Advice for Getting Started with Test Automation

Learn how to get started with test automation to shorten your dev cycles, avoid repetitive tasks and improve software quality.

White Paper

optimus information

A Software Tester's Work is Never Complete

It is a truism that software testing is never complete. Testing can only confirm the presence of bugs but says little about the defects that undoubtedly remain, even in the shipped product. However, no one is throwing up their hands and saying that we should forget about testing because it is ultimately a futile effort. In fact, the opposite is true, since it is an activity not of absolutes but of probabilities.

The more testing that is accomplished, the higher the probability that customers will be pleased with the final product. No one expects any software to be completely bug-free, but they do expect it to be 99.5 percent useable, to perform well, to not crash, to retain their data and to play nice with other apps.

Unfortunately, most QA organizations are understaffed and in danger of severely falling behind, which impacts not only time-to-market but the quality of the released product. One way in which test organizations are dealing with this situation is to automate as much testing as possible. This is especially effective for repetitive test scenarios such as regression testing but can be expanded to other areas.

Automation, if done correctly, can improve the efficiency of test teams by replacing many time-consuming, error-prone manual test procedures and by increasing test coverage. Automation tools can also generate test cases, produce test scripts and increase collaboration opportunities with development teams, which speeds up the bug fix cycle.

Test automation is not a panacea, but instead a vital tool in QA's arsenal. It cannot be applied to all types of testing, nor is it likely to reduce the number of bugs developers introduce, but it can help avoid extensive application rework, especially if applied early, and reduce support costs once a product is in the field.

Types of Automated Tests

Unit Testing

The main goal of unit testing is to test the smallest meaningful code units in isolation. These units are typically individual classes or functions. This type of code testing is the first line of defense against defects. The original tests are often performed by the developers themselves as they produce code. There are many script-based tools, some of which integrate with IDEs, to create these tests. The individual tests can be aggregated into larger scripts or called from the application build process. Eventually, the unit tests are incorporated as part of a suite of regression tests.

Regression Testing

As application code continues to be produced, it becomes subject to a level of complexity that introduces fragility. This complexity continues to increase for the application as a whole when components from different developers begin to interact. At this point, even small tweaks in the programming have the potential to break the application, revive bugs thought to be dead, or turn up other unwelcome side effects in behavior.

In order to detect such defects, regression tests are typically run after every daily build of the software. This process would be laboriously tedious if it were done manually. The best time to run the build and tests is after developers are done for the day and have checked in all changes.

Black Box Testing

Black box testing, also known as functional testing, is only concerned with showing that an application can produce a predictable set of outputs for a given set of inputs. It is not concerned with how the results are derived internally by the application. Correct outputs also include the absence of error messages or other erroneous behavior.

This kind of testing is a good candidate for automation, since it can be data-driven using inputs from files, spreadsheets or even random input generators and the results automatically compared against known good outputs. Not all kinds of black box testing can be automated, however. End user testing is another black-box type, but requires humans to supply the inputs.

Code Coverage

Code coverage testing's purpose is to quantify how well test cases are penetrating all paths through the software logic. To perform this type of testing usually requires instrumentation of the source code via compile-time switches. This is a type of white box testing because it is

