Are We Failing at User Interface Test Automation?

Jim Holmes
Jim.Holmes@Telerik.com

Abstract

Time and time again over the last several years I've run in to individuals, teams, or organizations that are struggling or outright failing with test automation at the User Interface level. Some of these people are members of teams that are writing great systems. If they're able to write great systems, where's the disconnect with having so much difficulty around test automation? These issues aren't isolated ones; rather, they're much more common than many in our industry are willing to admit.

This paper and presentation focus on some of the main reasons for difficulties in good test automation: lack of understanding of fundamentals with the technologies, failure to recognize that automation efforts are software engineering efforts, and a failure to focus on high-value automation over low-value tasks. This paper is completely technology agnostic—these issues cut across all domains.

In this paper I will discuss ways to avoid these pitfalls, and ways to ensure your teams and organizations are able to identify and overcome these challenges. We'll walk through the importance of approaching automation as a whole-team effort. You'll learn to identify candidates for good automation and which task to avoid. You'll also come away knowing why it's critical to have a solid understanding of the technologies involved.

Finally, you'll take away why it's so critical to keep your team and organizations constantly assessing their efforts, and adjusting where needed – ***JUST LIKE GOOD SOFTWARE TEAMS DO.***

Biography

Jim Holmes is the Test Studio Evangelist at Telerik. He has over 25 years in the IT field in positions including PC technician, WAN manager, customer relations manager, developer, and yes, tester. Jim has held jobs in the US Air Force, DOD sector, the software consulting domain, and commercial software product sectors. He's been a long-time advocate of test automation and has delivered software on a wide range of platforms. He co-authored the book Windows Developer Power Tools and blogs frequently at http://FrazzledDad.com. Jim is also the President of the Board of Directors for the CodeMash conference held in the middle of winter at an indoor waterpark in Sandusky, Ohio.

1 Are We Failing at UI Automation?

I define success in User Interface (UI) automation as building a suite of sustainable tests which are part of a team's overall strategy of delivering great value to our customers. Given this yardstick, "Are we failing?" has several answers depending on how you look at the question.

No: Great tools around UI automation are seeing exciting adoption rate increases. Selenium/WebDriver, Watir, and commercial tools continue to see growth across the industry. Selenium has eclipsed Quick Test Pro in jobs listings for the first time ever. More organizations and teams are beginning to understand the importance of a whole-team approach to test automation.

Maybe: In the software development domain we've seen an explosion of extraordinary methodologies such as Test Driven Development (TDD) or Behavioral Driven Development (BDD). We've seen a tremendous amount of books, articles, conference presentations, and training courses around these methodologies. The software craftsmanship movement is fostering lots of discussion around how to best build high-quality, valuable software. All that's great, but where's the corresponding content and discussions around UI automation?

Yes: Particularly in regards to enterprises and with newcomers to automation. Enterprises are often struggling with difficult development processes and environments. Automation can be difficult to implement in these places because of a lack of skills and poorly set expectations with the teams, stakeholders, and other management. Newcomers to UI automation are often lacking fundamental skills and knowledge, and are often put in positions with no useful mentoring to help them succeed.

These failures, and their impacts, are traceable to a number of critical human and cultural factors. There are also a number of problems relating directly to technology.

We can fix these problems, but we need to first acknowledge it's going to be a long haul and will require a lot of effort. We can look to the software craftsmanship movement as a great guide to help us bring the automation segment of our organizations up to a level of great proficiency, which will enable us to deliver even better software to our stakeholders and customers.

2 Impacts of Failure

Roy Osherove's *Art of Unit Testing* (Osherove 2009) opens with a preface about a project Roy was leading some years ago. Roy's team was new to TDD, and had decided to go 100% TDD with a new customer's project. Unfortunately, the team didn't have the skills to create good automated unit tests, and they quickly lost control of their test suite. Soon they were spending more time fixing their tests then actually building and delivering value to the customer. As a result the customer pulled the plug on the entire project. Please re-read that: the customer pulled the plug on the entire project. Not just the testing portion, ***the entire project.***

Automated tests, especially automated User Interface tests, can quickly spiral out of control into a huge morass of pain and frustration. It doesn't take much for a team to create a brittle, cumbersome, overly complex suite of tests that requires huge amounts of time for maintenance, leaving far too little time for actually delivering value to the customer.

Not only does this failure suck the morale out of the team, it can also lead to a fatal breakdown of trust with your stakeholders. The customers don't see value being delivered on a regular basis, and they start to question the value of any automation at all. It's easy to understand this point of view: "You're spending more time fixing your tests. Why not skip those and just write more code for the system you're supposed to be delivering to me?"

