

# Automate App Tests To Achieve Maximum Benefit

By Shaun Bradshaw

**A**utomating test execution allows organizations to achieve improved test coverage, reduced test execution time, decreased

production errors and increased repeatability and reliability. After companies invest time, effort and resources to build a test automation environment, it's important for them to capture costs and benefits in a meaningful manner to verify financial return on investment. Demonstrating financial success allows upper management to further justify investing in process improvement resources.

## Calculating Costs vs. Benefits

While the costs associated with test automation are relatively easy to track and to account for, the benefits can be tricky. Despite the claims of some test tool vendors, setup and implementation costs often go beyond tool licensing fees and additional hardware.

These added costs vary depending upon the test automation framework used. There are two sides of the equation for determining the financial success of a test automation implementation effort.

When reviewing any process improvement program, the primary value derived from the effort can be calculated in terms of cost savings. To calculate the amount saved, you need an initial cost basis against which to compare. With that in mind, it's important to establish metrics tracking prior to implementing test automation. Let's look at each benefit related to test automation and discuss what metrics can be tracked to determine the comparison basis, as well as the value/savings derived from the automation effort.

## Benefit 1: Improved Test Coverage

How can you track test coverage, and what metrics should you use to indicate how much of the application has been tested? Start by assessing how many test cases can be executed during a manual test effort. Note that test coverage isn't the same as determining how many defects exist or how many test cases have passed. It's simply a measurement of how much application functionality was tested based on the number of test cases executed regardless of the outcome (pass or fail).

Using the test coverage metric, simple analysis can determine an automation effort value. Use Tables 1 and 2 (see page 30) as an example.

Across the four product releases, the test team in Table 1 averages 15 minutes per test, or about four tests per hour. Table 2 assumes that all regression tests are automated, that all tests are executed and that automated tests can be executed in two minutes each, or about 30 tests per hour. (The two-minute estimate is based on an ongoing automation business case study by David Dang of Questcon Technologies, Greensboro, NC, published 2003.)

Comparing the results of the two tables, a 25 percent increase in test coverage can be achieved with no additional test execution time overhead. Increased test coverage allows an organization to test more in less time, utilize fewer resources to execute the same number of tests in the same amount of time, or both. Increased test coverage also improves the chance of finding defects that might otherwise make it into production.

## Benefit 2: Reduced Test Time, Cost

To calculate the savings associated with using automated test execution, it's necessary to track two more data elements in addition to the information derived from the test coverage analysis above: the loaded costs of a manual tester during test execution, and the loaded costs of an automated tester during test execution.

Shaun Bradshaw is director of quality solutions for Questcon Technologies, a software QA and testing firm.

Photograph by Aaron Springer

Reviewing the test coverage analysis above, it takes less time to execute all of the test cases using automation than it took to manually execute approximately 95 percent of the new tests and 55 percent of the regression tests. In addition to increasing the test coverage capability of a test effort, the cost associated with testing is reduced, and time-to-market decreases.

Applying costs to the execution time helps demonstrate the savings potential over the course of the four releases. Include the following two assumptions: Manual testers cost US\$50 per hour inclusive of overhead costs, and automation testers cost \$75 per hour inclusive of overhead costs.

By applying these resource costs to the data elements used in the test coverage analysis, Table 3 compares a fully manual test effort versus a test effort that includes automation of the regression test suite.

