Will Record and Playback test automation work for you?

By Subject7 and Matthew Heusser

Executive Summary

Record and Playback tools are a popular way to get into test automation as quick to start and with little training. Their long-term success rate, and real impact on delivery cost and quality, are more debatable. We find the tools work best with small, stable user interfaces and software, and when used by one person during tests. An immature or rapidly changing user interface may be best served by humans doing testing. As the application scales to multiple testers, teams, complex data set up, or complex evaluation, it makes more sense to test with Adaptive visual tools or using Software Engineering tools. The software engineering tools are essentially custom-built frameworks, and require highly skilled testers. We liken this to creating new software to test the actual software you intended to test. This report helps your group decide if they fit into the sweet spot of Record and Playback, along with the best way to get started.

Getting started - finding your tool

While some people stumble onto test tooling because they see an opportunity, most arrive at it because they have a problem. The actual cost of testing, as a percentage of software spent, varies widely. Capgemini pegs money spent on testing activities as up to 26% of the software project budget[1]. Whatever the true cost, most people can agree it is too high. The alternative, to let bugs slip through, might work for some startups and Silicon Valley darlings. From our vantage point at Subject7, we see that the era of the "Facebook Effect[2,3]", where people accept a perpetual beta in a service free to them, has ended. Today Facebook and Twitter are real companies, with stock for sale and annual reports. People that use the web to book a hotel, purchase a book, or transfer money expect the process to work seamlessly, even over wireless phone networks, on any device.

And testing is too expensive.

Not only is testing too expensive, it becomes even more expensive over time. A simple application built in ten weeks could be nearly twice as complex in twenty, and double again in twenty more. When the software can be tested in an hour on first release, it may take days to cover reasonably well after a year or two. We find companies want a tool to manage the complexity and keep the test effort stable. If system testing can run in under two hours thanks to automation, it can be part of the build, and problems introduced today can be found today and fixed tomorrow. Beyond that near-real-time window, problem identification and resolution increasingly slows down development and cripples delivery cycles. The call to get a tool is clear.

There are perhaps four major paradigms, or ways of thinking, about test tooling. The first, fastest, and easiest for a non-technical person to get started, is by running the user interface while a tool compares results, often called Record and Playback. Another is to write a computer program that drives the browser or phone as an object, which we will call Advanced Automation. A third is somewhere in the middle, combining some of the visual and spreadsheet-like qualities of record and playback, along with software engineering tools like code reuse, better versioning, and allowing the tests to change on-demand during a test run to adapt to a changing user interface. We call this third form Adaptive Test Automation, as it can enable more software engineering approaches (like version control) without sacrificing readability. We put other tools that might compare screens at different resolutions or under different browsers, in a fourth category. Those can often find compatibility bugs but rarely the "tip of the spear" to find core functional issues.

Today, we are focused on the method of Record and Playback as a testing method, to find out if it is a fit for your organization. Record and Playback is tempting, as it is easy to get started without a great deal of technical skill or even support from the software itself. The advice provided here will help you decide if Record and Playback is right for your project.

Factors bearing on your decision

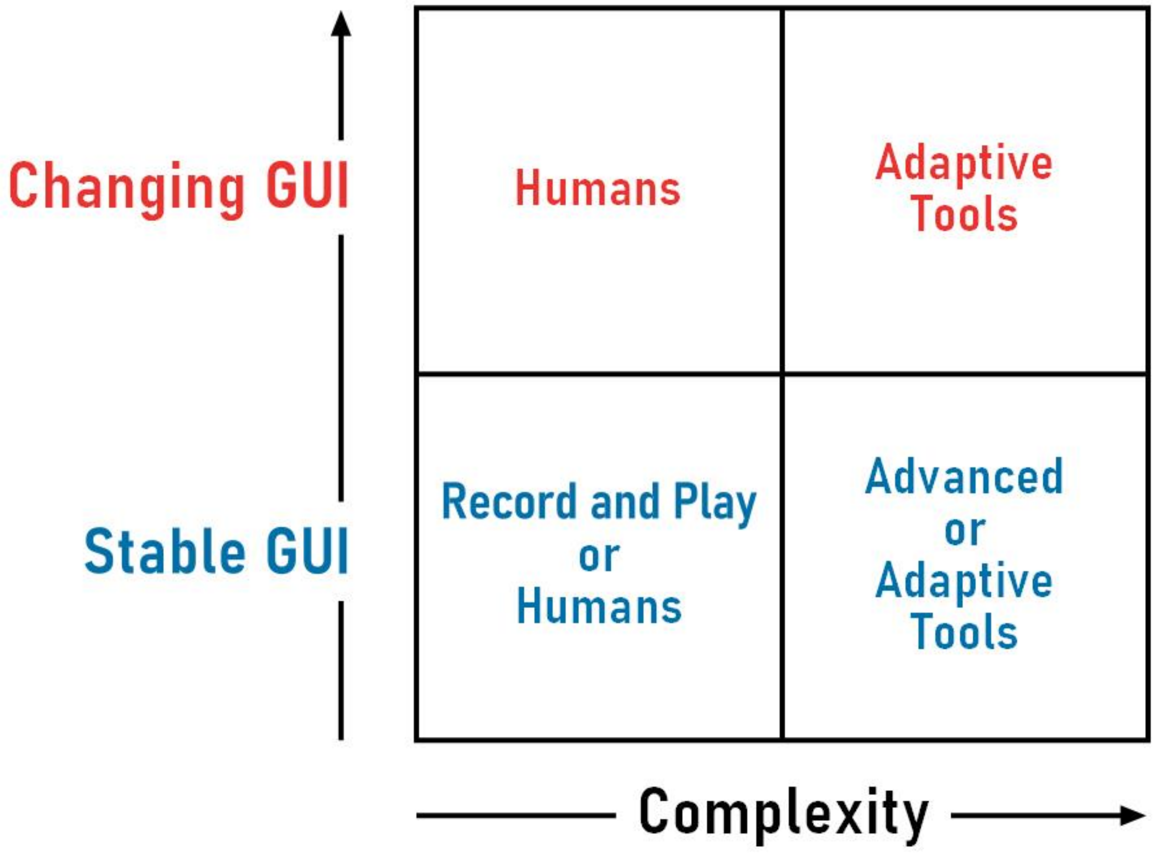
The complexity of the application and the pace of visual change are the two major factors to consider when looking at Record and Playback as your testing option. By first considering these two factors, you can determine which path to take for testing.

