QualityLogic

WHITE PAPER

# Automated vs. Manual Testing: How to Get the Right Mix

Learn what to automate and what to test manually to increase automation ROI and lower operating costs.

# Table of Contents


## Executive Summary

Balancing manual with automated testing helps control costs. For system code that is continuously changing, manual testing will be less expensive than trying to maintain automated test scripts. The cost of maintaining test scripts and dealing with the chaos created by false positives quickly exceeds the cost of simply using manual resources to perform the same tests. For system code that typically has few changes, automation maximizes coverage while keeping costs low.

The best ROI resides in finding the right balance between automated and manual testing, tailored to fit the development cycle needs of your organization.

# Overview

With eCommerce and entertainment businesses increasingly driven by marketing campaigns, both development and QA have been hard pressed to keep up with the pace set by rapidly evolving requirements. The months-long waterfall product life cycle that used to govern software releases has given way to Agile development sprints that have shortened it to one or two weeks. This has pushed the formerly separate development and QA functions into a 'quality is everyone's job' alliance to release new features and support new content on intensely tight schedules.

Mobile devices are now the dominant method used for web access, which has spurred the development of mobile-oriented sites and spawned their own application (app) market. Even as the mobile device industry created a plethora of OS versions and display distinctions, these devices brought about even greater user demands for flawless, quick-performing applications. Entertainment sites look for accurate, glitch-free rendering of videos and streamed performances.

*"Be sure your apps work on every device…every time with the right mix of manual and automated testing"*

eCommerce sites strive for ever higher conversion rates from smoothly shifting browsing into buying. Resource apps that derive their revenue from advertising have to attract and hold the members of their user communities amid ever-increasing competition.

Integrated development systems are offered to support rapid app engineering, and they include detailed emulators. But, to many engineering groups, these emulators are only sufficient for basic feature debugging.

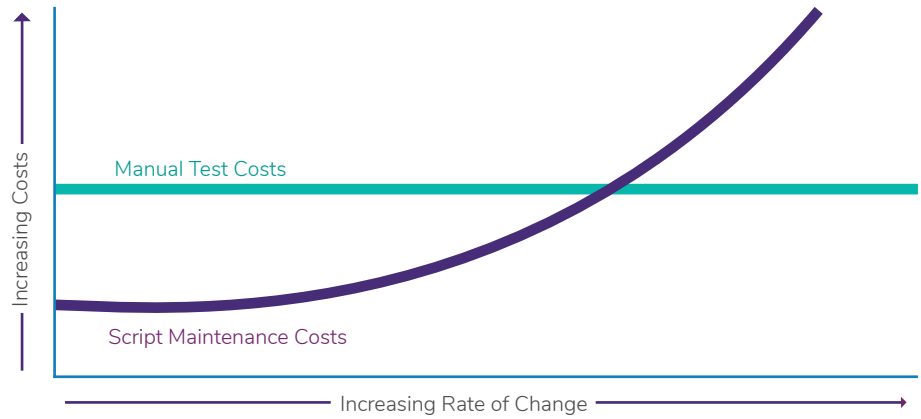
To achieve the quality with the app look and feel that this new market demands, most development organizations have mandated testing on actual devices. To further tighten the screws on the development/test processes, there is now a push for Continuous Integration (CI) to insert code into the system as quickly as features and bug fixes can be developed. CI and the push to test on actual devices have, together, focused attention on the idea of automating test cases. What better way to test the operation of a group of machines than with another machine?

Test automation offers the seductive lure of being able to write a test script once and then run it again and again at the push of a button or, better yet, tie its execution to the output of a build management system. That way, each new version is automatically tested by the system.

From this perspective, a conventional wisdom has grown up regarding mobile application testing that more automation is always better. As attractive as this concept is, it is not necessarily true. Implementing test automation on mobile applications or sites with frequent and significant content changes can quickly reach a point of diminishing returns.

The main issue with automating everything is that a massive body of coded test scripts is created that must then be maintained. This maintenance uses the same resources that would also be used to create new code, commonly resulting in those resources being drained away from script maintenance to code development. The reliability of the automated test suite suffers, its results are deemed irrelevant, and the management that wanted to automate everything now decides that test automation is a waste of time and money.

As with most tools, automation implementation should be viewed as a spectrum that runs in gradations from 'too much' to 'too little.' A far more effective test solution for rapidly changing mobile applications is a hybrid where there is a mix of automation and what we call Scalable Manual Testing. A careful look at the pros and cons of both manual and automated testing will help sort this out.



