Agile Test Automation: Transition Challenges and Ways to Overcome Them

Venkat Moncompu

venkataraman.moncompu@cognizant.com

Abstract

Agile development is now becoming mainstream, recent Gartner research highlighted last year 21% of development projects were based on agile methodologies. Though the adoption of agile is burgeoning, the fundamental challenges in adopting agile testing methodology remain. Nuances in Agile transition at an organization and process level are constraining the desired business outcomes.

A survey conducted by a leading tools provider mentions the two major resistances to agile adoption are lack of management support (27%) and company culture (26%). One of the prime reasons of failure at a project level is the inability to integrate test automation in an agile lifecycle. The traditional practice of automation towards the end needs a paradigm shift to an integrated and incremental approach across the lifecycle.

In this session, I will share my experience with teams making the agile transition and talk about technical and non-technical challenges at different organization levels. Discuss the Agile based rethink approach for time-to-market, cost reduction and increasing ROI. I will also share techniques around transparency and project visibility, waste elimination, technical debt management, continuous feedback, just-in-time decision-making for test automation, and what worked well and some that didn't.

I conclude with how these were successfully applied to transforming the test automation practice in some organizations of different sizes, levels of maturity and cultures.

Author: Venkat is an agile testing evangelist and enthusiast with 16 years of industry experience. Over the past few years, he has worked with teams that have mastered the agile software delivery as well as teams that have struggled through the painful process of adoption until abandoning agile development methodology altogether.

Currently, Venkat is a director in the automated testing practice within Cognizant Technology Solutions' test Center-of-excellence. He has a master's degree in engineering from Arizona State University and has participated in technology and quality conferences as a presenter and speaker - Software Quality Days 2013, PNSQC 2012, Rational VoiCE Conference 2011. He has also published in the Software Test & Performance magazine and Stickyminds in 2008.

1. Introduction

Organizational changes are some of the hardest to come by. Deep rooted culture, habituated practices and a deep sense of resistance to changes are some of the biggest obstacles to overcome. In the context of agile software development, these are amplified as the people part of the equation of people, process and technology simply lack proper guidance in their adoption to begin with and the perseverance to make the transition.

Given the significant advances in technology to enable agile development practices, their contexts of use are sometimes lost in the ways teams have adopted them in their projects. Prior to the advent and popularity of agile practices, engineering heavy practices such as architectural milestones were emphasized, accentuating the waterfall type of staged development with rich ceremonies of review and base lining. The processes were also supportive of the command and control leadership and encouraged a rich set of metrics that were rarely predictive of team dynamics and goals of the project.

In his book the agile survival guide, Michael Sahota talks about the ills that are plaguing the agile software development movement – change agents talk about adopting agile rather than transforming the culture of the organization to an agile mindset (Sahota, 2012). Adoption might be easier in large organizations, but transforming cultures to scale is a significant challenge altogether.

In the following sections, I will share some of my experiences as a tester and quality professional. I have faced challenges as a member making the transition to agile methodology and in trying to be an agent of change in larger organizations. I will also narrate some of the cultural challenges to self-organizing teams that span cultural boundaries as off-shore teams attempted these transformations.

2. Challenges of Engaging the Business

One of the benefits of agile principles is better alignment with business and engaging business throughout the development period. The adoption of continuous integration practices using unit tests provide guidance to the development and automated acceptance tests give an indication of 'Doneness' for the stories under development. The difficulty of customers and business to engage with development is that customers find it difficult to find time in joint application development (JAD) to actively engage with the teams that helps development teams to identify data or control flow through the application.

In some of the agile projects I have been involved with, we addressed this challenge by describing the specification as behavior itself. To the extent we could use predicates to describe outcomes and behavior; we succeeded in deriving acceptance criteria that we could validate with the customer. However, program-defined behavior specifications are particularly ill-suited for use in designing software systems (Wileden, et al., 1983). The problem with behavior is that describing them at the structural layer differs greatly with the functional layer. Practice of acceptance test driven development (ATDD) is a good way to design these tests and engage the customers. Studies have shown that there can still be significant gaps between the test coverage and code coverage for such automated tests. Robert Glass writes in facts and fallacies of software engineering that even when developers think they have "fully" tested their code, only 55-60% of the logic has been tested (Glass, 2002). We have also experienced defect leakage past these tests attributable to incomplete data flow coverage. We have tried to address this challenge by adopting boundary value decomposition and path analysis techniques (McCabe, 1982). We have been able to qualitatively show that with such techniques, we can increase confidence in early