Complexity of the application. Some teams support dozens of very small applications, each of which solves a very different problem. Others build a single, simple application at a time and give it away to a client, then go work on the next. In both of these cases, Record and Playback can provide an additional, new form of visual documentation of the application, and even be used as an instruction manual. It demonstrates to the customer what the software should do and how it should be used. While we endorse that approach for small and simple applications, in complex software, the sheer weight of maintaining the automation can become more expensive than the value the tests results provide.

Pace of visual change. When a Record and Playback scenario fails to achieve the expected result, it likely needs to be re-recorded. That might not be a "failure"; it might just be that the programmers made a change to the user interface. A button moved, the text changed, or a new text field was required. If that does not happen much because the user interface is stable, and the team holds back on creating new automation until new features are solid, then this approach can have some value. The greater the pace of visual change, the more needless re-work that will be required.

That leads us to this diagram.

Selecting Software Testing Method



There are plenty of exceptions to deciding what type of testing is best. We'll provide some additional commentary to help you decide if, for example, the application is simple but the organization makes it complex.

Additional Factors

Reality has a way of not quite fitting into an easy two-by-two decision matrix. Here are a few additional factors that might tip the scales to one side or another.

Number of overlapping users. A single user can create and run tests rather easily. This is a scenario where Record and Playback can be the most useful. As we introduce multiple users trying to modify the same "test," we run into challenges. Under the hood, Record and Playback tools are still code, and the tools need the capability to maintain two or more different user spaces, then reconcile those user spaces through version control. Software engineers use tools like Git to accomplish this. As a result, Record and Playback users typically "step on each other" as they are making changes, or else develop an informal tradeoff system of who will work on which system. Combine that with code that is version controlled and branched in yet another system, we can run into a real version control problem. Version control operations are usually complex. The simple "easy to use" properties of Record and Playback quickly diminish as users share assets, which leads to version conflicts.

Team Size/Number of teams. As we discussed earlier, if more than one person is testing the front-end, this can lead to conflicts when sharing Record and Playback systems. Most of these systems really only allow one person to modify a scenario at a time; otherwise, the two users will "step" on each other when they save. Even if people are working in different scenarios, a good record playback tool will store the shared objects in an *object repository*. That way, when the object changes the expected changes only need to be made once. This invariably leads to redundant objects, version control conflicts, or both. Scale this up to multiple teams without clearly defined boundaries and the problem only increases.

Setup and deploy efforts. Test automation needs software to run against, but also data to run against. For example, in eCommerce, search tests need to have known seeded test data. As of today, a book search on Amazon for "how to reduce the cost of software testing" returns four books. Hard code four "4 results for" into the expected results and the test will fail in production when a new book comes out, or in test when new test data is copied from production.

Verification points. Record and Playback needs to do more than click buttons and type numbers. It needs to check expected results and lots of them. In some cases, the results might not always be predictable. An example using invoicing software shows that if the terms are due in thirty days, the tester may need to write a utility to calculate the system date plus thirty to do the compare. So-called "zero code" tools rarely are. As a result, test tools require code, but lack re-use. Testers end up cutting and pasting verification code. When that code needs to change, it might need to change in a half-dozen different places. The extent to which a record and playback tool supports reuse can greatly impact the long-term effort involved.

Interaction Points. Many test strategies require interacting with objects outside of the browser. This can include database setup, calling APIs directly, security, performance or load testing. Even something as simple as simulating multiple user interactions, to test for race conditions, may be outside the capability of simple Record/Playback tools. Requirements like these make the problem domain more complex, pushing the solution toward advanced or adaptive tools.