## Where Automated Testing Excels

Test automation has a number of significant factors in its favor.

To start with, automated test execution is uniform. The test execution host runs each test exactly the same way each time with no variation whatsoever in the test steps or the reporting of their results. A script doesn't get tired and bored with the process of performing the same test cases over and over again.

Automated test scripts are for all practical purposes computer programs and, as such, they support long-term reusability. Test automation projects typically start with the development of script code pools commonly called test harnesses or frameworks. These are script development tool kits that provide reusable procedures which can be incorporated into any number of functional test scripts. A considerable expenditure of resources is required to generate both frameworks and automated test suites but that work can be leveraged across a vast number of quick, low cost executions.

*"Get the computer to do what computers do best and get people to do what people do best."*

– Dorothy Graham, Software Test Consultant
StickyMinds interview

Data-driven test scripts facilitate testing boundary and out-of-bounds conditions by offering the opportunity to specify entire ranges of variable values. Tables with sometimes interdependent values can be read during script execution to provide inputs that drive the feature under test across its design range, up to the edge of that range and substantially beyond it. This can reveal gaps in the intended operational span of the feature as well as verifying its graceful recovery from erroneous inputs.

Along these same lines, test automation supports multiple executions of large test arrays and complex tests. A wide variety of minor operational distinctions can be exercised by creating many copies of a test script and making subtle changes to each copy. A function that is coded in very similar but slightly different designs over a large span of instances can be tested thoroughly with a relatively small incremental effort to generate the necessary scripts. Additionally, complex test setups that require database record modification, value propagation through multiple levels of a decision tree or simply filling in a complex form over and over again are prime candidates for automation.

And finally, test automation compels more rigorous test design. Both feature code and test cases must be designed with care and foresight in order to support automation of the test process. Further, the script itself must be carefully thought out in order to properly execute those cases and test their associated code. You can't automate what you don't already know how to test.

## Where Automated Testing Fails

With the foregoing supporting the idea of automated testing, what objections could possibly be raised against it? Well there are actually a number of downside issues with automation not the least of which is up-front cost. Test automation tools can be really expensive. They are most commonly sold on a per host installation basis with prices running anywhere from $1,000 per installation to as much as $5,000 and these are fees that have to be renewed each year.

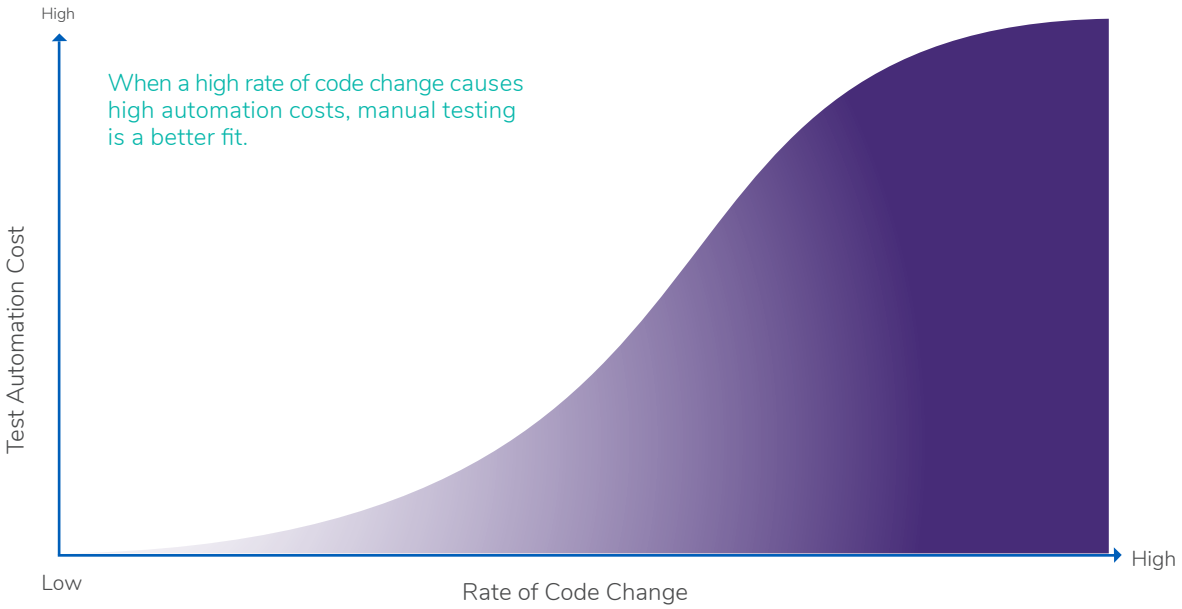
One way around this is to use one of the Open Source tools that are available with no license fee. Unfortunately, they tend to be difficult to install and maintain for anyone who is not a skilled code engineer and their tech support is provided by user forums rather than a paid staff. Either way, both proprietary and Open Source tools have associated learning curves. The implementation and use of such a system is not much different from learning a programming language and, indeed they bear a strong resemblance to code development systems.

