is negative.

This sensitivity analysis indicates a favorable return on investment is likely even if actual results for the assumed values are significantly worse than assumed.

Conclusion

Building Your ROI Model

To estimate ROI for your situation, you'll first need to estimate or determine realistic parameter values for your situation. Then, input them into the **Parameters** worksheet. This is in the Excel file AutoTest-ROI-Calculator.xls. All estimates in this model are computed from these parameters. To see the effect of different parameter values, just enter a new value. All cost, benefit, and ROI estimates are automatically updated.

- Be sure to save a copy of the original model, so you can restore if necessary.
- Enter your assumption for each parameter in the gold boxes, or just use the default.
- Enter zero for any parameter which is not applicable in your situation.
- Be sure to enter tool costs in the "Tooling Costs" worksheet.

Here's an overview of the worksheets in the model.

Tab Name	Model Content
Parameters	Input the basic model parameters on this sheet. See the CMD case study for examples.
Tooling Costs	Enter the cost for the test automation solution you're considering and other startup costs in this sheet.
Total ROI	The output of the model is summarized on this page. Enter your company name to complete the customization.
Quarterly ROI	The cost and benefit output of the model is detailed by quarter in this sheet. The ROI, net present value, and breakeven month is computed here.
Productivity	This sheet shows a side-by-side comparison of the productivity effects of manual and automated testing. The cost to convert manual to automated tests is computed. The relevant parameters are echoed at the top.
Cycle Time	This sheet shows a side-by-side comparison of the cycle time effects of manual and automated testing. The relevant parameters are echoed at the top.
Effectiveness	This sheet shows a side-by-side comparison of the effectiveness effects of manual and automated testing. The relevant parameters are echoed at the top.

If you have any questions about using the model, email or give us a call.

The Value of Test Automation

A recent NIST study estimates that ineffective testing costs the U. S. economy as much as $56 billion per year. At the start of this white paper, we asked why small and large organizations around the world spend $15 billion a year on testing products and services. The CMD case study suggests an answer: automated testing offers a compelling value proposition.

It's therefore likely that your company can benefit from automated testing. If you'd like to discuss how *m*Verify's advanced test automation technology can get you started, please contact us.

Paul R. Sand
VP Business Development
*m*Verify Corporation
Chicago, IL 60654

www.mverify.com/

Tel: +1 (312)-881-7337
Email: Paul_Sand@mverify.com

Sources

Planning Report 02-3: The Economic Impacts of Inadequate Infrastructure for Software Testing. U.S. Dept of Commerce, National Institute of Standards and Technology (NIST), Technology Program Office, Strategic Planning and Economic Analysis Group, May 2002.

Barry Boehm and Victor R. Basili. "Software Defect Reduction Top 10 List," *IEEE Computer,* vol. 34, no. 1, pp. 135-137, January 2001.

Capers Jones. *Software Quality: Analysis and Guidelines for Success.* International Thompson Computer Press: London, 1997.

Tilo Linz and Matthias Daigl. *GUI Testing Made Painless: Implementation and Results of the ESSI PIE 24306.*
http://www.imbus.de/engl/forschung/pie24306/gui_test_made_painless.shtml

Preston G. Smith. "From Experience: Reaping Benefit from Speed to Market," *Journal of Product Innovation Management,* vol. 16, pp. 222-230, 1999.