Worse yet, that breakdown of trust is rarely limited to the automated testing suite. The customer/stakeholder now begins to fear your team doesn't have the right skills or discipline to help solve their critical business needs. Your project may be in jeopardy because you're not able to get your automation efforts rolling along smoothly.

3 Failure from Human Factors Influence

Test automation is a complex domain, and there are many factors in our industry's failures with it.

As with every complex problem, human factors contribute to many of the largest issues. First, teams (I include stakeholders and management in this too) on projects don't realize or acknowledge that UI automation is at its heart a software engineering effort. UI automation requires skilled people, hardware, and time to get right. It's not something that's easily bolted in to a project after it is nearly completed. This same concept can be extended to the perception of Quality Assurance/Quality Control as a secondary effort in projects.

Additionally, manual testers can often be hesitant to get involved with automation efforts. Their reluctance is nearly always traced back to FUD, a great acronym used for Fear, Uncertainty, and Doubt: they feel automation will put them out of a job, they're fearful about being able to contribute anything of value to automation, and they're scared of having to learn the new skills required for automation.

Another problem involves a long-standing issue with those we look to for help: UI automation has had far too much snake oil sold to the industry from commercial tool vendors, consultants, and thought leaders in our testing domain. Too many commercial vendors tout their toolsets as a panacea and gloss over the long-term investment needed to bring a team up to an effective skill level. Those same vendors also fail to address the long-term view for test suite maintainability as a project ages.

Consultants don't get a pass here, either. Too many consultants have made their living focusing on pushing tools, both commercial and open source. They've pushed these changes in to environments without much thought about the long-term success with the teams that will have to maintain those suites, tools, and frameworks after the consultant has left.

Additionally, a number of thought leaders in the manual and exploratory testing realm speak dismissively of automation because they think it detracts from the positions they've staked out as leaders. This is a rather sad position for these leaders to take because it enforces the misperception that there is a single-facet solution to delivering quality software to our customers. The reality is that quality requires many different aspects from the whole team.

Making everything worse, we in the software realm do a miserable job effectively communicating the time and dollar impacts of UI automation, and we fail in communicating the value of UI test automation to stakeholders. This isn't surprising: we've taken far too long getting our customers and stakeholders to buy off on the value propositions behind quality software design and developer-level testing.

The final nail in the coffin is that, far too often, automation isn't seen as a software engineering problem – **but it is**! UI test automation has often been thought of as a test activity, or a problem solved by tools with no need for true development skills. Sadly, that's far from the truth. We need to acknowledge that test automation is a true software engineering effort and requires the involvement of the whole team from stakeholders through developers to testers.

4 Failure from Technology

The points above are all large factors in the success or failure of UI automation, but they're not the largest problem. At the heart of automation woes is one concise issue: It's a hard problem to solve!

UI automation hits a huge, complex array of facets across the entire project:

- Every piece of technology from the web servers down to the database for web applications
- Unfriendly platforms for automation in desktop applications
- Test-hostile UI designs
- Environmental issues (where do we test, and how often?)
- Hard to learn languages, platforms, frameworks, and toolsets

It's not hard to see that automation can be a struggle given the wide range of challenges I've listed. These are the same sorts of problems we run in to on the development side of projects. Developers have long had some great guideposts to look at in their domain such as Steve McConnell's *Code Complete* (McConnell 2004), Andy Hunt and Dave Thomas's *The Pragmatic Programmer* (Hunt 1999), or any of the great works by Robert "Uncle Bob" Martin. Sadly we don't have any similar works to look to for guidance around UI automation. Several works, like Fewster and Graham's *Software Test Automation* (Fewster 1999), come close, but they're not widely enough read, nor are they impactful enough to be considered in the same light as the aforementioned works.

Unfortunately, too many testers don't understand the underlying technical aspects of the systems they're working on, nor do they understand basics about automation. Time and time again I see teams of developers and tester who are missing the point when it comes to creating stable, repeatable tests. Too often they're relying on brittle, hardwired locators using XPath instead of looking to more appropriate things like ID values. I also see teams littering their test suites with pauses and fixed delays in order to overcome common problems with dynamic content from AJAX or similar service calls.

Testers who have a great grasp of focused, granular manual tests seem to lose their minds when working with automation. Automated tests cross numerous concerns, perform checks across multiple situations, and weave in to muddled, confused scripts hundreds of steps long.

Both the above situations lead to extremely brittle tests which fail intermittently and are incredibly painful to maintain when the system changes. Both situations tie directly back to concepts central to good software design: conciseness, simplicity, and solid design. These same general problems have been prevalent in the development side of our industry for some time; however, there's been a significant amount of attention brought to these problems over the last five to ten years from the viewpoint of software developers.