*"Automated testing doesn't make sense when a test team does not have the expertise to automate correctly."*

John Overbaugh, Microsoft Sr. SDET Lead
TechTarget SQA blog

Which brings up an issue mentioned earlier. Automation test script development requires the same skills and effort as code development. The QA engineers who create, debug and deploy automation scripts have the same job descriptions as feature programmers with the additional requirement that they must have the mindset that comes from software test experience. Not only are valuable resources employed in creating test scripts, their job is not over there.

A major issue with test automation is allowing test scripts to become outdated as the code they are intended to test is modified and improved. Test automation is a full-on development project that requires the same kind of organizational commitment as the development of a product, and it has to have dedicated personnel resources who are not diverted to other 'more profitable' tasks.



Even at its best, automation does a poor job of verifying screen formats and video displays. For example, a button that activates a feature may correctly perform its purpose and be correctly labeled. But the label could be the same color as the button, making it unreadable and a functional test script would not know to look for that problem. Programming a test script to assess the visual quality (color, image/sound coordination, smooth frame delivery, etc.) of a video presentation is also a considerable challenge and is typically considered impractical to automate.

After the set-up hurdles have been overcome and the automated test suite is up and running, someone has to examine its results. Automated tests, however well maintained, will result in some percentage of both false positives and false negatives. These test results have to be examined with the positives (test failures) carefully scrutinized to see that they really point to defects, and the negatives (tests that passed) spot checked to make sure that the test is still verifying the code operations that it was designed to exercise.

In addition, the automated test results have to be reviewed in the context of the test series that was being performed. This can require considerable backtracking through the test suite in which the defect indication occurred. This isn't an issue with manual testing because the tester is working through the test series and knows exactly what happened leading up to the defect indication.

If test automation is beginning to sound less and less like an unqualified winning plan, an examination of the ups and downs of manual testing is in order.

"A major issue with test automation is allowing test scripts to become outdated as the code they are intended to test is modified and improved."

## Where Manual Testing Excels

A primary benefit of manual testing is that the short-term costs are fairly low. There are no license fees to pay and the main startup effort goes into writing the test plan and cases. Often there is no appreciable learning curve associated with the system under test as the tester's ability to quickly figure out how to use it is the usability part of the test itself.

Since manual tests generally require lower skill level personnel than those required to write automation scripts, there is substantial savings there as well. And this makes manual tests easy to scale to whatever effort level is required.

If new releases show up in a group, a larger number of test techs can be added to the project on short notice to process the tests en-mass. A small group of QA leads can manage a fairly large group of testers to quickly come up to speed on a system and rapidly process a large number of test cases.

*"Exploratory testing is [an area] where humans still retain an edge. Likewise, if you are testing for something that is inherently a human-perceived quality, such as usability, you need humans in the picture."*

Kevlin Henney, Independent Consultant & Trainer
TechTarget SQA Blog

Another advantage is that manual testing can be extremely flexible in response to changes in the system under test, such as feature enhancements. When a feature has to be dramatically revised at the last minute to accommodate an unforeseen issue, the test plan can be modified quickly to accommodate that change with little or no disruption to the test schedule. By the same token, a radical bug repair can be accommodated by adding a tester to the project and handing him or her a list of test cases to verify the fix.

Manual test projects can roll with rapid release cycles that incorporate system-wide changes. When half of the test cases change to suit such an integral change, the test plan is rewritten to accommodate the new code and the team sets about executing that plan. There is no lag induced in the QA response and, most importantly, no additional massive cost caused by having to rewrite automated test script code.

Probably the single most valuable aspect of testing with a human technician rather than an automated test script is exploratory testing. When a defect appears in an automated test, the test system notes the result and goes on to the next test. A tech works through a test plan building a mental context of how the system under test works and how it should perform.

When the tech discovers a defect, that context points to its collateral effects and suggests other features that need to be examined in light of the failure that this test reveals. This prevents multiple bug reports caused by the same issue and may point out other problems and dependencies that a simple execution of the original test case would have missed.

