Test Tools and Automation

Automation Selection Criteria:
Picking the Right Candidates
Robert Galen, Velocity Partners

Test Automation Tool Review
LogiGear Staff

Avoid Epic Fail
Get Professional Help
Michael Hackett & Joe Luthy

Cheap and Free Software Test Tools
Randall Rice
Testing tools - very important, very often overlooked, and very often where mistakes are made.

First, the most common mistake people make about tools is thinking tools are only about test automation! False. Automation tools are merely one type of testing tool. We will try to balance this issue between test automation tools and other test tools.

I heard a story about test automation, very recently in fact, of a company that paid a large licensing fee to a major tool vendor, and invested a lot of training and time to develop a phase 1 automation project. The intent was to scale up the initial phase into a large product suite automated regression tool. After just 6 months the project was dropped due to high test maintenance costs, over-idealized expectations and time demands. A whole lot of money, time, and tester goodwill went down the drain! I have heard too, too many of these stories with test tools, and specifically automation tools.

This story is not new. I heard nearly the same thing in 1994. This really highlights the need to address issues like designing code for testability, high reusability automation frameworks, lower maintenance test design, unrealistic tool vendor marketing, unrealistic staff expectation, etc.

Even after twenty years, too many teams still suffer from the problem of shelfware. Shelfware is slang for owning or licensing software that you don't use (e.g., it sits on a shelf). It was a story I first early in my software career. And it’s particularly problematic with test automation tools.

Clearly, do whatever it takes to avoid shelfware at all costs! Be part of the tool evaluation process; Demand training; Get direct vendor support; Make sure teams are aware of possible long ramp-up efforts; Make sure there is significant set-aside automation time, separate from testing time. And, always, treat automation development as its own development project!

A great suggestion these days is to get professional services help to jumpstart your automation program. There are experts who can build a framework specific to your application environment or work with you side by side and who can give 1-on-1 coaching to get the project started right or repair a broken automation effort. Yes, shelfware is a preventable problem.

Tools are meant to be the solution not the problem. But evaluation and selection is a unique process. We provide a few tips and there are many great articles on how to go through a tool evaluation process that will help you and your team get the process right! Don’t just rely on a tool comparison article. For starters there are some past LogiGear Magazine issues devoted to test tools. In 2012 (logigear.com/magazine/2012/09), we focused on Integrated Test Platforms such as ALM tool suites like TFS/Visual Studio, Jira, Rally, Rational/IBM, Thoughtworks. It’s worth a read. When carefully selected, carefully implemented, and you allocate the time and effort to maintaining your automated tests, automation tool are a huge bonus!

In this issue, Joe Luthy and I recommend that utilizing professionals can help you avoid epic testing fails; Robert Galen of Velocity Partners explains that picking the right automation tool can drive down test time and impress business folks; LogiGear staff present a set of criteria to pick the automation tool that’s right for you; Jim Holmes reviews Implemented Software Testing by Elfriede Dustin, Thom Garrett and Bernie Gaff; I provide some insight into the varieties of test execution tools and utilities for your toolbelt and Randall Rice of Rice Consulting provides a list of free and cheap software test tools that can help you across all phases of testing.

Tools help you do things. They can help you test better, faster, more informed, and help you isolate issues. Good luck! I hope you add some useful tools to your test effort.

Michael Hackett
Senior Vice President, LogiGear Corporation
Editor in Chief
In this Issue

4 IN THE NEWS

5 AVOID EPIC FAIL
GET PROFESSIONAL HELP

Michael Hackett and Joe Luthy, LogiGear Corporation

Bringing in experts can set you up for automation success

8 AUTOMATION SELECTION CRITERIA:
PICKING THE “RIGHT” CANDIDATES

Robert Galen, Velocity Partners

Identifying which tests to begin with when starting automation is key to driving testing cycle times down and coverage up

14 TESTING TOOLS: IT AIN’T ONLY ABOUT AUTOMATION

Michael Hackett—LogiGear

Test execution tools and utilities that can make your job easier

17 TEST AUTOMATION TOOL SHOPPING GUIDE

LogiGear Staff

A short-list of selection criteria and popular automation tools

21 CHEAP AND FREE SOFTWARE TEST TOOLS

Randall Rice, Rice Consulting

Budget-friendly planning and management tools, plus a few others

25 BOOK REVIEW: IMPLEMENTING AUTOMATED SOFTWARE TESTING

Jim Holmes

A review of ‘Implemented Software Testing’ by Elfriede Dustin, Thom Garrett and Bernie Gauf

26 TEST TOOLS AND AUTOMATION GLOSSARY

Some of the terms used when discussing test tools and automation

27 VIETNAM VIEW: HISTORY ARCHITECTURE OF SAIGON

“The Pearl of the Orient” boasts some of south-east Asia’s best examples of historic architecture
In the News

Video: Test Automation - 10 (sometimes painful) Lessons Learned

Test Automation - 10 (sometimes painful) Lessons Learned by Dominik Dary & Michael Palotas, eBay International: https://www.youtube.com/watch?v=tJ0O8p5PaiQ

eBay's European quality engineering team has broad experience implementing end-to-end test automation in different software development environments (agile, waterfall, collocated, outsourced, distributed). This presentation illustrates the key lessons learned from a technical and business perspective:

- Everyone knows what's automated;
- Instant feedback is essential;
- Flip the testing triangle;
- Investment into the test infrastructure;
- Maintainability is king and manual testing is still very important.

Global software testing services in BFSI sector to grow at 6.91% CAGR

Research and Markets analysts forecast the global software testing services market in BFSI sector to grow at a CAGR of 6.91 percent over the period 2012-2016.

One of the key factors contributing to this market growth is the need to reduce operational time and cost. The global software testing services market in BFSI sector has also been witnessing a shift from on-premise software testing to cloud-based software testing. However, lack of data control in outsourced services could pose a challenge to the growth of this market.

An analyst from the team said: "The global software testing services market in the BFSI sector is characterized by a lack of skilled and qualified personnel to meet the testing industry standards. Therefore, this trend has affected many businesses and their efforts to deploy quality production."

Boeing passes tests on way to launch craft

Aerospace giant Boeing Company, which has a large presence in Colorado Springs, CO, is one step closer to launching its Crew Space Transportation program after having successfully completed hardware design and software safety tests.

The company said Thursday in a news release that its Commercial Crew Program completed a Critical Design Review for its Launch Vehicle Adapter — a device that connects the CST-100 spacecraft to the Atlas V rocket on which it will ride — as well as the rocket’s emergency detection system has also successfully passed evaluation.

The two tests/reviews are part of NASA’s Commercial Crew Integrated Capability agreement with the aerospace company.

Boeing’s CST-100 systems will be put through another round of software testing and design reviews with an Integrated Critical Design Review coming this summer and is “on track to meet all 20 of its CCiCap milestones in 2014, according to the release.”
Test automation isn't easy when your testing gets beyond a few hundred test cases. Lots of brilliant testers and large organizations have, and continue to struggle with test automation, and not for lack of effort. Everyone understands the value of test automation, but few testing organizations realize its full potential.

A year-long IDT (Innovative Defense Technologies) automated software testing survey showed that automated software testing fails to deliver the expected value for the follow reasons:

- Lack of time: 37%
- Lack of expertise: 20%
- Lack of budget: 17%
- Tool incompatibility: 11%
- Other (mix of the above, etc.): 15%

*Source: Implementing Automated Software Testing by Elfriede Dustin, Thom Garrett and Bernie Gauf*

It's not surprising that lack of time and lack of expertise are cited as the top reasons test automation fails. Test teams often get a tool thrust on them by non-QA decision makers- from a purchasing group that has no idea about testing, or a CIO who was influenced by a tool salesman, their previous company or, something from a conference. These non-QA people do not understand the coding or scripting skill level needed by the team, or may not realize the tool is harder to use than expected, or worse, the tool does not easily work with the application or system under test. As a result, the automation program languishes and then fails. And it's the test team that looks bad.

