Considerations Before Starting Test Automation

Steven Vore

Steven.Vore@Telerik.com

Abstract

Software is complex, and complex software doubly so. Our testing teams, comprised of fallible humans, are challenged with finding the tiniest of errors in a seemingly infinite domain. Once we're satisfied with a version we start all over again. It's no wonder that teams are looking to automation in hopes of easing their workload.

Company after company, testing team after testing team is asking, "How do we get started with test automation?" The wrong answer for our testing teams is to buy a product or download a framework and start coding against it. This paper and presentation focus on key questions to ask and discussions to have before pulling out the checkbook, keyboard, or mouse.

We will begin with assessing team skills; determining the level of technical skills and interest amongst the testers and developers. I will discuss the need for finding allies and for getting management buy-in before starting – or forgiveness after getting underway. You'll learn methods for identifying the areas of your project that would be good starting points for automation, and the importance of determining a course of action.

Wise testing team managers should remember to "look before you leap" into automation. Just as a development team would be mistaken in choosing a programming language or database tool before understanding a new project, we should be asking the important up-front questions before diving in.

Biography

Steven Vore is a Test Studio Evangelist at Telerik, currently working from his home office in Atlanta, Georgia. He has worked in software support and testing for the better part of two decades, and enjoys exploring ways to make software easier to use. Steven has worked on projects in defense contracting, networking software, telephony and medical information.

Steven has a degree in Information Systems Management from the University of Maryland and has earned several technical and project management certifications.

1. Introduction

The world of software development is a complex one, often encompassing multiple frameworks and technologies, using various patterns employed by architects and programmers of varying skill. Our testing teams, also comprised of humans with multiple backgrounds and skills, are challenged with finding the tiniest of errors, and once found and fixed we're expected to ensure that errors don't creep back into the product. It's no wonder that testing teams are looking to automation in hopes of easing their workload.

At conferences, in online forums, and over coffee tables, managers around the world are asking themselves and each other "how do we get started with test automation?" As with any question in a technical realm, it is easily answered with a pointer to a blog post to read, product to purchase or framework to download – but that is the wrong answer. Wrong answers are easy to get when the wrong question is being asked.

Tools and frameworks do not automate testing, people do. Before jumping to a technical solution, testing managers need to evaluate their organization and team members, asking themselves instead "are we ready to automate?" Some important questions to consider, the answers to which will help make this determination include:

- What are the norms and expectations in our organization?
- What level of skill and interest do the testers have
- What sort of acceptance and assistance can we expect from other teams?
- What level of support will our management provide?

Only after these have been discussed can a testing team realistically begin to identify the areas of their project that would be good starting points for automation, and to choose the tools to do so.

2. Determining your goals

There are many reasons for test automation; the most often-stated business goals include reducing errors and saving time. Traditionally testing has been a manual and often scripted, repetitive process, and humans are easily bored and make mistakes. Manual testing is also slow, and the time required grows as an application is built. Subsequent modifications and versions require re-testing, which increases both the time required and opportunities for errors overlooked due to tedium.

An underlying goal for the testers themselves, which should be made more explicit, is that of having an opportunity to do their best work. Recent studies have shown that there are two internal desires that drive knowledge workers – challenge & mastery and making a contribution. Testers, developers, and others are motivated and do their best work when they are given the freedom to learn and master new skills and then use those skills in a meaningful way. (Pink 2011) Automating the repetitive and boring tasks can result in organizations that are more effective and, in the long run, more profitable.

It is important that not only the QA manager or test team lead make sure that all members of the team are in agreement on their goals, and that the rest of the organization – "upper management", developers, and others – understand and are on board as well.

3. Assessing the organization

3.1. Organizational Support

Software testers do not function in a vacuum. We interpret requirements from business analysts and customers. We seek information from and provide feedback to developers. We report our findings to product owners. Each of these interactions comes with some level of expectation and support. It is important to understand how others in the organization view the testing team, and how changes to our testing methods will be supported (or not).