As mentioned above, having a technician perform a manual test suite verifies the human usability of the system. An automated system, however well designed, will not catch the fact that an on-screen instruction to the user is poorly worded and misleading. Nor will it recognize that the sequence of using a particular feature set is counter intuitive and will ultimately aggravate the user by making use of the system unnecessarily difficult.

*"Having a technician perform a manual test suite verifies the human usability of the system. An automated system, however well designed, will not catch the fact that an on-screen instruction to the user is poorly worded and misleading."*

## Where Manual Testing Fails

Flexible as it is, some tests do not lend themselves to manual testing. A principle example is direct-code tests of system aspects such as APIs. These are difficult to test manually because of the requirement for ancillary parameter management and manipulation of database contents, and the fact that there is no user interface for a tech to directly interact with. There may also be a succession of test steps that have to be performed, each of which is based on the outcome of the last.

Other examples of this issue are load, performance and soak testing. These all require high intensity inputs representing dozens to thousands of users and access threads. It is neither efficient nor feasible to perform these test types manually as it would require too many test techs coordinated over too short a time.

*"When testing requires a methodical and repeated execution, that is better offered by machine than human. Humans are good explorers, but compared to computers they are incredibly poor at executing loops."*

Kevlin Henney, Independent Consultant & Trainer
TechTarget SQA Blog

Another problem with manual testing is that the results achieved depend on the focus of the tester. If the test tech is working through a series of test cases that exercise a login process, that tech's focus is likely to be on the data entry and response process. It will probably not be on the banner that changes at the top of the screen and has a format error in its third iteration.

The bottom line is that test techs are human. They will make mistakes in test case execution, results evaluation and defect reporting. Having a QA lead review the efforts of a tech test team will eliminate a large proportion of these kinds of oversights but there will still be a few.

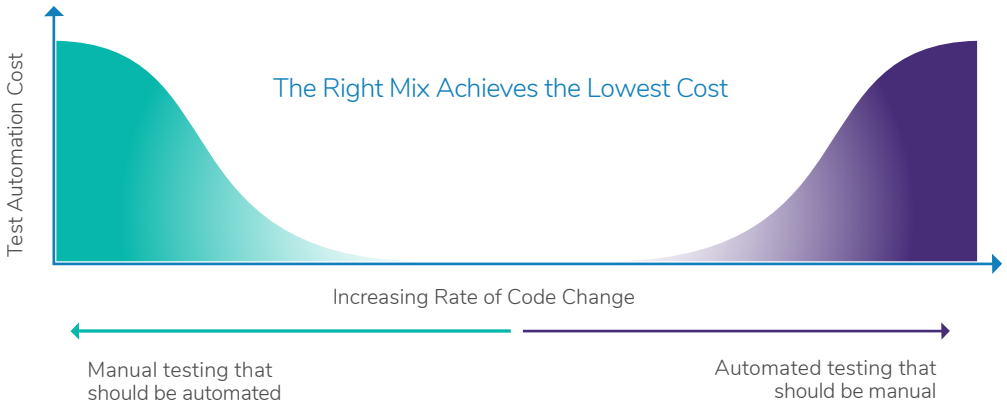
And techs perform tests at human operator speeds. Pushing them to process test cases faster simply results in more of the procedural errors described above.

And finally, while test cases are reusable, manual test effort is not. The time consuming aspect of automation is writing and maintaining test scripts. But a script can be executed continuously. Manual test effort is an expended resource that is fully consumed each time it is used.