If you’re starting a test automation program for the first time, or are charged with making testing faster and better a lot of the challenges will be the same. Test automation success begins at a high level. This is where the lack of (automation) expertise can cripple a project before it gets started.

Many people have unrealistic expectations about how easy or how quickly a program can ramp up and fail to take into account the impact on test schedules. Commitment to time and training and cooperation with developers must come from the highest levels in the organization. Having an outside expert share experiences and educate managers and directors about the complexities and realities of automation programs can set the right expectations and set you up for success by providing:

- Strategic consulting—assess your current practices and create the strategy for your automation program
- Automation Consulting—develop and prioritize the automation effort and supporting activity specific to your needs
- Training—instruction in tools, processes and methods
- Coaching—hands-on guidance that helps teams become proficient faster

What follows is where organizations can benefit from bringing in a qualified partner to help with their test automation.
Creating the Automation Strategy

Test automation is software development. It needs to be treated like its own project running parallel to production code development, with its own schedule, resourcing and project management. You need commitment and support from the top of the organization for a significant automation suite build-up.

Bringing in someone with prior success in automation implementation—a consultant—to assess your current practices and develop a custom-tailored plan will avoid making costly mistakes. A well-developed strategic plan will define the testing goal and outline the roadmap to achieve the goal. It will also establish an understanding of testing/quality assurance across the organization, and prescribe staff development to assure the team will have the necessary skills for test automation.

The strategy will guide other aspects of the automation effort, including tool selection. Typically tools are evaluated in this order: cost, ease of use, and features/functionality. There are low-cost easy to use tools—to a point—but getting the tool to do what is needed may require coding/scripting skills the team may not have. Also support is important. Support runs the gamut, from non-existent with open source tools to an included number of hours with commercial tools. The amount of support required will depend on the skills of the team, and it’s good to have when you run into a wall. Also, over-time, you application is likely to undergo quite a bit of change and having a tool that can be customized may be worth paying extra for. These considerations are generally not obvious to teams with little or no prior automation experience and this is where having a well-defined plan will avoid a costly long-term mistake.

Test Automation

This is the stage where the strategy gets implemented, but before just diving in and starting test automation, there’s a lot of work that needs to be done.

You’ll want to build infrastructure that includes naming conventions, standard or common coding practices, source control, error handling, managing data and reporting.

Teams should anticipate spending time to clean up existing tests before automating them. There are significant differences between tests written for manual execution and tests written for automation. Programmers typically don’t pay attention to “designing for testability”. They’re either unaware of how code changes impact automation or don’t effectively communicate changes. The result is that automation is negatively impacted. Failing to address test design issues will result in a large mass of test code that isn’t effective and requires a lot of time just to maintain. When teams understand good test design for automation it will become apparent there will be tests that should be thrown out and replaced with better written tests.

You will also need to develop and prioritize the automation. Taking 100 manual test cases and trying to create 100 automated tests will always be a bad idea. Not all tests are candidates for automation. If the application under test is not stable, it will impact which and how many tests can be automated. Then too, there will be manual tests that can and should be automated, and a lot that shouldn’t be.

Consultants with experience in successful automation implementations can help with all of these areas and enlisting their help at the beginning will get you started on the right track. Once you’re up and running it’s easy to build on success.

Training and Coaching

Training is the essential first step to gaining test automation expertise. There are a lot of good training resources. They range from free, on-demand recordings to generic-instructor led courses, to customized training. We’re seeing more and more companies opt for customized training as it’s not easy to translate generic concepts to specific applications.

The most important training for any team doing test automation, but especially those in Agile and Continuous Integration (CI) environments and teams doing large, high-volume automation, is learning test design. Up-front test design will determine how efficient tests are to run and how easy they are to maintain. Getting this right is what makes it possible to scale automation.

No matter where you are in your automation process,
learning low-maintenance test design will enable you to get the most from automation. Action Based Testing (ABT) is a method of test design that lets you do this. It allows you to create common functions and function libraries that make for efficient test design and simplified maintenance. The limitation to training is that it only provides information. You have to turn the knowledge you receive into application, which can be slow, and you’ll probably make mistakes along the way. Getting hands-on instruction will shorten the learning curve dramatically.

Hands-on coaches are the equivalent of personal trainers for testing. Coaches are able to work 1-on-1, and with a small group on your actual testing projects. They can guide you in the test design and the correct approach specific to the application or system under test, and even sit at a keyboard and walk you through a process. Coaches not only help teams become proficient faster, they are also able to jump in and help resolve complex issues so the project keeps moving as expected.

Outsourcing Sense

Automation will not immediately reduce the test effort. In reality the effort typically increases when automation is undertaken. Building the testing infrastructure takes time and utilizes a lot of resources. Designing and creating new tests will also take time. The additional work load can overextend teams already taxed to keep up with current testing demands. So it’s no surprise that lack of time is the number one reason test automation fails.

Outsourcing some or all of the initial work makes a lot of sense. It will be done by people who have done it before. This saves time and avoids mistakes and gets everything integrated and provides you with properly designed tests from the start. Farming the work out can let you keep up with existing testing, and then have a smooth transition to the new program. When you factor in everything, most of the time this will provide a better ROI than going the in-house route.

Summary

Test automation isn’t easy, but it can be made a lot easier. Expert consultants can assess the structure, staffing, the corporate quality goals, and provide exactly what you need for an effective test automation program. Going this route is not only smart, it will save time, produce long-term benefits, and most of all avoid epic fail.
So there you are. You’ve done a little research and made the business case to upper management regarding test automation and they bit on the proposal. Surprisingly, they supported you all the way and are extremely excited about how much faster testing can really move, or upper management comes to you with an edict to start automating your testing. You’ve got to improve testing cycle and turnaround times and do it with fewer resources. They clearly believe automation is the only way to achieve these goals. You’re given a nearly blank check to quickly bring automation into play for the next major release—3 months away…

In either case, you’re in the enviable position to be poised to begin developing test automation.

Automation can provide a tremendous boost to your team in technical challenge & contribution while wowing the business folks in driving testing cycle times down and coverage up. Frankly though, this can also be an intimidating time—especially if this is your first time trying to sort out where and how to begin, which is exactly the focus of this article.

This is the second installment in a series of articles targeted towards improving your management and setup of testing automation. The first was focused towards the Automation SDLC and in the next one we’ll explore developing a solid Business Case. One of the basic assumptions I’m making for this article is that you’ve been creating test cases and manually testing as an organization for a while. That is, you’ve built up some sort of repository of manual test cases that are potential automation “candidates”. Given that you can’t automate everything at once, the question of where to start and how to properly orchestrate your efforts over time becomes a challenge.

I’m also assuming that you do not have infinite resources nor time to produce visible results. That is, you have other testing responsibilities besides the automation, for example testing and releasing your products. So prioritization and establishing a work balance becomes a challenge as well.

We’re going to examine three key areas to help you craft a strategy to meet these challenges:

1) First I’ll explore a few common anti-patterns that impede good test case selection.
2) Then we examine a solid set of good practice patterns for test case selection.
3) Finally, we’ll wrap up with prioritization and criteria adjustment factors so that you can truly be nimble over time.

**Automation Selection Criteria – Picking the “Right” Candidates**

Identifying which tests to begin with when starting automation is key to driving testing cycle times down and coverage up

By Robert Galen
Common Anti-Patterns for “Starting” Automation

Anti-patterns have been applied to software design, coding, configuration management, and just about any activity within software projects. I think exposing what not to do is useful in setting the stage for some of the recommendations I’ll be making later on, so here are a few general antipatterns for selecting good test candidates for automation development.