Automation, while streamlining our tasks and saving time in the long run, will require some amount of additional time in its startup phase. If the testing team is valued as part of the development process, this time will be seen as an investment.

For many organizations, however, the "QA" or testing team is considered to be a necessary evil and an expense. This can be seen by a lack of budget for salaries, tools, and training. Other, more subtle, indicators come in the form of attitude: development teams who do not consider testability to be a desirable feature of their code, managers who allow scope creep to squeeze testing into a last-minute rush of late nights and weekends. "When an organization lacks an overall quality philosophy and pressures teams to get the product out without regard to quality, testers feel the pinch." (Crispin and Gregory 2009)

In this sort of environment, it is difficult to justify the investment required to properly automate testing. Test managers should begin by first increasing awareness of the value testing brings to the organization. Defect counts and time required for bug fixes before product ship are perhaps the easiest metrics to gather but are easily gamed and are not seen as terribly useful. Overall product quality, though much more difficult to measure, should be the goal. Product quality – or the lack thereof – can be seen by product owners, customer support and client service groups, and by looking at the number and severity of customer-reported issues. These people and metrics can have considerable influence in raising the testing team's perceived value.

Managers of testing teams should look to insert themselves and their team members into the product planning, design, and architecture process as early as possible. Working directly with product owners will afford an opportunity to point out adjustments to features which would make them more testable and often more helpful by end users as well. The testing team will also become more familiar with features' intended purpose, which helps design tests that do more than verify that the product matches the specifications – they can aid in insuring that the specifications are describing a product that will meet the users' needs.

3.2. Own the data

In order to automate any testing – UI or not – the tests must be repeatable without human intervention. This means that the results need to be predictable, and for this the underlying data must be in a known state. You may or may not think of your system as having data, and it may or may not use some sort of database back-end; for all but the simplest applications there is some sort of data on which you need to be able to rely. Your system likely also has user accounts (usernames, passwords, permissions and other settings). Think of any element of information, which, if changed, would cause a tester to diverge from a specific character-by-character script. For automation to be viable, the testing team must be able to control all such data in the testing system.

Does the development team use the testing system's database for experimentation and exploration? Does the product owner or sales team use the test system for demonstrations and training? Does the production support or operations team use the test system for their own "playground," verifying data-manipulation processes prior to implementing them on a production

system? If any of these are normal in your organization – especially without the testing team's knowledge or authorization – the data may not remain in a stable.

Evidence of this is easy to find: simply examine the language used when describing testing tasks. If testers are required to look up values or independently calculate results to verify tests' success or failure upon each execution of the test, it is unlikely that the data is stable enough for the tests to be automated. The lookups or calculations themselves could be codified into the automation, but this would then become code which itself would be subject to inspection and testing.

In an optimal world, the testing team would have its own environment. The team would control software builds, configuration, and data upon which tests would be performed. There are many reasons why this is often not possible; the most-often cited are financial and manpower. Organizations are unable to purchase or lease additional servers and licenses, and developers, testers, and system administrators are not given sufficient time to setup and manage another set of systems.

Fortunately, most applications can be sufficiently managed so that this is not an insurmountable problem. As is often the case, the solution comes down to conversation, agreements and cooperation rather than technology. Each team can, for example, agree to use a their own set of user accounts. As long as changes in the data used by the various accounts does not impact other user accounts' data, each team can function independently. These agreements can be – in fact need to be – seen as beneficial to all. Not only will a stable data set give the testing team an opportunity to streamline, and perhaps automate, their testing; training and demonstrations can become more stable.

Other teams may be reticent to agree to such a split; often this is simply due to the time required to setup new data or a lack of knowledge of how to do so. In cases such as this, the testing team can offer assistance with creating data setup and cleanup scripts. This can ease the agreement, making this an even more palatable to other teams. It can also give you time and opportunity to document the data model and setup process, if that has not yet been done.

As part of any discussion, all parties will need to come to an understanding and agreement on management of application-wide configuration as well as scheduling of deployments.