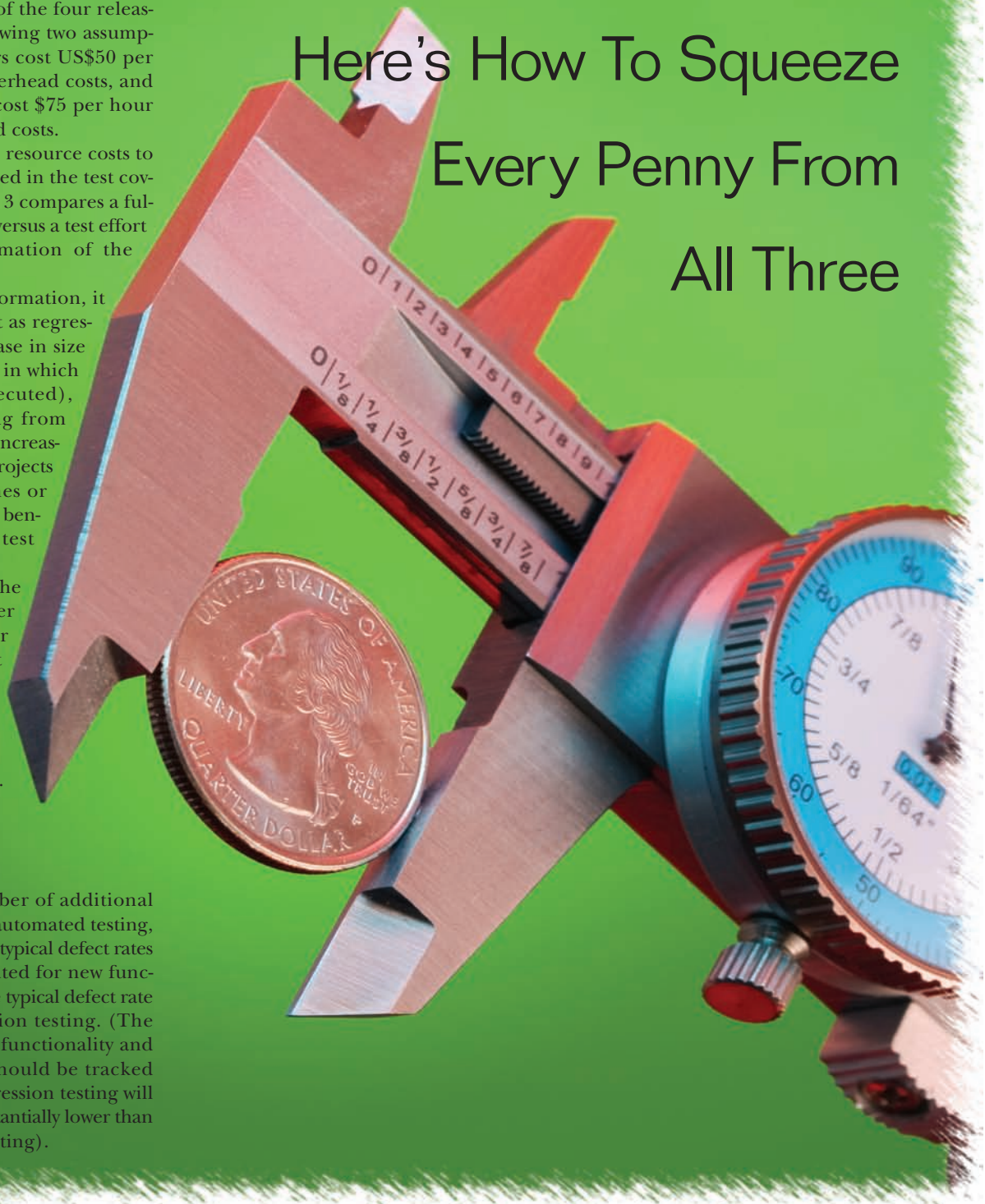
Based on this information, it can be deduced that as regression baselines increase in size (or in the frequency in which they need to be executed), the savings resulting from test automation also increases. This means that projects with frequent patches or new functionality will benefit the most from test automation.

Continuing with the earlier example, over the course of the four releases, utilizing test automation can increase test coverage by 25 percent and decrease test execution costs by \$2,304.

### **Benefit 3: Decreased Production Errors**

To predict the number of additional defects found using automated testing, you must understand typical defect rates when tests are executed for new functionality, as well as the typical defect rate when doing regression testing. (The defect rates for new functionality and regression testing should be tracked separately, since regression testing will almost always be substantially lower than new functionality testing).

# Tests Are Gauged By Code Coverage, Execution Time and Cost— Here's How To Squeeze Every Penny From All Three



**TABLE 1: MANUAL LABOR, BAD**

Project ID	Number of New Tests <sup>1</sup>	Number of Regression Tests	Number of Tests Executed <sup>1</sup>	Test Execution Time <sup>2</sup>	Required Test Execution Time <sup>3</sup>
Release v.1.0	200	0	190	46 hours	49 hours
Release v.1.0.1	20	150	103	25 hours	42 hours
Release v.1.4	80	165	172	42 hours	59 hours
Release v.2.0	160	235	290	71 hours	96 hours

1. Number of tests executed includes new functional tests and regression tests actually executed.
2. Test execution time includes time to execute the tests and to research and communicate defects. It doesn't include delays occurring during the test effort that aren't directly related to executing the tests. For example, if the DBA has to load reference data prior to test execution but is delayed because of a production problem, this type of delay shouldn't be included. On the other hand, if executing a test case requires that a 2-hour batch job be executed, that delay should be included.
3. Required test execution time is the amount of time required to execute all available tests.