# Finding the Balance

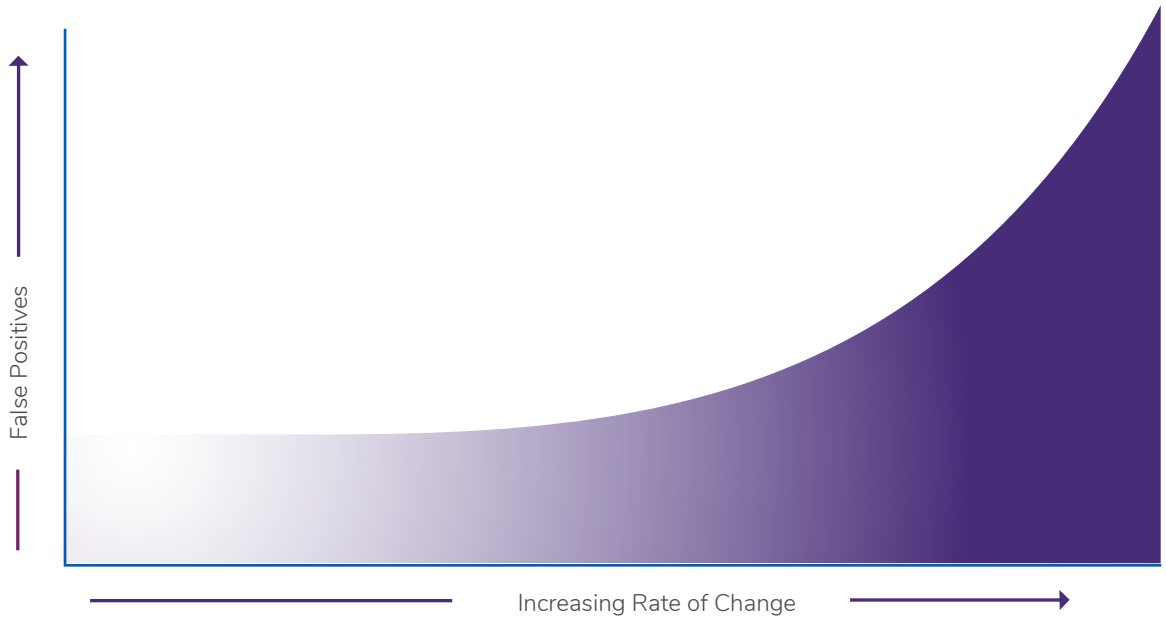
Manual testing has the process flexibility to rapidly adapt to continuous, wide-spread system change, follow rapid release cycles and keep up with Agile development methods. Automated test scripts support deep test complexity, eliminate human error and offer fast execution times. Determining the mix of manual versus automated testing requires evaluation of several interactive aspects of the system under test.

A powerful driver behind mobile application test automation is continuous integration of code changes into the software development life cycle. Rather than having staged releases weeks or months apart, daily builds are done on the work in progress, typically with automated test sequences run as part of the daily software builds. This works really well up to the point where the rate of change overwhelms the ability to keep scripts in sync with the builds. The resulting test failures at each build cycle, mostly false positives, tend to get ignored without investigation and the effectiveness of the entire automated testing process gets compromised.



The code change rate versus the occurrence of false test script results is a primary consideration. Core functional testing for rapidly changing mobile applications is most effectively done using manual resources. Attempting to synchronize updates to automated test scripts in parallel with rapidly changing application code creates a chaotic situation where each regression test has a large number of failures, most of them false positives (tests marked as failed that in reality should have passed).

Apps that are more stable will typically support some level of test automation with the most common being a sanity check to make sure that code changes don't disrupt basic functionality in apparently unrelated areas of the system. This is the operational balance point of mobile test automation. Is there enough stability in the rate of code change to make a restrained level of test automation feasible?



Automation also provides a measurable level of test coverage that is documented by the test scripts as compared to the feature list. A rapidly changing system will cause enough uncertainty to defeat this coverage measurement and require a manual test process to keep up with the changes.

Another major issue is cost control. Systems that have a high feature change rate will be poor prospects for test automation. The process of creating and maintaining test scripts will meet or exceed the costs of the system development effort. A manual test project can roll with these changes and be flexed in both its content and staffing.

*"It only makes sense to use automated testing tools when the costs of acquiring the tool and building and maintaining the tests is less than the efficiency gained from the effort."*