4. Assessing team skills

4.1. Technical skills amongst the testers and developers

Do any of the testers have a background or training in programming, scripting or automation? Are they comfortable writing scripts for your system (Unix/Linux/OS X shell scripts, Windows PowerShell scripts, SQL query scripts), or will they need to be trained from scratch?

Let your testers' skill set determine what portions of the application's testing to automate first. For example, testers with strong SQL skills may start with writing queries, scripts and stored procedures to create or setup data, and then to test backend processes and application-level stored procedures. Testers with knowledge of HTML could begin with a record-and-playback tool, preferably one that will allow them grow into custom-coded steps as they progress. Team members who have knowledge of programming could work with the developers to enhance unit tests, or explore one of the frameworks available for User Interface testing.

Start with some wins, providing positive feedback to the team and showing positive movement for managers. This will ease the transition into training, learning, and experimenting with less-familiar territory.

4.2. Interest amongst the testers and developers

Not all testers are interested in making the move to automation. As with any profession, unless your hiring and training practices specifically target individuals with scripting and automation skills you will have a range of interest and motivation among your team members.

Managers should being by asking themselves “do I have any team members who have already shown an interest in automation?” Look for testers who have spent some of their spare time looking into what can be done to streamline their tasks. When putting data together, for example, you may have testers who have created spreadsheets and batch files rather than typing fresh data or copying from documents. These individuals can also often be found talking with developers, looking to understand how the product is being developed. Given encouragement and a little freedom, testers such as this can grow into the stars they want to be.

Developers: how testable have they made the application thus far; are they using Test Driven Development methodology or unit tests at all? Testers will often need support when coding automated tests. If testing, in your organization, is considered to be a menial task to be done by “the QA team,” what amount of support or assistance will the developers be willing or able to provide? Many of us have worked with developers who eschew any form of testing, choosing to “throw code over the wall” to testers without any earned confidence in its quality.

A former co-worker of mine often claimed that his “unit tests passed” – though he had coded no tests at all; his claim was based entirely on the fact that the application opened in a browser. Accompanying this was a refusal to add IDs to HTML elements. His attitude toward testing made his relationship with testers uncomfortable and their attempts at automation extremely difficult if not downright impossible.

5. Identifying allies

5.1. Why

Although most tool vendors wouldn’t consider it a popular statement, automated testing will always require some level of coding for several reasons.

Code-generation and Record & Playback tools have improved over the years. However, so have the applications they’re being used to test. Applications grow not only in technical complexity but also in user interface creativity. The likelihood of tool creators to be able to foresee every desired set of actions/verifications is dubious at best.

Most testing scenarios include a need for setup before executing or cleanup afterwards. For example, a set of tests may be required to ensure that new user accounts are created with the proper set of attributes based on different authority or roles of the creating user. As discussed under ‘own the data,’ these scenarios are only repeatable if the underlying data is in a known state. A well-written test scenario – for humans or automation – should include steps to ensure that the test does not fail merely due to duplicate data, and that data created during the test is removed in a proper fashion afterwards. These are not “test steps” in the classic sense, they are not steps upon which failure or success should be reported, but they are required nonetheless. Such setup and cleanup steps typically involve more direct access to the system than the test steps; they may call a stored procedure created expressly for, if not by, the testing department. Setup steps should be automated wherever possible; not only as part of test automation but also to assist manual testing.

Testers often produce code for tasks which simply cannot be done by Record & Playback tools. Samples of this include data creation, manipulation, and cleanup; defect tracking and consolidation; task and product documentation and reporting. This type of work supports the testing effort, and is “code that helps the product get out the door sooner.” (Page 2013)

The idea that testers “might code in different ways, for different reasons, but they can be coders too” (Christie 2010) is not new. Some test team members are comfortable writing their own code. Their coding skills are often not formally recognized, and in many organizations training in programming techniques and practices is not provided. The result is that testers rely heavily on their programming-centric peers for assistance. In some cases assistance is sufficient, lending suggestions and support as a tester writes their own code; in other cases there is a complete dependence, with little or no programming actually done by the testing department.