We Don’t Need No Stinkin’ Criteria

It simply amazes me how many groups simply dive into automation development without a strategy surrounding what test cases to automate. They simply start automating somewhere within their pool of test cases, often picking an arbitrary starting point such as first character in the tests name, and then moving serially forward from that point. Another part of the No Criteria antipattern is never reevaluating your lack of criteria as you make forward progress.

A Good Start – But What’s Next

I also see teams who start well, for example, picking a set of “low hanging fruit” automation candidates that make sense to initiate the automation program. Typically the set is small, intended to get the effort going quickly and to realize some short term successes.

However, after the team accomplishes their initial plans, they fall back into a no criteria, select anything you want mentality towards automation.

By George, Let’s Do It All

Another frequent anti-pattern is the view that all test cases need to be automated. This creates churn because the team is frenetically trying to do it all. Frequently working in parallel with a mainline software project and struggling to automate test cases on a moving functional target. It also drives the mentality that all test cases need to be automated independent of the level of effort or returned value associated with that endeavor. Which is simply not business savvy nor realistic.

In Tools We Trust

This anti-pattern is focused towards the myriad of automation tools available. Often they can lull the team into this false sense of security that the tool will take care of the details surrounding automation—thus removing the need to understand the tools and technologies being used. It also masks the teams from understanding the strengths and weaknesses of each tool as it relates to different technologies. And trust me every tool has “problems” that need to be considered.

Make A List, Then Stick To It!

In this final anti-pattern, teams do everything well at the beginning. They take an inventory of their test cases and pull together a thoughtful ordering that starts building good automation and contributes positively to their projects. However, they become stuck in their initial list and never adjust it for discoveries and changes as they progress. As time moves on, their original approach becomes more and more irrelevant.

Clearly these anti-patterns set the stage for the remainder of the article—changing the focus towards what to do regarding selecting the best candidates for automation and developing your overall implementation strategy.

Selection Criteria – Patterns

Taking a key from the anti-patterns just mentioned, there are some general selection patterns that I normally use to govern my thinking around selecting good test cases. We’ll examine these practices with the view towards establishing them within your own automation efforts.

Low Hanging Fruit

As I alluded to in the anti-patterns section, this pattern is focused on generating momentum and not in picking the best, most impact producing automation targets. It usually happens in the beginning of an automation effort or when you’ve received a major new version of your automation tools. In any event, low hanging fruit test cases are just that – they’re usually
small, localized to well understood and stable functional components and generally quick to implement.

I’ll usually focus new automation developers on these cases for learning curve and to measure their capabilities before assigning them more challenging work. A final important part of the pattern is making your successes visible—both internally within your testing team and across your organization. So showing early success off a bit is an important part of the pattern.

Fleshing Out The Infrastructure

All good automation efforts establish an infrastructural layer that wraps their tools in order to support efficient automation development. Usually this is focused towards implementing automation development templates, standards, naming conventions and other guidelines that enforce consistent practices and procedures for development. It often also involves writing some wrapper code that programatically reinforces the consistency and serves to reduce startup time and learning curve.

These practices need to be verified and debugged, so selecting good candidates to flesh out and verify the infrastructure becomes important. Usually these candidates are distributed along the architectural boundaries of the Application Under Test (AUT)—so that the infrastructure is verified to support all aspects of the application environment.

Minimizing Rework

There are clearly two factors that can drastically effect your efforts and influence rework—requirement stability and application stability. While you can’t always control either, you can choose when to automate and thus mitigate impacts from the two. In my experience, the easier to control is requirement stability. In this case, you simply want to wait until there is sign-off or agreement on requirements. Another indicator here is that the development team is beginning construction. Normally this is a good indication that if the requirements are stable enough for development they might be stable enough for automation.

Application stability is a tougher factor to control. I usually like to trigger automation development until after we’ve at least had a chance to exercise applications features manually. If manual execution is impossible, for example with an API, than any demonstrated testing and positive results will serve as a trigger for automation development.

Of course a good strategy in both of these cases is to defer your automation construction until after the specific application release is deployed. In this case, you take both aspects virtually out of play and drastically reduce rework risk.

Driving with Value

At a fundamental level, automation value is driven by the coverage it provides contrasted against the time it takes to automate it versus the time saved by the automation. The potential lifetime of the automation then is a strong factor in determining its value. I see many teams who write automation for very transient application features. They’ll run it only a few times and then the application morphs in another direction. They’re stuck either taking the maintenance hit to change the automation or retiring it. In both cases, the value of that particular piece of automation was impacted. Value is also determined by the customer value associated with the specific feature. How central is it to their business needs and frequency of use?

You should think of the return potential for every automation decision. More times than not, you’re looking to automation capabilities that are stable—so that the automation has a high lifetime before maintenance is required. These features also need to be of high cus-
customer value—leading towards heavy interest in repeated testing cycles.

**Planning – Consider Complexity & Packaging Tempo**

One of the more important factors to consider is simply how hard things are to implement. I wish more product development teams would consider this when developing their products. If you have 100 test cases that you want to automate and they’re all exceedingly complex, then you’re somewhat stacking the deck against yourself for delivery.

What I prefer to do is create a more normally distributed set of test cases, say 25% very complex, 50% moderately complex, and 25% relatively straightforward, when developing an automation release package. I’ll also create a goal for the package that represents one of the overall patterns in this section. For example, after my first automation Low Hanging Fruit package development I’ll often go for a cycle time or speed focused goal of **Project Impact – Consider Speed** release.

This way I immediately start leveraging automation impact on the project. Regardless of the approach, you should develop a packaging strategy that uses goals and short iterations to create a tempo for your automation release.

You also never want to underestimate hand-off considerations as the story in Sidebar #2 illustrates.

### Project Impact – Consider Speed

As testers we always want to maximize the impact we have on our projects, which usually focused on risk mitigation. Automation also has the capacity to drastically impact testing speed—so examining how quickly a package of automation can be pulled together for product execution becomes a very compelling view. In this case, identifying test cases that take the longest time to execute manually and automating them can have a drastic effect on testing cycle time.

Of course you need to consider all aspects of time when picking these candidates, including test setup time, data preparation time, execution time, and results analysis time. Clearly you don’t want to automate the smallest or fastest tests first if you want to have a high impact on cycle times within your project.

**Project Impact – Consider Risk**

I’ve found that one of the most powerful applications of automation is as a risk mitigation factor within your projects. From that perspective, you may want to choose candidates that align with some of the more challenging areas of the application. Areas where you know the development team will struggle and where you can make a difference. This may mean that you take less on—in order to have a larger impact on the products’ maturity stabilization.

One key practice to consider here is the Pareto Principle or 80:20 rule. Basically it states that 80% of the bugs (or risk) will be driven by 20% of the overall application. If you can identify and isolate these 20% areas and apply your automation efforts towards them, you’ll be well on your way towards effectively mitigating project risk. And your developers and project managers will love you for it!

**Project Impact – Consider Maintenance Effort**

One of the truly hidden properties behind developing test automation is that it is indeed a software project.

Given that, it also succumbs to maintenance issues both in the short term, as you try and automate a volatile application, and long term as the application evolves and matures. I’ve found that nearly 20-30% of my automation development time is spent in handling traditional maintenance level activities. And this was, at least from my point of view, using well architected automation solutions so it can potentially get much worse than this.

One way to handle maintenance work is to steal a technique from the development lifecycle and have periodic maintenance releases of automation. I’ll accrue maintenance work for these sorts of releases and time them to coincide with possible slack time within my projects. This also allows me to better understand, isolate, measure and communicate the costs as well.
Clearly you don’t need to consider all of these patterns in every selection decision. And they all have different weights depending upon your specific context. The primary goal is to create a breadth to your analysis and consideration that truly targets test case selection towards making a visible and fundamental impact for each of your projects. It should clearly be seen as a differentiator when the team discusses project success factors. Sidebar #3 serves as a decisionmaking checklist for establishing this broader view.