John Overbaugh, Microsoft Sr. SDET Lead
TechTarget SQA Blog

## Summary

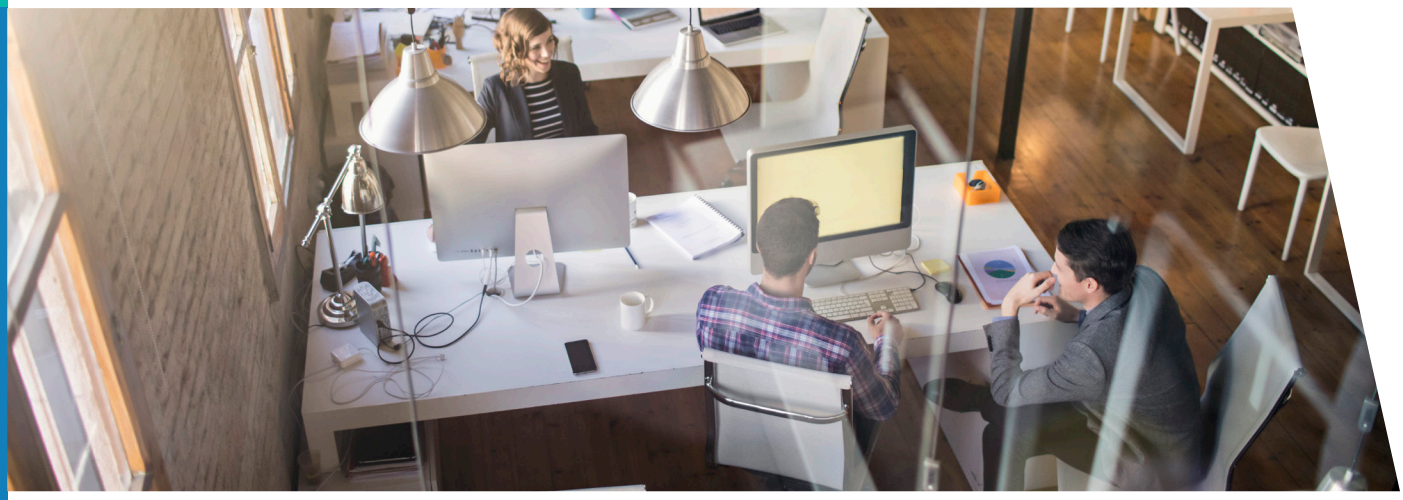
A thorny issue with comparing manual testing with automation is quantifying their differences. It is one thing to make qualifying statements like those above but quite another to actually assign numbers to these assertions.

Unfortunately, the pursuit of test automation tends to become a binary issue. Too often it is either seen as the silver bullet that will solve the testing bottleneck or a waste of precious resources that would be better applied to developing revenue-generating code.

In the mobile application arena, automation has its place but it needs to be recognized as being only a part of the solution. In the rapid code change environment that mobile development projects tend to be, test automation needs to be balanced with the flexibility offered by manual testing.

*"Cost, time and quality are not independent…You cannot be successful with one unless you are successful with the other two."*

John Scarpino, Director of Quality Assurance & University Instructor
TechTarget SQA Blog

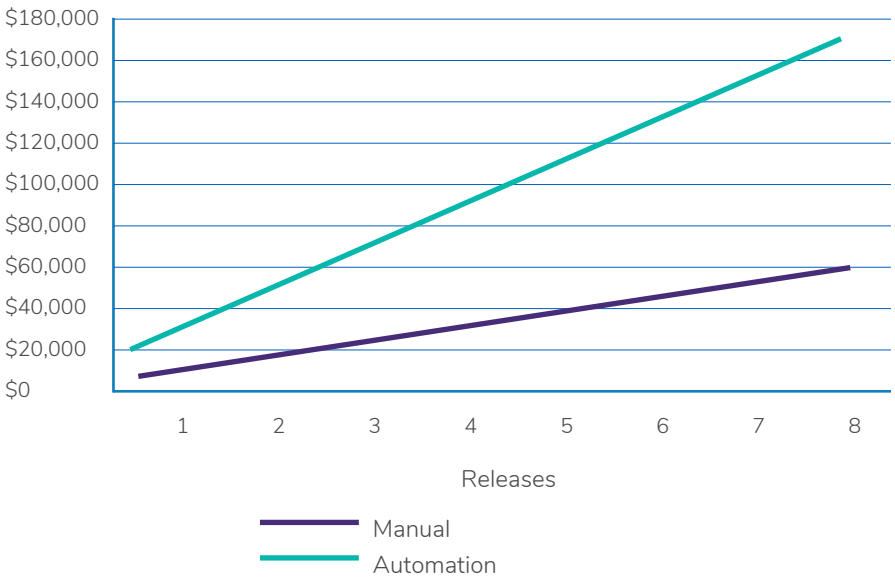


## Finding Your Balance Point

To help determine where your unique balance point lies, QualityLogic has created a spreadsheet calculator for directly comparing the costs of both approaches to a test project.

There are many ways to look at the tradeoffs between manual and automated testing. The following graphs offer a sampling of those tradeoffs using the ROI calculator.