Perhaps of most importance is automating tasks that directly support not only testers but developers as well. This list includes automated build and continuous integration systems as well as unit tests. These are typically built and maintained by development groups, but if they are not in place the testing team may need to take the lead in making that happen.

5.2. where to find them

Given that testers need assistance with coding and related tasks, from where does this support come? The obvious answer is “the developers,” the organization’s formally-hired and -trained coders. Testers should be cultivating good professional ties with the developers even if this were not the case, and this symbiotic relationship benefits the organization as a whole as it fosters quicker, higher-quality product delivery. Product development and testing should be seen as a team effort, with developers and testers both bringing their skills to bear.

There are likely others in the company who have development expertise, and to whom testers should extend a hand. There is often quite a bit of coding done in the production support or “DevOps” team, for example, by the individuals who are tasked with keeping systems running day-in and day-out. These teams quickly learn, usually through hard-earned experience, the internal workings of the systems. Database and scripting tasks are their specialty, as are the connections required to deploy and manage the application. This is invaluable information for a tester.

Additional support – technical and morale – can be found outside of the company as well. In many cities, user groups can be found using online tools (meetup.com, lanyard.com, etc). These groups are valuable resources for more than QA/testing topics; groups exist for discussing and learning processes and methodologies, such as Scrum and Acceptance-Test Driven Development, as well as technology-specific topics including programming languages and frameworks. Organizers and attendees welcome – and rely on – guests and new members who are eager to learn. In-person provide an opportunity not only for learning, but also for general discussions during which ideas can be vetted and to see how our practices compare with those of our peers. Managers of test organizations should not only attend external meetings but also encourage their team members to do so as well.

6. Discussions with Management

Often, when the topic of test automation is discussed with IT Directors or other “upper management,” the initial thoughts are of UI Test Automation and questions about cost. Testing team managers should be prepared to address these topics. Approaching “upper management” unprepared can lead to unanswerable questions and projects being shot down before they even get off the ground.

As discussed above, automation within the testing realm consists of a variety of coding activities, of which UI Test Automation is a visible part. I suggest that the team discuss their end-to-end environment with this in mind, determining a list of all the possible points of automation, which have the potential for best long-term time savings, and suggestions for methods of attack. Each area of potential automation should be examined with regard to tools available; in-house, open-source, and commercial.

Where interest is shown, and small slices of time can be made available, small “skunkworks” projects can be valuable. These have a short timeframe and specific goal in mind – to determine the viability of a particular tool or technology for use in automating a specific slice of the testers’ duties. The idea is to give the team room to grow and prove or reject a concept without interrupting the existing workload. After one or more of these have been completed, a testing team can make an informed presentation.

Test managers should take care with regard to presentation of in-house or open-source tools and frameworks as “free.” There is a tendency to see cost of software separately from the cost of the time required to use it properly and effectively. Discussions should always include this investment in time as well as any monetary requirements, and a reminder that results may not be seen immediately.

7. Where to begin

Testers, being technical people, have a tendency to pick a tool or technology that appeals or is popular and dive right in. It is important for the team to step back and ask some questions first – “where will automation be the most beneficial,” and “what’s the most painful or boring part of our jobs today?” Taking the answers these questions in conjunction with tools such as the Test Automation Pyramid (Cohn 2009) will lead to good starting points. To many, “automated testing” means using tools or code to drive an application via its user interface, but this should constitute the smallest effort as it provides the least ROI. These often test then entire application rather than individual components, and are therefore relatively slow to execute. Tests at lower levels of the pyramid are more less fragile, longer-lasting, and more effective.