Changing Your Selection Criteria

One of the challenges associated with selection criteria is that they don’t remain constant, or at least they shouldn’t. All sorts of conditions change with your automation scheme and projects that should cause you to pause, reflect, and adjust your criteria. Here I’ll simply list a few of the primary conditions that usually cause me to reflect and potentially change selection criteria.

While I’m certain there are others, these will certainly illustrate the point for criteria adjustment.

1) Changes in Skill Set – Either positive or negative can certainly change your criteria. If your capabilities increase, you can and should take on more with each project and automation iteration. If they decrease, then you must take on less, which places more importance on what you select.

There are many factors that can influence this, attrition, outsourcing, changing project priorities, and changing tool sets come to mind as frequent drivers. The real point is to closely monitor your performance capabilities and adjust as they do.

2) Changes in Application Technologies – Quite often developers adopt new technologies that clearly disrupt ongoing automation efforts. If you’re lucky, you get an early warning so you can experiment to ascertain the impact prior to product release. However, most of the time you’ll become aware of automation implications at the point of implementation.

If I anticipate technology changes that will impact your automation tools or strategies, I often plan for a small, impact evaluation iteration. Within it I look to fully understand the impact of the changes, evaluate the impact to my tools and infrastructure, and carefully estimate the change impact to my existing set of automation. Once I have a good feel for the full impact, I’ll adjust my plans and criteria as required.

3) Changes in Team Balance – I often like to think in terms of “producers” (developers) and “consumers” (testers) when planning my testing and automation efforts. That’s why developer to tester ratios are so important for capacity and workflow planning. If you get a drastic change in your ratio, then it will certainly have an effect on your automation capacity.

Again, as in the case for Changing Skill Set, I want to reevaluate my prioritization to ensure that I’m selecting the more relevant candidates.

If you don’t have a dedicated automation team this becomes an ongoing challenge as you multi-task and reallocate automation resources towards other project-driven testing activity.

4) Ongoing Setup Time – The time to setup and teardown your automation run-time environment can be a significant factor in how you select automation candidates. This becomes particularly important in some database applications that require vast amounts of time sensitive data in order to physically setup and run the automation. Or any automation candidates that simply requires exorbitant time to establish an initial state.

I usually prefer to attack time sensitive test candidates quite early in my automation efforts as a risk mitigation strategy. Often, I don’t understand the full implications of the setup nor the impact it will have on my scheduling other, much shorter, automation.

5) Evolution of Maintenance Costs – I’ve found that automation, depending upon how it’s architected, can be particularly sensitive to maintenance costs. Part of this burden is naturally driven when you try and implement too early—before interfaces or functionally within the application have stabilized. Another aspect is internally driven, for example when you make infrastructural changes or upgrade versions of your automation tool-set.

One reality of maintenance costs is that you need to pay attention to detect them and address them immediately. However, there is good news in that they are frequently cyclical in nature so that once they are addressed you can fallback to your normal selection scheme.

6) Evolution of Development Methodology – Clearly if your organizations SDLC or methods change it will probably impact your automation efforts—see Sidebar #1 for an overview of the factors. The most drastic
case of this resides in the Agile Methodologies. In these cases, you’ll want to shift your automation focus towards each of the development iterations, while helping to automate the unit and acceptance testing activity within the context of each.

In addition, you’ll want to target automating candidates as they “exit” the iteration. However, there can be tremendous volatility across the feature and interface set, so you’ll want to carefully consider the stability of each feature or candidate before implementation.

7) Changing Business Conditions – The can take two primary perspectives, budget & investment or changing business value and requirements. Often it has the same effect as changing skill sets or technology in that it causes you to change your view to automation selection either positively or negatively.

8) Retirement Consideration – Just as software incurs technical debt over time, so do your automation efforts as test cases can lose their relevancy and connection to the current application. Or what was a high priority candidate is now something that needs to be executed infrequently or not at all. As part of my changing selection criteria planning, I usually drive reevaluation of automation from a retirement perspective. Retire can imply, just that, a removal of an automated test case from the execution environment or a reduction in the frequency of execution. In either case, you should formally note the change and, if removing it, move it to a retired automation archive.

My regular interval for reevaluating my automation environment and selection criteria is either on a quarterly basis or before starting a significant new project. That way I cyclically evaluate the landscape and consider any important changes that might influence an adjustment of my overall selection criteria.

Wrap-up

The overall focus of this article was to influence your planning and strategies surrounding the selection of automation candidates. A few of my early reviewers reacted badly to all of the factors—concerned that considering all of them would bog down the automation process with overbearing planning and simply take too long.

However, my experience is just the opposite. I’ve found that when I develop a selection criteria that maps to my overall automation strategy that I actually develop more automation. It also has greater impact on the business and projects, while increasing the visibility of my testing team in the overall product development lifecycle.

One final point, don’t ever think that your automation efforts are done. It’s an ongoing challenge and I think you need to stay on top of new tools, techniques, maintenance, products, etc. Viewing it as a short term exercise with a succinct end point is certainly not the right mental model.

About Robert

Agile Evangelist & Coach – Velocity Partners

Bob Galen is an Agile Methodologist, Practitioner & Coach based in Cary, NC. In this role he helps guide companies and teams in their pragmatic adoption and organizational shift towards Scrum and other agile methodologies and practices. He is a Principal Agile Evangelist at Velocity Partners, a leading agile nearshore development partner. He is also President and Agile Trainer & Coach at RGCG.

Bob regularly speaks at international conferences and professional groups on topics related to software development, project management, software testing and team leadership. He is a Certified Scrum Coach (CSC), Certified Scrum Product Owner (CSPO), and an active member of the Agile & Scrum Alliances.

He’s published two agile related books: Scrum Product Ownership, in 2009 – 2’nd Edition in 2013 and Agile Reflections in 2012. He’s also a prolific writer & blogger (at - www.rgalen.com ) and podcaster (at www.meta-cast.com )

Bob may be reached directly at: bob@rgalen.com or bgalen@velocitypartners.net or network via: http://www.linkedin.com/in/bobgalen
Testing Tools: It ain’t Only About Automation!

Test execution tools and utilities that can make your job easier

By Michael Hackett, LogiGear

My first exposure to the necessity for testers to have an array of tools was from the groundbreaking article “Scripts on my Toolbelt” by Danny Faught. Danny laid out the ideal approach to any testing job, and it got me thinking “How can I leverage tools to help me?” I needed a group of tools to get my tasks done. I needed to have a full toolbelt! It makes a lot of sense. A carpenter can’t do everything with a saw, he needs dozens of tools to get the job done.

Typically when people think about testing tools, they immediately think about test automation tools. Clearly, test automation is an important task, but it’s only one task. Most test automation is traditional black box test automation. What that means is that automation lets you run black box tests faster than you can run them manually. I’m not against test automation or saying test automation is bad. What I am saying is we need more than just test automation tools. After all, there’s a lot more to software testing tools than black box test automation.

What do Software Testing Tools Do?

I want tools that accomplish these 3 tasks:

1) Do what I do faster, only faster.

This is where test automation comes in, it merely does what you could do yourself hopefully (if you’re investing in a test automation program) a lot faster and a lot more!

An example of this is using a tool for fault injection. Fault injection, or forced error testing (http://en.wikipedia.org/wiki/Fault_injection) is an essential test task in every testing project. This can be done so much faster by a tool than any human. You can see a dozen tools listed on the Wikipedia page for fault injection.