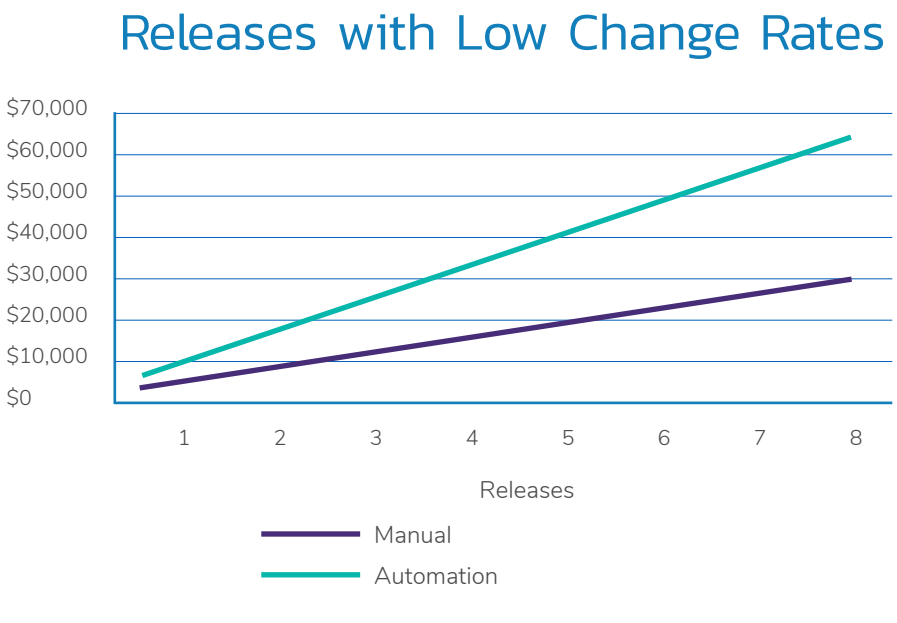
### Releases with High Change Rates




Large amounts of change between releases will cause much higher cumulative automation maintenance costs across multiple releases as compared to the cumulative cost of performing those same tests manually.

The calculator allows the user to input assumptions, such as test case scope, device use, development and execution times, and labor costs. It then asks for parameters for the scope of the test project, such as base, new, retired and revised test cases, and the platform dependency of automated test scripts It will then calculate labor costs across eight project revisions.

Our recommendation for applications with a high rate of change is to create a very small set of automation routines that can be easily maintained, regardless of the rate of application change, with the goal of catching catastrophic problems as quickly as possible, while offloading the testing of new and existing functionality to the manual testing process.




Code with smaller amounts of change between releases will cause much higher cumulative manual testing costs across multiple releases as compared to the cumulative automation maintenance costs when automating those same tests.

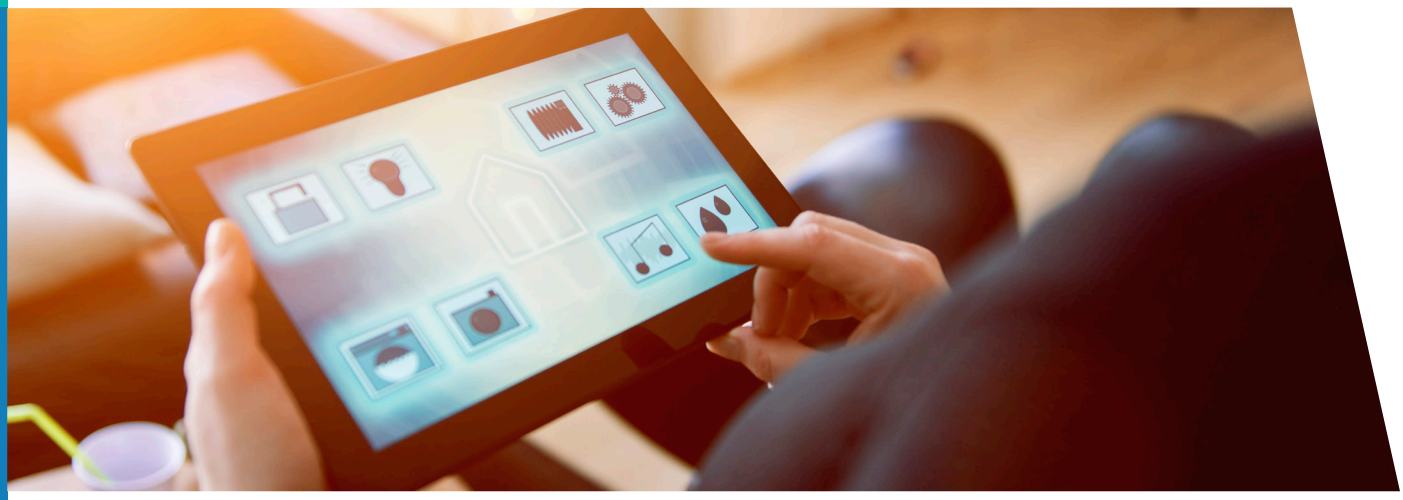
## Scalable Manual Testing

QualityLogic's Scalable Manual Testing for your high change rate mobile applications offers:

- More effective test coverage without the false positives
- Scalable and flexible test resources
- Continuous Integration support
- Lower costs

For areas suited to cost-effective test automation, QualityLogic provides resources to automate your tests. Our veterans of the test automation process are skilled with the latest tools and have the time and expertise to thoroughly plan out development of the automation test harness, and then execute those plans.