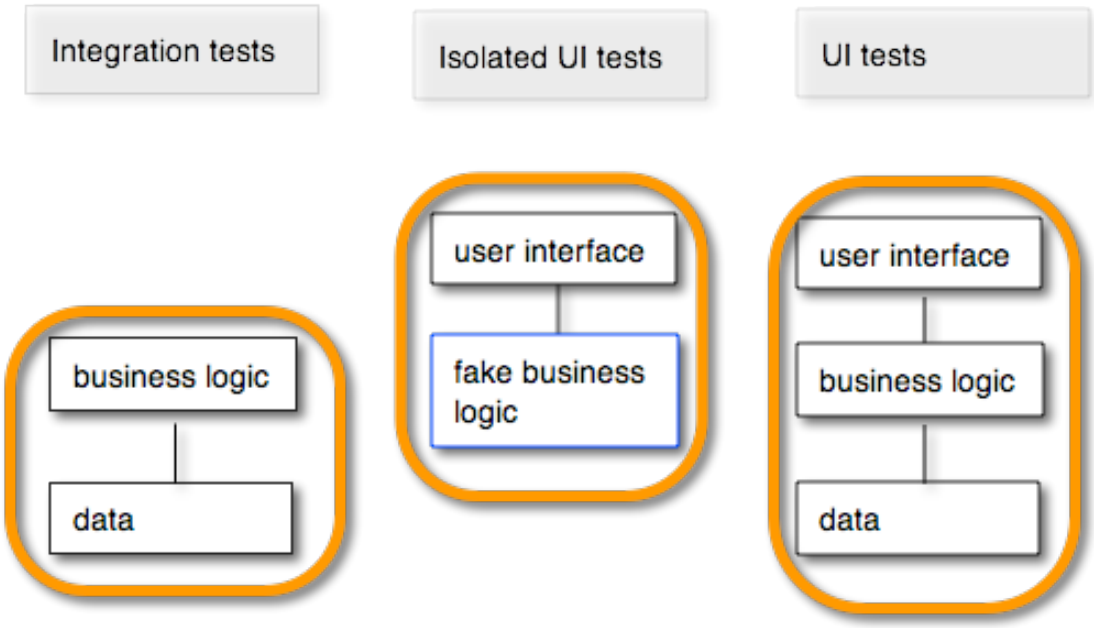
The lowest level possible is the base on which the pyramid sits, the foundation of good current development practices. Without a stable, automated and quick build process, both developers and testers suffer; this should be the first thing on any team’s automation list. This will need to be a joint effort between developers and testers, perhaps with assistance from system integration or production support teams in the case of new environments.

Unit tests, in theory at least, would be the province of the development team rather than the testing team. Unit Testing is sufficiently important, and testers should be aware of the developers’ use of unit tests, encouraging and assisting wherever possible.

The bulk of testers’ time, with regard to automated testing, should be spent working – alone or with developers’ assistance – in integration or “under the UI” testing. Integration tests exercize business logic, often combined with database or other back-end access.

User Interface tests where possible, should use an interface which calls “fake” logic. This will allow the UI iteself to be tested quickly and independently. This is becoming increasingly important with client-side validation and actions become popular.

Finally, a minimal number of traditional end-to-end UI tests will be created.



8. Summary

As we have seen, Test Automation can cover a wide variety of activities, from unit tests and continuous integration to driving the actual user interface. Testing and development managers and team members have several topics to discuss while considering automation; as with any activity, preparation is key. Wise testing team managers will remember to "look before you leap" into automation, leading their team through the important – and often non-technical – discussions that can lead to success.

9. References

Christie, James. *Testers and coders are both developers.* October 12, 2010. http://clarotesting.wordpress.com/2010/10/12/testers-and-coders-are-both-developers/ (accessed June 2013).

Cohn, Mike. *Succeeding with Agile: Software Development Using Scrum.* Boston, MA: Addison-Wesley Professional, 2009.

Crispin, Lisa, and Janet Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams.* Boston, MA: Addison-Wesley, 2009.

Page, Alan. *Coding, Testing, and the "A" Word.* June 5, 2013. http://angryweasel.com/blog/?p=649 (accessed June 2012).

Pink, Daniel. *Drive: The Surprising Truth About What Motivates Us.* New York, NY: Riverhead Books, 2011.