When I ask thought leaders in the testing industry about these same problems I tend to get blank stares and responses like "That's a problem? We solved that with our team ages ago." This really shows me that long-time practitioners of test automation have completely lost sight of the gap between their skillset and the sad state of the rest of the industry.

5 How to Fix It

First off, there aren't any quick fixes for this large problem. It's actually a number of problems, and most of them trace directly to human factors and cultural issues. Problems with those root causes are never easily fixed. We have to set our expectations that this will be a long-term effort with a lot of setbacks along the way. We must be willing to adjust failed approaches and think up new ones.

We also have to acknowledge that this problem has been around since the start of test automation. In 2001, Dawn Haynes wrote a tremendous article for Rational Edge (Haynes, 2001) highlighting many of the same exact problems we're still facing 11 years later.

A great starting point for this long haul is acknowledging that automated testing is a software engineering effort, not some after-the-fact bolt on QA task. Simply looking at the problem in this light helps us realize we need to look outside the narrow silo of QA/testers/whatever and bring in the rest of the project team to help us succeed.

Using a whole team approach to automated UI testing is critical to successful efforts. Bringing in all voices to the automation work gets us invaluable input from many viewpoints:

- **Stakeholders** identify what their most critical risk areas are. This helps the entire team understand what to focus automation efforts on. “Show me the money” tests might be important for some stakeholders; validating cross-browser functionality may be more important for others.
- **Project Managers/Scrum Masters/etc.** have a broad picture of the workflow and can contribute greatly to ensuring the right priority on value decisions around test automation happen. Additionally, they’re roadblock destroyers who can help when more resources and people are needed.
- **Developers** need to be pairing up with testers to create UI automation. Developers’ skills for software design apply directly to ensuring tests are avoiding duplication and are well designed from a software view. Moreover, developers can quickly flip over to the system under test and alter the UI as needed to improve testability. Additionally, they can assist with writing backing APIs to help with setup, teardown, and configuration steps.
- **Testers** need to be involved on nearly every project larger than a few pages for a static website. Developers and PMs don’t have the domain knowledge or skill to create high-quality tests that aren’t simply “happy path” tests.

Bringing the whole team in to automation efforts helps emphasize that automation is a software engineering task. Time and resources (*resources are **not** people!)* need to be allocated to get the tests done in the first place, and time needs to be set aside for ongoing technical debt payoff and refactoring. Moreover, thinking of automation as a software engineering effort helps us ensure we’re including automation in every step of whatever process the team is using. “Done Done” can’t be “Done” if there’s no automation around the feature!

If we’re buying off on the whole team approach - and we absolutely need to be! - then we need to work hard to bring manual testers in to the fold. As I mentioned earlier, I repeatedly see manual testers who are extremely resistant to automation because of simple FUD. They’re worried they’ll be out of a job, or they’re worried they won’t be able to contribute in a meaningful fashion. The team has to show manual testers their domain knowledge has tremendous value. We have to show them that automation frees them from rote repetition of regression suites and frees them up to spend time doing high-value exploratory testing. They’ll finally be able to put their brains to great use instead of simply following scripts!

Continuing in this educational bent, we must raise the basic skill levels of our folks working with test automation. People working on automating web applications can’t continue without a fundamental understanding of page load lifecycles, DOM structures, element locators, and dealing with dynamic content. Teams shouldn’t be looking to conferences to save them in these areas. Instead, these areas are easily covered through things like brown bag lunches, user groups, meet ups, and many other self-organized opportunities.

This dovetails right in to a larger problem we have across the entire software industry: a miserable state of mentoring. Focusing in on the automation issue, we need the more experienced people on our teams to step up to the plate and mentor those behind them. Empowering teams to get rolling on automation, and supporting them as they work through the difficult learning process is critical. (See the Conclusions section for some specific action items on this topic.)

That said, we need to ensure we’re getting more technical presentations at conferences. Too many well-known conferences are deserts when it comes real-world, take-this-home-and-put-it-to-use practical sessions involving anything around the technical side of testing. Developers need to start submitting talks to conferences traditionally thought of as QA-only domains. People working with test automation regularly need to be submitting talks and white papers about their experiences. Passing this knowledge along to conference attendees will be a great boost to the overall skill level of the industry.

Tool vendors, both commercial and open source, need to step up to the plate, too. Stop pitching the tools and frameworks as panaceas and be forthright about the level of effort that will be necessary. Consultants pitching their efforts around tools, commercial or open source, also need to be much more honest about the long-term impacts on the organization after the consultants leave test suites to the organizations when the projects are done.

Open source leaders need to step back and really focus on getting their projects' documentation up to snuff. Getting workers to write documentation for open source projects isn't ever easy, but it's time to step up to the plate and get usable, current documentation in place that educates newcomers. Simply pointing to Google searches for blogs doesn't cut it any more.