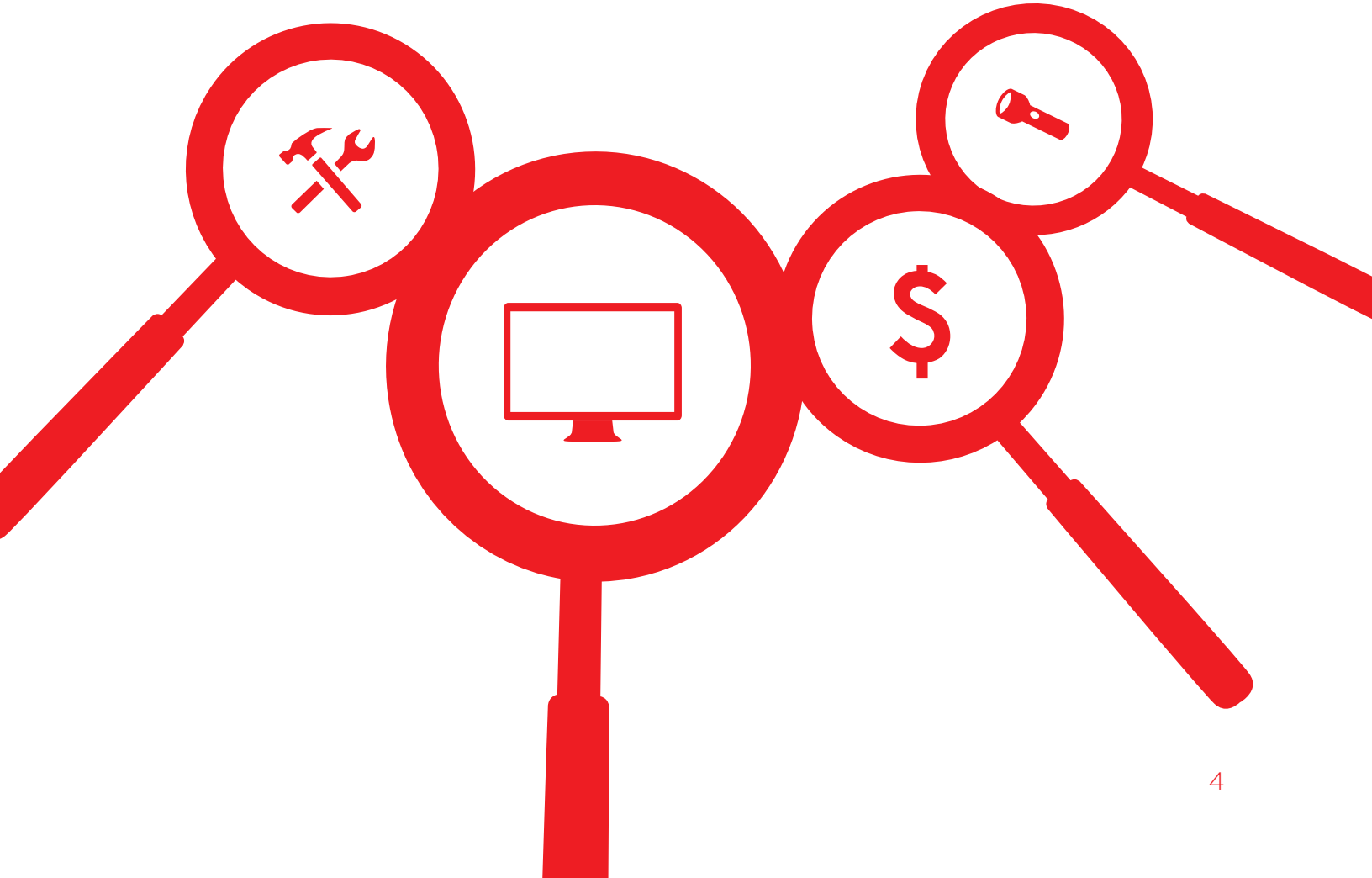
measuring internal effects, not outputs. Insufficient coverage of the code can mean one of two things: 1) the test cases need to be augmented, or 2) there is "dead code" in the application that will never execute and should be removed.

Load Testing

Load testing is essential for all software applications but especially for mobile applications, which often work under severe hardware constraints such as reduced memory, network latency and low power requirements. Native and web-based applications both benefit from automated load testing as the inputs are combinatorial, which makes it impossible to achieve sufficient coverage manually. This is especially true if the inputs are driven from GUI controls.

Compatibility Testing

It is rare that software is targeted to run on a specific make and model of hardware. Thus, the software must be tested on various kinds of computers, OSs, networks and various peripherals attached. This is another area where automation provides improved test coverage. To test on a wide array of devices, especially mobile devices, cloud-based automation tools avoid an organization having to acquire and maintain in-house all of the hardware, OS versions and so on on which to run compatibility tests.



Six Pieces of Advice for Getting Started with Test Automation

1 Calibrate Expectations

Especially if test automation does not currently have deep penetration within the QA department, realistic expectations should be set as to what automation brings to the table. In general, it should not be seen primarily as a way to shorten development cycles but rather as a method for increasing the ultimate reliability and overall quality of the software product. Its main purpose is to increase and improve overall test coverage and thus lower the probability of failures.

It should also be kept in mind that not everything can be automated or at least not automated effectively. Some tools assist with mundane tasks such as test requirements, test case generation, test scripts and maintaining a test library, but their real strengths are aimed at batch execution, data-driven tests. The ability to take advantage of that depends somewhat on the type of software being tested and the skillset of the QA team.

If the organization is relatively new to test automation, then the effort to train testers and integrate the tool or tools into the current development process must also be taken into account. This process will be initially disruptive but gradually pay dividends over several months. A step-wise approach is best.

Finally, what test automation can never do is fix a broken QA process. In fact, it is even possible that it makes it worse by diverting focus from implementing best practices or by creation tension within the department. Before considering automation, be sure that the QA department is up to snuff with regard to all other aspects of their test process before introducing another factor of complexity.

2 Start Small

It is tempting once a comprehensive automation tool is introduced to a department to integrate as much of it as possible in the shortest amount of time. This is likely to cause congestion in the QA department and affect development teams, especially those that practice agile methodologies. Far better is to apply the tool to a new project and, if the automation tool has the capability, apply the tool to unit testing automation before moving on to regression tests and so forth.