We work alongside your developers and QA staff to make sure the test cases keep up with the development work. And using QualityLogic's engineers means you won't have to pull your engineers off of their development work.



The alignment of manual testing and continuous integration can be achieved by using the Scalable Manual Testing process. Key components of this process include:

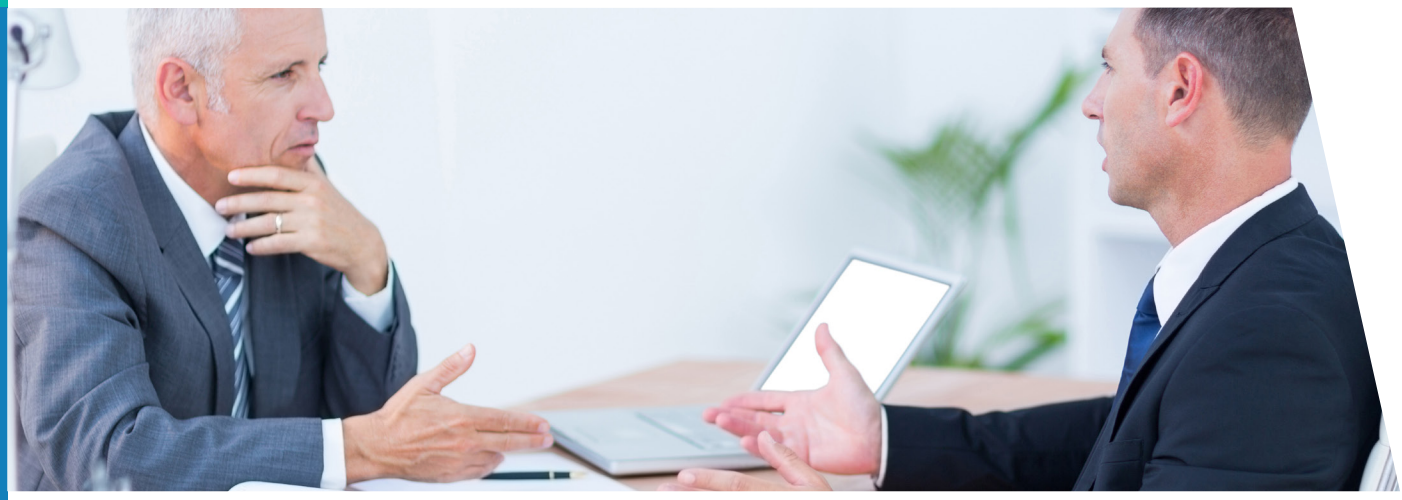
- A structured test procedure that includes written manual test scenario outlines
- The ability to deploy large numbers of low cost on-shore test technicians on a high-availability testing basis
- A substantial library of on-site mobile devices to eliminate device availability issues common with cloud-based device resources
- Processes and tools to rapidly report bugs, including crash logs and other system information provided by test technicians with excellent communication skills
- The ability to test in diverse domestic and international markets over live regional carriers
- Mobile device and test script selection processes that optimize test execution times without compromising coverage

Combine a judicious use of test automation for monitoring major functional integrity with QualityLogic's Scalable Manual Testing and you have a winning QA strategy for your mobile development projects.

## About the Authors

John Lunsford is QualityLogic's Chief Test Engineer and brings a strong, diverse background in software, peripherals and quality assurance to QualityLogic's Boise, Idaho lab. John has accumulated 40 years of experience in the computer industry, including 10 years each in software development, software quality assurance and automated systems implementation.

Jim Zuber is a co-founder of QualityLogic and serves as CTO and Chief Test Architect. Many of the testing products he architected for QualityLogic have become de facto testing standards in the smart grid, imaging, facsimile, and telephony industries.




## About QualityLogic

QualityLogic is a highly respected provider of QA and engineering services, offering a flexible menu of services that scale to meet customers' needs.

The Company is well known for its quality assurance and competitive analysis expertise in the printer and telecom industries. Fortune 1000 companies also rely on QualityLogic to make sure the content of their primary marketing tools – their website and mobile applications – meets their quality objectives and promote and protect their brands.

## For More Information

Visit www.QualityLogic.com or call +1 208-424-1905