Silos. If the user interface is organized in clean elements, that can actually make testing much easier. For example, an ERP system might have an HR module, a manufacturing module, and a finance module, all of which can be built and tested separately.

Parallelism. Some teams speed up test runs by running several scenarios at the same time, also called "in parallel." The newest version of Selenium IDE does have a way to run in parallel in the cloud. In practice, coordinating these types of tests creates extra overhead, confusion, and timing issues.

Our Recommendations

Over the years, we've seen a series of practices work for Record and Playback. These are strategies that add value almost immediately, do not age easily, and will not require extensive training or dedicated assets.

If you want to chase the Record and Playback dragon, here's your playbook.

Quick Smoke Tests. Create a series of independent scenarios that test the core functionality of the application. No scenario should run in more than five minutes, the entire suite needs to take less than an hour. The smoke test demonstrates that the programmers *probably* haven't broken anything *big* functionality wise.

Automate Setup of Data. If you can't do setup through the backend, and it is required, you can create a Record and Playback scenario to do setup, then call it with different variables.

User Interface (UI) elements that are stable. Create scenarios around a stable UI—or at least stable enough. If the picture is changing but the workflow is still the same, then try to use static IDs and elements that are predictable. Avoid long connections to elements based on their locations in the web page (XPath or CSS).

First-pass to export into another system. If the Record and Playback system has an export feature, record a first pass then export it to a more robust test system for long-term work.

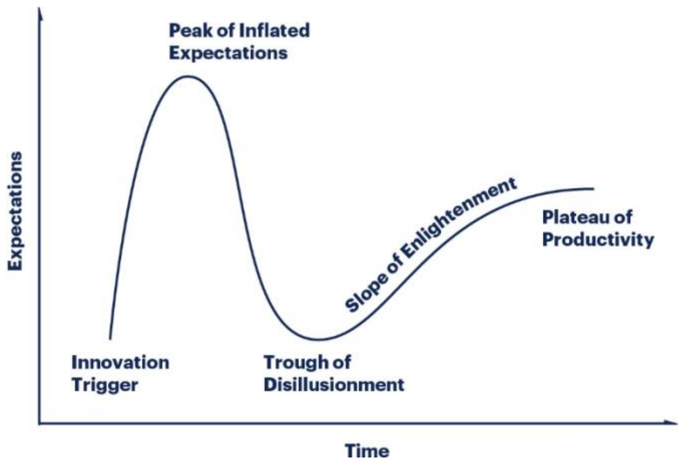
Jump out once you have to add code. When you start adding logic in a tool not really designed for variables or loops, then it might be time to move to something more robust. This is especially true if that code needs to be manually copied to other scenarios.

Track and limit scope. Forget ideas like 100% coverage. Instead, create a scale for how deep to go testing features. Publish the depth and feature map so the depth of your testing is well understood, along with the depth of your exposure. 100% is not a realistic goal, 20-40% might be...

Evaluate carefully with an eye toward ROI. Plan the investment of time and energy, compare that to the results in terms of defects found and actual time investment. Consider how that time would be spent without the tool - is it saving time? Enabling faster release? Catching bugs? Worst of all, could "passing" scenarios be creating a false sense of complacency?

Going Forward

The Gartner Hype Cycle[4] includes two early phases - the "peak of inflated expectations" followed by the "trough of disillusionment." Most Record and Playback customers go through these phases. A few manage to find a middle place, a realistic place called the "plateau of productivity." Customers who reach that point have been through a fair bit of work and a fair bit of pain. Our goal in this paper was to minimize the pain while accelerating the learning.



Still it brings up a question. Now what?

Record and Playback tools often involve a free trial; a few are just plain free as in Open Source. The analysis above should tell you if the tools are even worth trying at all, or if you need something more robust. If the tools are worth a try, then go ahead and try them. SeleniumIDE is a fine place to start. The advice above can keep a tester or small team engaged for weeks, long enough to learn if the tool will work.

After a month of real test work or three months of playing, make a plan to see if the tool can really meet all your needs. Think about longer-term issues like version control and complex scenarios. Consider realistic regression tests, verification points, tables that run the same test with different inputs and expect different results, along with the actual impact on the regression test window. Also consider the other test areas you may need to attack like backend, security, performance, etc. Go into the experiment understanding the test coverage you may aspire to, what can be realistically achieved using Record and Playback, and any other testing areas you may be leaving on the table.

If the bugs you have are a certain type of housefly, then the Record and Playback fly swatter could work just fine. If the tool you need is a sledge hammer … that should reveal itself rather quickly as well.

[1] https://www.capgemini.com/us-en/service/world-quality-report-2018-19-north-america/
[2] https://www.satisfice.com/blog/archives/5198
[3] https://www.stickyminds.com/blog/matt-heusser/software-testing-dead-not-yet
[4] https://www.gartner.com/en/research/methodologies/gartner-hype-cycle