Working with HTML? If you know W3C standards, you can read HTML code and verify code compliance, but even better, or if you don’t know the standards, you could feed the code into the W3C Validator, and get the analysis done for you in seconds. http://validator.w3.org/

2) Give me information I typically would not see.

Memory meters and memory leaks detectors give me great information. Memwatch is a memory leak detection tool I often use, and there are many other tools like it. http://www.linkdata.se/sourcecode/memwatch/

3) Do tasks I cannot.

Parsing a log is a good example. Many logs are mammoth in size. Sorting thru, finding only 1 type record, tailing only the final x number of lines... these tasks are impractical for a human, but utilities can do the sorting, filtering and formatting and then export the output for attachment to a bug report in seconds. http://www.log-expert.de/ and Microsoft’s LogParser 2.2 are tools I often use.
Scope of Testing Tools

Test tool discussions today are very, very different from tool discussions just a few years ago. There are test management and measurement tools, which I won’t even attempt to cover. Even the scope of tools available for use during test execution is vast. There are thousands. I’ve also had to leave out security, performance, load, monitoring, all kinds of developer tools from unit testing harnesses to static and dynamic code coverage and analysis tools... There are just too many different types of tools to do justice to in a single article. My goal is to open your eyes to a broader array of QA testing, even black or gray box testing tools.

The mobile explosion has led to a surge in mobile platform tools and cross-platform tools that are often differentiated by how many testing tools they include. And, there are new testing frontiers many teams did not previously face, like testing download speed, connectivity speed, the number of server requests, amount of data downloads etc.. All of these things impacts what we as mobile users are changed, and they need to be tested. We need tools that will give us that type of information.

ALM tools have mainly been used for task management, project management and measurement, yet a big selling point for administrative tool suites are the test tools integrations. Every ALM suite now has integration of a unit testing harness and continuous integration (CI) tool, and some also integrate with GUI or UI driven test tools, code coverage analysis tools, capture/reply tools, and various others.

There have always been some variety of testing tools packaged with IDEs (Integrated Development Environments) but they almost universally focused on tools for developers, and were very limited in scope. This has changed. Take Visual Studio/TFS for example. In addition to the breakthrough Coded UI tool, it now has an Action recorder, intelliTrace, event log, system info, network emulation, test impact and built-in video recorder, all of which can provide great benefits to testers. Coded UI can be extended from a white-box unit testing tool to a black-box test automation tool by using the TestArchitect for Visual Studio (TA4VS) plug-in, a action-driven test development extension for creating and maintaining, Coded UI tests without the hassle of coding.

More importantly these days, there is a greater focus on testing tools instead of only developer tools. I would like to think this is due to greater attention to releasing quality product, but may merely by a result of the need for speedy delivery. Higher demands for speed and quality, from faster development practices, faster distribution and deployment and particularly with mobile development, the speed of delivery, instant feedback and cross-platform nature of mobile apps have all made test tools an integral part of any testing practice rather than nice-to-have.

Many people look exclusively for free or open source tools. This is a big mistake. The justification for free or nothing is often that their company will not pay for testing tools. That does not say very much for that company’s commitment to quality. While there are some great free and open source tools, there are some that are well worth the cost. My arsenal of tools is a mix of the best tools I can get, free or not. BTW, tool and utility are often used interchangeably, but there are some technical distinctions.

Utility software is system software designed to help analyze, configure, optimize or maintain a computer. A single piece of utility software is usually called a utility or tool.

*Tool* is sometimes reserved for items that are more robust, such as having a GUI or more functionality.
Choosing Tools

Many people have written about the steps to evaluate and procure tools that are right for you, including integration with other tools sets, cost, usability, etc. Cost is often the easy part. Of course, everyone wants full cross-platform support but that’s a big ask. My focus is only to point out key points of selecting the tools to solve your problems.

First, define your tool needs,

- What problem are you trying to solve?
- What tasks take too much time?
- What information do you not have?

After finding a tool to solve your testing problem, you need to evaluate, among other things:

- Building expertise and training
- Installation and maintenance
- The amount of programming skill needed

It is important to remember a great source of testing tools—your development team! They will be familiar with the exact needs, they can perfectly fit the tool to the platform, and more. The biggest problem with your dev team creating tools for testing is lack of time. Most dev teams I know rarely have time to build tools for their test team. But it is worth asking!

Random Tools for Your Toolbelt

I have a prejudice toward testing tools that give me information I can’t get through black box tests. I like tools that give me information at the system level. I like to see what APIs are loaded. I like to find out how much memory is being used when I do certain tasks.

One of my favorite tools is Dependancy walker. (http://www.dependencywalker.com/) It is a free utility that builds a hierarchical tree diagram of all dependent modules. For each module found, it lists all the functions that are exported by that module, and which of those functions are actually being called by other modules. To test SQL Injection for databases, definitely find tools! A couple of others I like are Mozilla’s SQL inject me add-in, https://addons.mozilla.org/en-US/firefox/addon/sql-inject-me/ and SQL map http://sqlmap.org/. SQL Map has many utilities including penetration testing.

Tips for Finding Tools

Don’t forget logs. You may or may not be aware of all the logs readily available for your reference. From keystrokes to databases and everything in between, every action is logged somewhere.

What platforms or ALM suites are programmers developing your product on? Perl, c#, JavaScript? HTML? SQL? Objective C? every development IDE has lots of tools. Check them out and ask the developers.

See what your team already has. Many times with staff turnover, tools that are a perfect fit get “lost” or stop being used. Also ask other testing teams in your company. For some strange reason, test teams sometimes live in a silo with little contact. They may have tools you can share and visa-versa.

Use in Bug Analysis

Bugs are very often obvious but just as often the causes of bugs can be mysterious. A special use of tools is defect isolation, reproduction and reporting. Many of the tools I’ve listed can assist you in dealing with bugs and reporting excellent bugs in excellent bug reports. Along with recorders, trace analyzers, logs, DependancyWalker, and screenshots bugs do not have to be so mysterious.

Summary

My recommendations to you:

- Look around to find sites that list tools specific to your development platform
- Look for tools built into your ALM or IDE.
- Ask your Devs.
- And, Go hunting— you may be surprised at what you find.

BTW, Monkey has become a very popular name for tools in the mobile arena. A monkey is a cute name for a utility or tool that does a boring, mundane task for you.

Go hunt for some monkeys!

About Michael

Michael Hackett co-founded LogiGear in 1994 and leads the company’s LogiGear University training operations division, setting the standard in software testing education programs for many of the world’s leading software development organizations.

Michael is co-author of the popular Testing Applications on the Web (Wiley, 2nd ed. 2003), and has helped many clients produce, test and deploy applications ranging from business productivity to educational multimedia across multiple platforms and multiple language editions.
Test Automation Tool Shopping Guide

A short-list of selection criteria and popular automation tools

By LogiGear Staff

There are a lot of test automation tools available in the market, from heavy-duty enterprise level tools to quick and dirty playback-and-record tools for browser testing. For anyone just starting their research we’ve put together a short list of requirements and tools to consider.

Requirements

Selecting the right tool starts by identifying what you need to achieve. If you only need to automate web browser testing, your requirements will be different than if you’re testing a custom built banking application.

Team Skills

Manual test design and automated test design are vastly different. There is no use investing in a tool the team won’t be able to use effectively. This will also be a factor in determining the level of support required from the tool vendor.

Programming Resources

Some tools require the skills of knowledgeable programmers to implement. Easy to implement tools lessen the need for programming support but may sacrifice features and functionality, so the tradeoff must be weighed carefully.

If the tool supports the code your programmers are using, it’s much easier (and simpler) to get programming support for difficult automation tasks. It also allows for the potential of reusing code from unit tests, which saves time and cost of creating automated tests.