Moving away from the tooling aspect of this and returning to the human factors/cultural issues, I think that we in the testing industry can actually take a lot of heart in the passion, traction, and change fostered by the software craftsmanship movement. This movement has a great many voices and opinions, but at its root is an acknowledgement that things we've been trying haven't worked well, and that we need to get fired up about our work. The software craftsmanship emphasizes team empowerment and responsibility as vectors for getting software built right: care enough about what you do to take time to learn how to do it well. Then do it well.

We can draw off many experiences from the craftsmanship movement, and frankly we shouldn't be trying to generate our own similar movement. Automation is a software engineering effort, so let's just hitch up to the craftsmanship movement and add our voices and experiences in there!

6 Getting Started: A Practical Example

I can look to my own history with automation for guideposts to a successful implementation. (I'll save you the war stories of my various unsuccessful efforts. Yes, those have happened.)

First off, set the tone with the entire team, from stakeholders to your operations support folks. Getting going with automation requires acknowledging up front there will be a learning curve. It also requires acknowledging there will need to be involvement from both testers and developers, regardless of the toolset being used. This is especially true for tools like Selenium IDE or commercial tools seeming to hold the promise of "codeless" solutions for automation efforts.

Once you've got those expectations correctly set, begin with small steps. Have your team identify a few high value test cases to target for automation. Your whole team needs to be involved in this discussion as you select cases and decompose them. Stakeholders should have a say in the highest priority test cases—they've the best exposure in to what the business values. Testers, especially the manual ones, will have great ideas on how to flesh out simplistic cases in to realistic exercises designed to avoid regressions. Finally, developers will know what value they can add by updating the UI where needed, or by creating backing APIs to leverage existing web service or database calls to handle setup and teardown steps.

Work hard to have your developers and testers create tests via paired programming. Both will have great ideas on how to get the best test cases built in the quickest, most maintainable fashion. Having developers assist in this effort can help ensure the cases follow good development practices (granularity, avoiding duplication, etc.). Getting the testers writing tests gets better coverage with more robust cases covering sad paths, odd validations, etc.

Make your automation part of each feature you're working on. "Done Done" shouldn't be "Done" until you have working UI automation cases for it. Don't release new features to production unless you have test automation as part of that work. This isn't an easy thing to do—you'll have to get the entire team to commit to this, and it's extraordinarily easy to let this commitment slip when things get rough.

Above all, you'll need to empower the team with a powerful, passionate mindset for continual improvement of the test suite. We need to look at our test suites just like we do our production codebases: they're best when we constantly pay off technical debt, refactor out poor approaches, and occasionally completely re-architect the solution if needed.

Successful automation projects always look at the test suite just like production code—***Because it is!***

7 Conclusion

Are we failing with UI automation? As I pointed out in the opening, it depends. We have tremendous failures, we have tremendous wins. Successful UI automation is within the reach of every software team. It requires discipline, training, and constant refinement. It also requires the whole team to focus on what the most important aspects of that automation effort are.

If you're involved in even modestly successful automation efforts, share your views. Present at conferences, write articles for magazines, or start a blog. Don't be intimidated by what you think are meager accomplishments—other teams out there are struggling through the same issues you've overcome. They're looking for information on others who've gotten through those struggles. Too often successful teams in our industry forget the number of people still struggling through issues they've solved months or years ago.

If you're looking to start working on UI automation, then get involved by monitoring automation mailing lists like the Selenium user (http://groups.google.com/group/selenium-users) or developer (http://groups.google.com/group/selenium-developers) lists. Grab Jeff Morgan's page-object gem from Github (https://github.com/cheezy/page-object) and start exploring it.

All off this requires us to keep a very patient, long-haul view. It's taken decades for the software development side of our industry to work up to its current state, and I don't think anyone would call it perfect. Instead, we need to acknowledge there's a lot of work to do. Let's be about it!

References

Haynes, D. 2001. “Automated Testing: A Silver Bullet,” *Rational Edge*, June, 2001

Hunt, A. and Thomas, D, 1999. *The Pragmatic Programmer: From Journeyman to Master*. Pragmatic Press, 1999.

McConnel, S. 2004. *Code Complete: A Practical Handbook of Software Construction* 2nd edition. Microsoft Press, 2004.

Morgan, J. Page-object Gem, https://github.com/cheezy/page-object.

Osherove, R. 2009. *The Art of Unit Testing: With Examples in .NET.* Manning.

Selenium Developers Google Group, http://groups.google.com/group/selenium-developers.

Selenium Users Google Group, http://groups.google.com/group/selenium-users.