automated testing and the value therefrom. We captured benchmarks and baseline metrics of complexity with coverage using some of the commonly used metrics. By correlating the number of tests identified against complexity measures, we were able to show a corresponding increase in coverage through the increasing number of tests with higher complexity. When we tied the higher coverage with the number of defects, we could not necessarily get a quantifiable measure of effectiveness but could demonstrate subjectively the *defect prevention* had improved. This has addressed the concerns regarding return-on-investment (ROI) which we are often faced with to demonstrate quantitatively on the benefits of these behavior-acceptance tests to the customer.

3. Challenges Faced By Traditional Test Automation

When teams transition to agile testing, the traditional testers who are used to the late testing mind-set have to think very differently about the value provided of early testing on partially complete code. The automated testers have to develop tests behind the GUI which they are not used to. It calls for programming skills, tools knowledge and understanding of the solution architecture – at least to the extent that they can relate objects and behavior. This transition is very profound and the learning curve fairly steep but not insurmountable.

Building the competency, skillsets and enabling the tester is one aspect, the bigger challenge is the shift in mindset of the tester to think about continuous value realization during the sprints of an agile cycle. Moving the teams from functioning in siloes to highly collaborative development involves overcoming a high level of distrust and antipathy toward the QA teams. Additionally, in some of the organizations I was involved in, we resorted to coaching, facilitation of team games to build collaboration and cross-functional training to build self-organizing teams.

The notion that automated unit tests and acceptance tests can serve as a safety net for the team to maintain the cadence is significant in that testers are able to directly relate to the value they could provide. In my view, this form of empowerment of the testers and the value that developers realized through continuous integration was instrumental in bringing the teams together. Complemented with exploratory manual tests, the coverage increased and the automated tests could be adapted throughout the lifecycle.

We adopted a script-less approach for the GUI-based regression tests. Our approach was tool-agnostic and relied on object oriented techniques such as reflection and dependency injection. Object oriented languages allow instances to dynamically inspect code in the same system referred to as reflection. Using this technique, users can extend the framework by defining custom functions that can be invoked for handling those object interactions that are not supported natively by the web driver. And with dependency injection, objects can be injected to a class to instantiate it rather than relying on the class to instantiate it. The approach addresses issues of *technical debt* of automated scripts that accrues with tighter timelines and quicker turn-around as tests scale and story points grow, reduces waste by self-documenting tests instead of *apriori* documented scripts by leveraging the action-keyword implementation of test scenarios and just-in-time automated test execution through late-binding. This approach also fostered collaboration between the user experience (UX) and testing teams as this approach relied on communication of the UX design through wireframes and user navigation to develop automated test scenarios early in the development sprint.

The script-less approach also enabled teams to realize benefits such as accepting late changes, lower maintenance costs, concurrent automated test design with code development and quicker turnaround. This also ensures that test automation effort progresses in-line with iterations.

The other challenge GUI test automation teams face with automated testing tools is that of object recognition and the extent of identification support by different accessors. By engaging early, the automation teams can work to resolve them while the code is being developed and get the help of developers to include 'testability' into their design and implementation, which addresses these problems. The benefits observed as shown in figure 1 is based on an audit application project that had over 6500 regression test cases (or steps) automated by a team of 7 spanning 3 months over 1 major release and 2 minor releases, the duration of the project release being 1 year and 3 months. With over 80 business user stories that were automated, the maintenance effort of the test assets was less than two percent (2%) of the investment (fixed cost + variable cost) to keep up with the product features. The highly readable automation test scripts also serve to communicate with cross-functional teams very effectively.