Furthermore, the tests themselves should be small and focused. This is good practice whether or not automation is present. Compact tests themselves can be debugged more easily and are more portable and reusable for other types of testing. The last thing that is needed is a huge testing program that itself is buggy.

3 Organize Tests and Functions

Identify the types of tests being produced and under which category of testing they fall, such as unit, functional, white box, load, etc. This test organization should correspond to the original documents outlining testing requirements, test cases and the overall test plan. All tests should be kept under versioning control as well.

There are always a number of test or supporting scripts that can be tuned for reusability, such as functions used for logging, initializations, data comparisons, conversions and so on. These should be kept separately from other test suites. This practice saves time as tests are developed, new projects arise or when new testers come on board.

4 Design Tests for UI Resistance

Testing UIs is tricky even without automation. Manual testing by having a human press buttons, icons, resize windows, etc. is time-consuming and error prone. It becomes impossible if the application is intended to be cross-platform compatible. Automation tools can assist by recording manual UI tests into test scripts. The results should be accurately repeatable by other testers or developers and can be incorporated into bug reports. However, such tests may fail when even the subtlest changes are made to the UI depending on how the tool is encoding UI components.

When such tests break, it is usually because the underlying tool is recording the UI interactions as location coordinates, which may not translate if a UI object is moved or the screen changes size or orientation. The way to avoid this is to give all UI objects permanent, unique names. This is a type of instrumentation that has to begin in the development team. Choosing automation tools that support naming UI objects when recording manual UI tests can avoid this and add a great deal of automation to what is often a manual process.

5 Support Your Automation Engineers

Testing UIs is tricky even without automation. Manual testing by having a human press buttons, icons, resize windows, etc. is time-consuming and error prone. It becomes impossible if the

application is intended to be cross-platform compatible. Automation tools can assist by recording manual UI tests into test scripts. The results should be accurately repeatable by other testers or developers and can be incorporated into bug reports. However, such tests may fail when even the subtlest changes are made to the UI depending on how the tool is encoding UI components.

When such tests break, it is usually because the underlying tool is recording the UI interactions as location coordinates, which may not translate if a UI object is moved or the screen changes size or orientation. The way to avoid this is to give all UI objects permanent, unique names. This is a type of instrumentation that has to begin in the development team. Choosing automation tools that support naming UI objects when recording manual UI tests can avoid this and add a great deal of automation to what is often a manual process.

6 Carefully Select Tool Features

When evaluating automation test tools, weigh their feature sets against the commitment of the QA organization to adopt automation. If commitment is low, purchasing a tool that promises to do everything is likely to be overwhelming and meet heavy resistance. Even if commitment is solid, choose a tool whose features can be introduced step-wise to smooth integration.

Here are core aspects to consider when evaluating tools:

- The number of testing types it is able to automate
- The set of OSs, development libraries and hardware platforms it supports
- Test script generation that does not require programming skills
- Test record/playback/logging capabilities
- Ability to do inter/intra test check-pointing
- Report generation capabilities
- The set of IDEs with which it can interface
- How many types of test data sources it handles
- Details of how it supports automated GUI testing

Be careful of oversells and upsells when talking to test automation tool representatives. If features are missing or inadequate, try to find another tool to do the job instead of relying on a salesperson's promise that the next release will cover that feature. If at all possible, obtain a trial copy of the software and give it a thorough workout.

Summary

In order to produce a quality software product, an efficient and effective test program is equally as important as the development effort. Bad testing is better than no testing, but ultimately what is to be avoided is having the software tested by paying users. The consequences of that are far more severe than what it would cost to put in place a top-notch testing organization in the first place.

Test automation improves the QA process by providing the ability to quickly run frequent, repetitive, data-driven test suites that provide far greater test coverage than could be obtained manually. Full-featured automation tools also improve test generation, script generation, logging and collaboration with development teams. They can also be used to expand compatibility testing over more OSs, runtime libraries and hardware platforms.

These are testing activities that best lend themselves to automation:

- Capturing test cases and automatically generating scripts
- Runtime code coverage testing
- Unit testing, especially if it integrates with developers' IDEs
- Large, repetitive testing that must be done regularly such as daily regression tests
- Manual testing that is slow, difficult to record and is prone to error
- Compatibility testing over multiple platforms or devices, especially for mobile apps
- Load testing that requires combinatorial input patterns

The more testing can be automated, the more QA can redirect resources to testing other software aspects that require human analytical skills to perform effectively such as usability testing. As the quality and coverage of an enterprise's QA improves, so does its competitive edge.

About Optimus Information

Headquartered in Vancouver, Canada with delivery centers in Canada and India, we work as a trusted partner to medium and large businesses to solve their software and technology challenges. With a team of 150+ people Optimus Information provides global organizations with scalable, flexible and cost efficient solutions. **Optimus Information provides global reach with a local presence.**

604-736-4600 | **info@optimusinfo.com** | **www.optimusinfo.com**