Controls and Objects

Operating systems provide dozens of various controls that can be used in tested applications like: edit boxes, list views, check boxes, menus, buttons and others. The tools you consider should include features that simplify working with controls used in your application.

Operating System Support

The tool should support common operating systems and be easy to update for the latest versions. Commercial tools provide maintenance for this, whereas open-source tools not so much.

Language Support

If you need to test applications against multiple languages, you don’t want to have to create tests in every language. The tools you consider should have multiple language capability.

Customization Support

Specialized and complex applications will likely require customization. If this is the case you will want to focus on tools that can accommodate your needs and even offer support services that can do the customization for you.

Ease of Use

With training and experience most tools are easy to use. Record & playback tools satisfy ease of use, but not a lot else. Instead ease of use being a primary selection criteria, it is better to use it as a comparison between the top choices.
Distributed Testing

If you’re doing large-scale testing you will want to have a tool that allows tests to be executed over multiple machines that can be controlled from a single location.

The Tool Short List

Here’s a list of a subset of test automation tools available. Each tool has pluses and minuses. The best tool for any specific application will need to be determined by conducting a thorough evaluation.

HP® QTP (Quick Test Professional)

HP is the market share leader for software test automation tools. QTP works with most Windows and web-based applications. The latest iteration of the tool is UFT (Unified Functional Testing), which includes QTP for front-end web-based testing, and ST (Service Testing) for back-end Web Services-based testing, as well as mobile testing capability.

Pros

QTP has many included web and windows target automation platforms. There are also numerous integrations to 3rd-party add-ons like Flex, QT, Delphi XE2, Stingray 11.1, Web Dynprop ABAP for NW 7.31, etc., that can be purchased for specific application needs.

Novice and non-technical testers can create basic, low-level automation scripts with the record and playback feature.

QTP is integrated with HP’s Test Management tool, Quality Center (QC), to map QTP-created test cases that can then be executed in QC.

Cons

QTP test cases are heavily reliant on scripting. Test cases created with the record and playback functionality will have to be re-recorded whenever the application under test changes.

QTP only supports VBScript (a readable English-like language).

As test cases scalability grows, test case maintainability becomes a larger part of the test effort.

QTP is a heavy weight on the cost side for both licensing and computing resources. Licensing cost varies depending on the number and type of licenses. Some versions of QTP require extra-cost add ins/plugins to work with DotNet components.

Microsoft® Visual Studio® Coded UI

Visual Studio 2010, included Coded UI, as an integrated approach to get testers and developers on a unified platform. Now on the third rendition, Visual Studio 2013, Coded UI, and Team Foundation Server (TFS) with Microsoft Test Manager (MTM) form the foundation for UI test automation is a software development and application lifecycle management (ALM) environment.

Pros

Microsoft’s Coded UI integrates across Microsoft’s with Team Foundation Server (TFS) and Microsoft Test Manager (MTM).

User actions can be recorded while running a test case using MTM (Microsoft Test Manager). The automated recorded actions can be repeated any number of times after a change is made in a build.

In Visual Studio 2013, Coded UI tests support a variety of technologies, including XAML-based Windows Store apps, IE 11 and SharePoint 2013, in addition to previously supported technologies. It provides logging support for cross-browser tests.

Cons

Test creation is time consuming and Coded UI generates a large amount of C#-only code, even for small UI operations, complicating test maintenance.

Coded UI uses almost all the property-value pairs to identify and locate the UI controls, in reference to the UI MAP, which can affect the performance of the script.

Even if a set of code has been generated for a particular action, Coded UI will regenerate exactly the same code when the test actions are repeated, without leveraging the value of code reusability.
TestArchitect™ for Visual Studio®

This version of TestArchitect is an extension for Visual Studio (2012/2013), providing built-in functionality. The tool uses Action Based Testing (ABT), an advanced keyword method that provides a number of key advantages over Coded UI automation.

Pros
Coding and scripting is replaced with reusable actions for test creation. Built-in, preprogrammed actions simplify test creation.

Tests have high reusability, reducing the amount of time needed to create and execute tests. Test cases can be managed in TFS and executed in MTM.

TestArchitect uses the construct of test modules to organize test cases. This functionality makes it possible to abstract tests to the business-logic level, with objectives or user stories for each test. Tests are aggregated to the same or similar scope, making test flow organization a key focus of test development.

Cons
TestArchitect for Visual Studio only supports platforms that are supported by Coded UI. There is no iOS or Android mobile support.

TestArchitect™

TestArchitect is the enterprise test automation tool that integrates a test design methodology (ABT), test automation, and test automation life-cycle management (TLM). It addresses the needs of large-scale test automation for complex applications by focusing on test and automation maintainability.

Pros
Actions are used in place of scripts or programming for test creation. A centralized repository allows remote teams to leverage reusable test assets.

Allows distributed test execution on multiple physical or virtual machines. Includes a variations feature that allows a testing against multiple languages with a single test.

A programmable harness is provided to support customization for proprietary or third-party platforms and controls.

Cons
TestArchitect is not ideal for small-scale test creation. The systematic approach to test design and test flow is best suited for high-volume testing.

A steeper initial learning curve is required to take advantage of the ABT test methodology, than recording and scripting.

SmartBear TestComplete®

TestComplete is targeted at small to midsize test teams. It includes keyword testing to make the tool usable for a wider audience of testers. Smartbear also offers a SaaS version of the tool.

Pros
TestComplete’s record-replay feature makes it simple to record and automate basic tests. The tool works well for small to mid-size teams looking to automate a portion of their software testing.

The tool has an attractive price and there are a number of add-ons that can be purchased to extend basic capability.

Cons
Record and play-back tests require a significant amount of maintenance when the application under test is changing rapidly.

Hand-scripting test cases are an alternative to record and playback, but hand-scripting can be difficult or for novice or non-technical testers.

Micro Focus SilkTest®

SilkTest is used for automating functional tests for enterprise-level software applications. SilkTest’s automation capabilities make it a consideration for regression, cross-platform and localization testing across a broad set of application technologies.

Pros
SilkTest supports a range of application platforms, including AJAX/Flex/Web 2.0, Java, .NET, client/server system. It also provides cross-browser support, without test script modification.
The embedded VB .Net scripting language enables test customization and automation through a visual-based or editor-based editing environment.

**Cons**
SilkTest is limited to supporting only Windows applications and only through Internet Explorer and Mozilla browsers on Windows.

SilkTest use 4Test scripting language, an object oriented language similar to C++, which requires a programming-centric learning curve for novice and non-technical testers.

**Selenium**
The Selenium Suite is a popular open source tool exclusively for testing web-based applications through browsers. The Selenium suite offers the choice of the Selenium IDE or Selenium RC (Resource Control) (previous version) or Selenium WebDriver (newer version), depending on the level of testing needed.

**Pros**
The open-source software has a zero up-front cost, and focuses on easy to create tests for web browser testing.

It has a record and playback feature for creating tests without scripting in a test domain-specific language (Selenese).

User actions can be recorded in popular languages like Java, C#, Perl and Ruby.

**Cons**
Selenium is for automating browser testing only. Tests are challenging to scale and maintain for large-scale projects.

Selenium tests need to start the browser and stop the browser between each test. Tests can sometimes break due to long page load times.

Support and functionality updates are only available through the open source community. Updates and maintenance require strong programming skills.

**Summary**
There are a lot of choices when it comes to test automation. Every tool has its plusses and minuses. Which one, or which ones, is the right choice for your organization will have many dependencies. Starting by thoroughly evaluating your needs and the team (and their skills) that will be working with and managing the tool will help narrow down your choices. Getting it right will provide a significant boost to your testing effort.
I have been researching sources of test tools and other resources that are inexpensive or free. In some cases, you get what you pay for. However, it is possible to automate some or a lot of your testing using these tools. The good news is that the investment at risk is low.