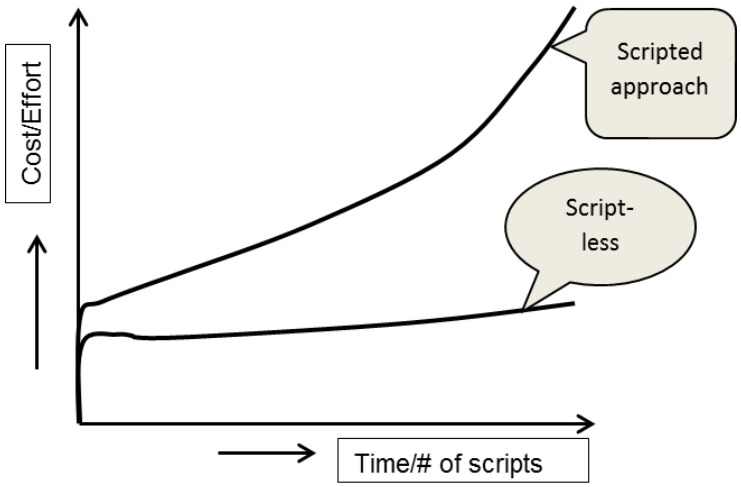

Figure 1: Cost savings from script-less compared to scripted test automation

Agile testing looks to test automation to providing insights and feedback on application quality as the iterations progress. The classifications of tests where automation can provide this feedback to the different stakeholders are best captured in the agile testing quadrants described by Brian Marick (Crispin & Gregory, 2009). These are reproduced here for reference in the figure 2 below. These test approaches need some discussion here: BDD or behavior driven development tests are those that describe the behavior of systems and is used to guide design and development. This behavior of system can be specified in a domain specific language called Gherkin that lets you describe software's behavior without detailing how the behavior is implemented. A Gherkin DSL example follows in the subsequent section; however the same can be used to describe acceptance criteria for product backlog item or user story. When this is done to verify *doneness*, it is called Acceptance Test Driven Development (ATDD). Both BDD and ATDD are primarily business facing tests and typically automated. While these tests leverage a specification that is at a higher-level of system abstraction, the unit tests written prior to the writing of the code facilitates test driven development (TDD). At the end of the day, all of these tests provide the safety-net for developers to focus on managing technical debt through the red-green-refactor cycle.

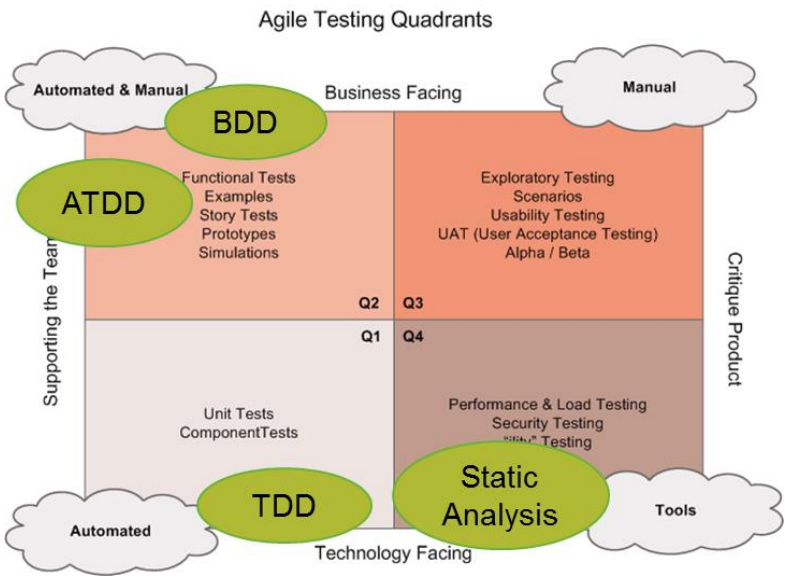

Figure 2: Brian Marick's Agile Testing Quadrants (Reproduced with permission, Courtesy: Crispin & Gregory, 2009)

The quadrants can be used to articulate the value proposition from automated testing in agile projects. The significant difference between the linear development cycle and iterative cyclical agile testing is that the testers have the opportunity to continuously adapt tests in line with the principle of welcoming changes late in the development process. Customers are often heard retorting, “that’s what I asked for, but that’s not what I need.’ Without continuously validating the builds, a lot of waste (rework) ensues. Teams have to look beyond GUI testing to build-in quality and get early validation toward a converging solution. I often use the quadrants overlaid with the techniques available to the testers to highlight the insights these provide and the respective groups that benefit therefrom. In order to ensure continuous test adaptation, I encourage teams to combine different approaches to review and balance the automated tests with their manual counterparts. This helps in capturing the metrics such as reduction in defect leakage, defect detection by sprint normalized to the story points realized. Through high visible information radiators, teams have achieved seen greater participation by the stakeholders. These metrics in addition to the commonly used ones such as burn-down charts, burn-up rates and velocities is very helpful. The steps to create these highly visible information dashboards are straight-forward – determine the defect detected by user story, then using the story estimation point as weights to compute the weighted average. This ensures that the normalized defect detection is also relative to the team just as estimates and velocities of different teams also vary for the same set of stories and seen in the context of the team. A couple of the metrics shown in the information dashboard are shown in figure 3 below.