Note: It's rare that every test case will be or can be automated. We make this assumption to simplify the example.

**TABLE 2: AUTOMATION, GOOD**

Project ID	Number of New Tests	Number of Regression Tests	Number of Tests Executed	Test Execution Time
Release v.1.0	200	0	200	49 hours
Release v.1.0.1	20	150	170	10 hours
Release v.1.4	80	165	245	25 hours
Release v.2.0	160	235	395	47 hours

My company has found that most development organizations can expect a defect rate of about 25 to 30 percent on new functionality, and 1 to 5 percent during regression testing. Regardless of whether you execute tests manually or use an automated tool, the defect rate can be calculated using this formula:

$$\text{Defect Rate} = \text{Total Failures} / \text{Total Test Executions}$$

Total Failures are the total number of test case failures, including failures of re-executed test cases. Total Test Executions are the total number of test case executions, including re-executions.

Knowing the defect rate won't reveal how much can be saved from finding and correcting those errors during the test phase as opposed to correcting them after deployment. The best way to determine those costs is to examine historical data related to previous releases.

When deriving the costs, it is important to know who is involved in analyzing the preproduction defects—testers, developers, business analysts and DBAs—and how much of each group's time it takes to repair and retest them. For postdeployment, it might be necessary to add users and help desk staff to the list of those analyzing and testing defects.

If you want to use a shortcut approach rather than deriving your own costs, beware! Relative costs increase as defects are found later in the SDLC (see Figure 1).

Keeping the numbers simple, let's assume an average loaded cost of US\$100 for everyone involved in correcting the defect.

Using the information in Figure 1, the cost to repair a single defect is:

- \$500 during test
- \$1,500 during production

Let's go back to our example so that we can apply some of this information to determine the value of test automation relative to the increased number of defects found.

Table 4 shows the number of defects that we expect to be in the application under test, based on a defect rate of 30 percent for new functionality and a defect rate of 5 percent for regression testing.

Table 5 then applies the costs associated with finding the defects in test versus production to the total number of defects found in each phase. The results reveal the difference (assuming 95 percent test coverage of new functionality and 55 percent test coverage of regression functionality for manual testing, and 100 percent test coverage for both new and regression functionality using automation testing).

Based on this information, automated testing offers the following benefits over the course of the four releases in our example:

- 25 percent increase in test coverage
- \$2,304 saved in test execution costs
- \$19,000 saved in defect repair costs

#### Benefit 4: Better Repeatability, Reliability

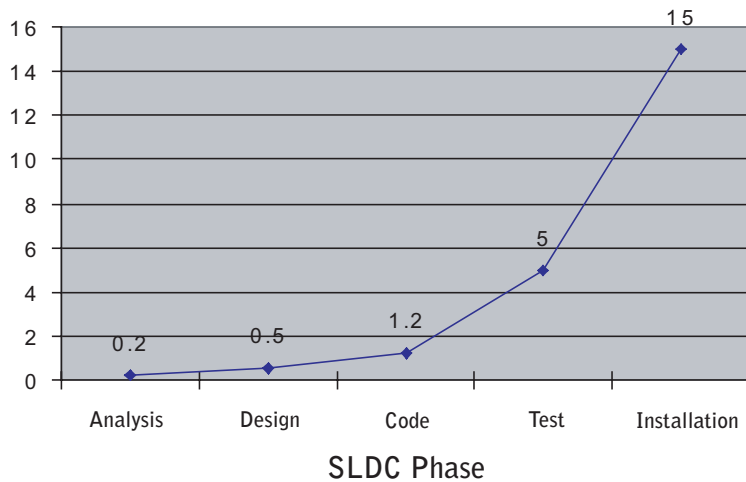
The ability to easily and reliably repeat an automated test suite varies with the type of automation approach implemented. The five approaches are Record and Playback, Data-driven, Modular, Keyword and Database-driven. Each has its pros and cons. Depending on the size and scope of the test automation effort, an inverse relationship often exists between the ease of automated test case creation, implementation and script customization versus test case maintenance.