I have not included prices, since they change often ($ indicates there is a cost). Please visit the web sites to get more information on these tools.

**Test Design**


**AllPairs** (Free) - [www.satisfice.com](http://www.satisfice.com)

**PICT** (Free) - [http://msdn.microsoft.com/en-us/testing/bb980925](http://msdn.microsoft.com/en-us/testing/bb980925)

More pairwise tools at [www.pairwise.org](http://www.pairwise.org)


**METS** (Free) - Minimal Essential Test Strategy These are Excel spreadsheets but help in planning and estimating tests. There is also an iPhone app in the iTunes store. [www.gregpaskal.com](http://www.gregpaskal.com)

**Shapes** (Mac, $) - [www.shapesapp.com](http://www.shapesapp.com)

**yEd Graph Editor** (Free) - [http://www.yworks.com/en/products_yed_about.html](http://www.yworks.com/en/products_yed_about.html)

**GraphWalker** (Free) - [http://graphwalker.org/](http://graphwalker.org/)

**Test Management**

These tools help you plan and organize testing.

**Bromine** (Free) - [http://bromine.seleniumhq.org/](http://bromine.seleniumhq.org/)

Part of the Selenium suite of tools. Web-based QA tool for Selenium, that enables you to easily run Selenium-RC tests and view the results.

**QA Book** - Windows-based - Free - Allows you to create, edit and manage requirements, test cases, defects, environments, project success criteria, reporting, automated testing and more - [http://www.freetestmanagementtool.com](http://www.freetestmanagementtool.com)

**QA Traq** (Free) - Provides a central location for all test documentation with full revision control. [www.testmanagement.com](http://www.testmanagement.com)

**Testopia** (Free) - A test case management extension for Bugzilla. [http://sourceforge.net/projects/testopia](http://sourceforge.net/projects/testopia)

**qaManager** (Free). [http://sourceforge.net/projects/qamanager](http://sourceforge.net/projects/qamanager)

**TestLink** (Free) - Web-based Test Management sys-
TestMaster (Free) - A test case logging, reporting and test automation tool, much like the commercial product Test Director.
http://testmaster.sourceforge.net/

Zeta Test (Free and $) - An integrated test management environment that enables you to perform black-box tests, white-box tests, regression tests or change management tests of software applications. www.zeta-test.com

XStudio (Free and $) - A fully-graphical application developed in Java (so theoretically deployable on any OS) (Free) - http://www.xqual.com

Test Frameworks

Open2Test (Free) - The Open Source Test Automation Framework is composed of a standard set of keywords that doesn’t change, no matter what automation tool you use to automate testing - http://www.open2test.org

STAF (Free) - An open source, multi-platform, multi-language framework designed around the idea of reusable components, called services (such as process invocation, resource management, logging, and monitoring) - http://staf.sourceforge.net/

Test Automation

Apodora (Free) - A framework/tool for automating functional testing of web applications. It provides the user with programmatic control of the web browser allowing them to interact directly with the browser’s user interface. It uses a database backend in order to remember how to find your html elements. This also makes your scripts easier to maintain - http://www.apodora.org/

AutoIT (Free) - http://www.autoitscript.com/autoit3/index.shtml


Cubic Test (Free) - Graphical Eclipse plug-in for writing Selenium and Watir tests. It makes web tests faster and easier to write, and provides abstractions to make tests more robust and reusable - http://www.cubictest.org/

Cucumber (Free) - Cucumber lets software development teams describe how software should behave in plain text. The text is written in a business-readable domain-specific language and serves as documentation, automated tests and development-aid - all rolled into one format. Works with Ruby, Java, .NET, Flex or web applications written in any language. It has been translated to over 40 spoken languages. http://cukes.info/

Expect (Free) - Expect is a tool for automating interactive applications such as telnet, ftp, passwd, fsck, rlogin, tip, etc. Expect really makes this stuff trivial. Expect is also useful for testing these same applications. And by adding Tk, you can also wrap interactive applications in X11 GUIs - Expect can make easy all sorts of tasks that are prohibitively difficult with anything else. You will find that Expect is an absolutely invaluable tool - using it, you will be able to automate tasks that you’ve never even thought of before - and you’ll be able to do this automation quickly and easily* - http://expect.nist.gov/

Fake (Mac $) - www.fakeapp.com

FitNesse (Free) - http://fitnesse.org

Frankenstein (Free) - http://frankenstein.openqa.org/


Selenium (Free) - A Firefox add-on that will do simple record-and-playback of interactions with the browser. http://seleniumhq.org/

Sikuli (Free) - http://sikuli.org/
WET (Free) - Commercial grade Opensource Web automation testing tool - http://wet.openqa.org

**Macro Recorders and Players**

These tools capture keystrokes and even create simple scripts that can be replayed.

iMacros (Free) - https://addons.mozilla.org/en-US/firefox/addon/imacros-for-firefox/

Macro Scheduler (Free, $) - Macro Scheduler is my favorite because of its full-featured scripting language - http://www.mjtnet.com/

Mouse and Key Recorder (Free) - http://www.kratronic.com

iOpus Internet Macros ($) - http://www.iopus.com/iim.htm

Macro Express ($) http://www.macros.com/

**Comparison Tools**

ExamDiff and ExamDiff Pro (Free) http://www.prestosoft.com/examdiff/examdiff.htm

Beyond Compare ($) - http://www.scootersoftware.com/

Compare It! ($) http://www.grigsoft.com/wincmp.htm

Total Commander (Free) - http://www.ghisler.com/

WinMerge (Free) - http://winmerge.sourceforge.net/

**Load and Stress Tools**

Web Server Stress Tool (Free, $) - http://www.paessler.com

Load (Free) - www.pushtotest.com

LoadUI (Free) - www.loadui.org

Jmeter (Free) - http://jakarta.apache.org/jmeter/

**Screen Video Capture**

These are great tools to document a test. Screen video is captured and saved in either avi format or wmv format, which is about 1/10 the size of avi files.

Windows Media Encoder (Free) - Search on the term "Windows Media Encoder" to find it on the Microsoft site - www.microsoft.com

Jing (Free) - http://www.techsmith.com/jing.html

**Screen Shot Capture**

Clarify ($) - Capture, sequence and annotate screen-shots to create documents that are a great alternative to screen recordings - http://www.clarify-it.com/

**Test Drivers for Unit Regression Testing**

TestMaker 3.3 (Free) - http://www.pushtotest.com

NTest for .NET components (Free) - http://sourceforge.net/projects/nunit

JTest for Java (Free) - http://www.junit.org

**Interactive Test/Debug**

PEBrowse Profession Interactive (Free) - A very powerful, versatile, and customizable Win32 user mode debugger/disassembler. The debugger fully supports Microsoft .NET managed processes and seamlessly allows interop or mixed-mode debugging. It can be set as the startup debugger using the system registry Image File Execution — PEBrowse Professional Interactive
Configuration Management

Syncronize It! ($) - http://www.grigsoft.com/winsin.htm

FolderMatch (Free, $) - http://www.foldermatch.com/

Test Data Generation

This is a really cool web script that you can use for free and even download the script for use on your own server - http://www.generatedata.com

Training

http://www.onlinecourses.com/ is a free and comprehensive resource that is a collection of open college course that spans videos, audio lectures, and notes given by professors at Harvard, Princeton and MIT. These include highly relevant courses such as iPhone Application Development from Stanford and Cyber Humor from Oxford.

http://testingeducation.org/wordpress/ - Free courses in software testing from Kaner, Fiedler & Associates, LLC.

https://www.udacity.com/course/cs258 - One or two testing courses, but many other free courses for web development.

http://udemy.com - Has many free and cheap online training courses for a wide variety of topics, including testing. I personally take courses on this site.