**TABLE 3: DOLLARS AND SENSE**

Project ID	Manual Test Execution Costs	Automated and Manual Test Execution Costs	Difference
Release v.1.0	\$2,300	\$2,300	\$0
Release v.1.0.1	\$1,250	\$625	\$625
Release v.1.4	\$2,100	\$1,413	\$687
Release v.2.0	\$3,550	\$2,588	\$992
TOTAL			\$2,304

**TABLE 4: EXPECT DEFECTS**

Project ID	Number of New Tests	Number of Defects Expected (30% Defect Rate)	Number of Regression Tests	Number of Defects Expected (5% Defect Rate)	Total Defects Expected
Release v.1.0	200	60	0	0	60
Release v.1.0.1	20	6	150	8	14
Release v.1.4	80	24	165	8	32
Release v.2.0	160	48	235	12	60

**FIG. 1: RELATIVE COST VS. PROJECT PHASE**

the metrics prior to implementing the automation tool. This will establish a substantial comparison basis to measure against. Key metrics to track include:

- Test coverage percentage
- Test execution costs
- Defect repair costs (in each phase of the SDLC)
- Implementation and maintenance cost for the automated solution

With these metrics, the potential savings offered by automating a test effort can be established. Various test automation frameworks also can be evaluated to determine which provides the least expensive overall solution based on the number of test cases to be automated and the frequency with which the application is changed.

Using the metrics in this article and your organization's actual resource costs, you can identify the point of convergence where automation setup and maintenance costs are offset by the savings derived from improved test coverage, reduced test execution time, decreased production errors, and the increased repeatability and reliability afforded by test automation. ☒

**TABLE 5: PROJECTED DEFECTS**

Project ID	Projected Number of Defects Found		Cost Differential
	Manual Test	Automated Test	
Release v.1.0	57	60	\$3,000
Release v.1.0.1	10	14	\$4,000
Release v.1.4	27	32	\$5,000
Release v.2.0	53	60	\$7,000
TOTAL			\$19,000

**AUTHOR'S NOTE**

While cost savings account for the primary value derived from test automation, increased revenue—through shorter development cycle times and faster time-to-market—can also be achieved. However, in many instances, increased revenue is much more difficult to calculate and therefore isn't discussed in this article.

For each of the five approaches, the implementation costs are based on three factors: the time spans allotted for test case creation, customization and maintenance.

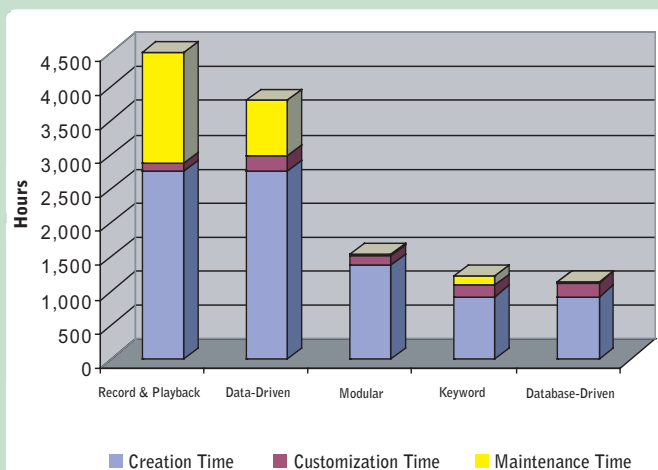
By applying a "rule-of-thumb" estimation model for each of the automation approaches, we can see in Figure 2 that over the course of the four releases, the more advanced approaches (modular, keyword and database-driven) take substantially less time to maintain and to execute than the two simpler approaches (record and playback and data-driven).

From this example, as the number of test cases to be maintained increases and a more frequent release schedule is applied, advanced approaches (which generally have higher up-front costs) more than pay for themselves over the long run.

Because the more advanced approaches tend to be easier and cheaper to maintain, they actually increase the likelihood that the test group will be able to maintain the test suite and use it in subsequent releases. Those approaches also increase the repeatability and reliability of automated tests.

**Metrics First!**

Tracking metrics for an automated test effort isn't much different from tracking them for a manual test effort. In most cases, the same metrics are used. The key to calculating the financial success of an automated test effort lies in tracking

**FIG. 2: AUTOMATED TESTING TAKES LESS TIME OVERALL**

Our "rule-of-thumb" estimation model allowed 4 to 6 hours per test case for (1) record/playback or (2) data-driven automation approaches, 2 to 3 hours for (3) modular and 1 to 2 hours for (4) keyword or (5) database-driven approaches. Allotted customization time per test case was 10 to 15 minutes for (1), 20 to 30 minutes for (2), 4 to 6 hours per module for (3), 3 to 4 weeks for the engine for (4) and 4 to 5 weeks for the engine for (5). Estimated maintenance time was 1 to 2 hours for (1), 30 minutes to 1 hour for (2) and (3), and 3 to 4 hours for engines of (4) and (5) plus 5 to 10 minutes per test case/release for (4) and 1 to 2 hours for the suite of test cases/release for (5).