My free videos on YouTube - I have several playlists, including one for testing concepts and principles, and another for "How To" topics.

(Why do I list free courses when I sell my courses? 1. You may have limited funds, 2. To be helpful, 3. That’s how I am.)

About Randall

Randy Rice is a thought leading author, speaker and practitioner consultant in the field of software testing and software quality. Rice has worked with organizations worldwide to improve the quality of their information systems and optimize their testing processes.

His website is www.riceconsulting.com.
His blog is randallrice.blogspot.com.

Other great LogiGear resources for test automation

We have a lot of resources available to help with getting the most from test automation.

Past Articles:

Action Based Testing: The solution for Agile test automation.

Misconceptions About Test Automation
http://www.logigear.com/magazine/issue/misconceptions-about-test-automation/

Leverage Automation to Make Mobile Testing Manageable
http://www.logigear.com/magazine/mobile-testing/2645/

The Real Cost/Benefits of Test Automation

Check more articles on testing under the resources tab at www.LogiGear.com:
This book isn’t for everyone, but everyone can get some value out of it. What I mean by that rather confusing statement is that folks working in Agile environments will likely want to throw the book across the room while folks in more bureaucratic environments like CMMI or other waterfall environments will likely get a great deal of value from the book.

I’m an Agile fanatic and I had a difficult time dealing with book’s approach which emphasizes spending large amounts of time creating documentation such as requirements traceability matrices, detailed test plans, etc. My preferred approach is to have testers working side-by-side as part of a team, creating specifications from user stories/requirements and moving those right in to automated test suites via tools like Selenium, Cucumber, or RSpec.

That said, I did indeed get some good value from the book. I found the discussions on making hard evaluations on what to test very worthwhile reading: teams can easily vaporize large amounts of time creating large suites of brittle, unmaintainable automated tests. This book has several really good chapters on using business cases to drive return on investment (ROI) decisions for testing, understanding automated test pitfalls, and adjusting your testing as you progress through your project.

Additionally, one of the book’s high points was on building the test team: “Put the Right People on the Project – Know the Skill Sets Required.” This is a great chapter which emphasizes starting the search by focusing on how to interview test team members – and how those testers’ skills are greatly different than other members of the team.

The book’s very academic, dry tone makes for some difficult reading, and few concrete examples are used until very late in the book. Having spent a large number of years either in the DOD or working for DOD contractors, it quickly became apparent that much of the book seemed targeted to folks working in those environments – too many dry acronyms are scattered through the book, adding to the difficulty in reading.

The lack of examples using real tools frustrated me. While the appendices contain some examples of comparing various tools, the book doesn’t actually show how a real world testing environment would use those tools. One appendix, eight or nine pages in length, is touted as a “Case Study” but falls short, in my opinion.

Overall it’s a decent book. The dry tone and lack of real environments is balanced out by the excellent coverage of team skills and emphasis on selecting how and what you test.

About Jim

I work for Falafel Software and get to chat with lots of great folks about what works and what doesn’t in the world of automated testing – something I’m very passionate about. I’m also a Father trying to remain sane while trying to build great software, herd my kids around, fix school lunches and handle the yardwork.

Twitter: @aJimHolmes
Blog: www.frazzleddad.blogspot.com
Test Tool and Automation Glossary

Utility: A program that performs a specific task related to the management of computer functions, resources, or files, as password protection, memory management, virus protection, and file compression.

Tool: A program or application that software development teams use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to programs that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object. The ability to use a variety of tools productively is one hallmark of a skilled software engineer.

Automation Engineer: A discipline within systems engineering that focuses on how to design and manage complex engineering projects over their life cycles. Issues such as reliability, logistics, coordination of different teams (requirements management), evaluation measurements, and other disciplines become more difficult when dealing with large or complex projects.

Coach: A coach is a person with special skills, knowledge or competency who works with individuals to improve their skills or performance. As opposed to just providing instruction, a coach works alongside or with an individual or small group to assist them in learning.

Jump Start: Slang for energizing a start-up process so that momentum can be created faster. It was adopted from the actual meaning of temporarily connecting a fully charged battery to drained battery in order to start a combustion engine.

Test Harness: A test harness is a collection of software and test data configured to test a program by running it under varying conditions. The test harness is a hook to the developed code which can be tested using an automation framework.

Shelfware: A slang term used to describe owning or licensing software that you don't actually need or use (e.g. it sits on a shelf). Shelfware is a phrase that is commonly used in the enterprise and corporations where volume license prices are given and the corporation purchases more software than really needed to obtain that discount.

Test Automation Framework: A test automation framework is an integrated system that sets the rules of automation of a specific product. This system integrates the function libraries, test data sources, object details and various reusable modules. These components act as small building blocks which need to be assembled to represent a business process. The framework provides the basis of test automation and simplifies the automation effort.

Application Lifecycle Management (ALM): ALM is the product lifecycle management (governance, development, and maintenance) of application software. It encompasses the requirements management, software architecture, computer programming, software testing, software maintenance, change management, project management and release management.

Integrated Development Environment (IDE): An IDE is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. It can also include test tools.

Continuous Integration (CI): CI is the practice of merging all developer working copies with a shared mainline several times a day.

Testability: The effort required to test a product to ensure that it performs its intended function.

Design for Testability: The architectural and design decisions in order to enable us to easily and effectively test a system.

This includes traditional software development terms such as Controllability, Observability and Test Point Insertion. But also designing a low maintenance UI, and includes low level design ideas such as using as many standard controls as possible and as few custom controls as possible. The awareness of “design for testability” and the effort toward this type design will have a big benefit for testing and a huge benefit for lower maintenance test automation.

High Volume Automation: A test automation program not for a small set of tests, but for a large set of tests, devices, platforms or data. More thought must be used in architecting a larger system with better test design, lower maintenance, better defect analysis, easier data management, more visibility into what is actually being tested and better reporting. Small scale automation is, in many cases, no longer worth the effort. While high volume automation presents much greater value, it needs to be more carefully built.
Vietnam View

Historic Architecture of Saigon

‘The Pearl of the Orient’ boasts some of SE Asia’s best examples of historic architecture. While much has been lost, the city still offers a plethora of beautiful old buildings

By Brian Letwin, LogiGear Corporation

Saigon, once known as ‘Pearl of the Orient’ claims some of South East Asia’s most stunning historic buildings. While lack of preservation laws have resulted in the demolition of 53% of the city’s heritage buildings over the past decade, Saigon still boasts some very impressive examples of colonial and art deco architecture. Here are some of my favorites:

The HCMC People’s Committee Building

Built in 1902, this building was originally known as the Hotel de Ville. Located in the city’s central district, it was converted to government offices in 1975. Even so, it retained all of its original ornamentation and is one of the city’s most photographed buildings, especially when lit at night.

Saigon Opera House

The Saigon Opera House (aka the Municipal Theatre) is one of Saigon’s oldest buildings and one of the city’s best examples of classical French architecture. Constructed at the turn of the 20th century, it has performed many functions throughout its history, from theatre to refuge shelter to legislative offices.

Ben Thanh Market

Built by French contractor, Brossard et Maupin, the city’s central market was completed in 1914. More than 100,000 people came out to enjoy the celebratory parade and fireworks when it opened. The market operated continuously until 1985 when it underwent a major renovation. Today it is a prime location for tourists and locals alike, highlighted by a large swath of food stalls in its interior.

Notre Dame Cathedral

The Notre-Dame Cathedral is one of Saigon’s oldest and most iconic buildings. Built with imported materials from France, it was constructed between 1863 and 1880 in an effort to promote Catholicism in Vietnam. Due to its long history, it has been a centerpiece of Saigon’s urban narrative for over 150 years and an enduring reminder of the city’s colonial heritage.