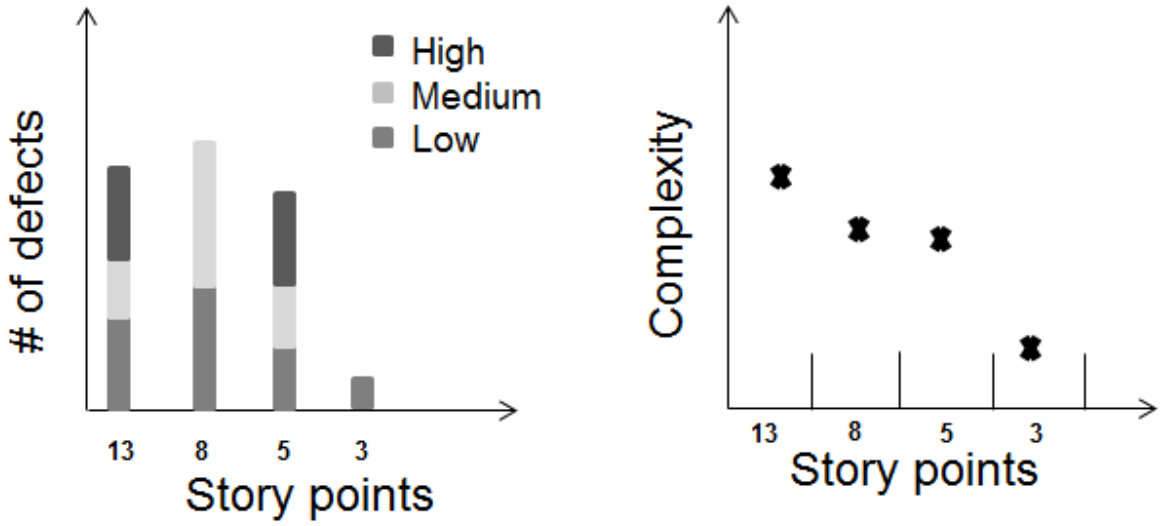

Figure 3: Some Measures Tracked & Presented in the Sprint Dashboard

Note that the complexity referenced here is one of the commonly used measures one of which is the cyclomatic complexity but for the purposes of this discussion this could well be replaced by another measure without losing the point.

4. Technological Advances and Innovative Practices

Technological advances in the recent decade in the areas of continuous integration and delivery have encouraged teams to realize the value from completed iterations. This fundamental change in approach also increased collaboration among the teams as the automated tests quickened the feedback cycles and increased the opportunities to *build-in* quality.

In and of-itself, testing teams face challenges of re-tooling and building skills of software development for designing *adaptive* automated tests – reusable (Hunt, 1999), extensible, maintainable and receptive to late changes (Beck, 2000). With script-less GUI-automation, most of these challenges are addressed implicitly as the development of action keyword processing engine is a one-time activity and skills of the development team can be leveraged to design, develop and test the script-less test execution engine. The action keyword processor is the part of the engine that processes the interaction and navigation flows of the application whereas the script-less test execution engine provides other capabilities as scheduler service, threaded parallel execution capability and also self-executable test script that can be reused by cross-functional team. The low-level action keyword sequence would look something like the following as shown in table 1 – typically they follow a page-object model. A higher level sequence of these low-level tests can be grouped by logical business function and managed by the script-less execution engine as shown in table 2. This also gives the flexibility of creating self-documenting tests and increases re-usability. The low-level action keyword processor is an open-closed solution in that it is open to extension but closed to modification (Martin, 1996).

Object	Action	Parameter	Execute
Browser	Launch	IE	Y
URL	Load	https://abc.company.com/login.asp	Y
Login_Field	isVisible		Y
Login_Field	SetText	“user_1”	Y
Password_Field	SetEncryptedText	“Ah#i@0JWb&”	Y
Submit_Button	isEnabled		Y
Submit_Button	PerformClick		Y

Table 1: Low-level action keyword sequence for a login function

Script	Execute	Argument
Login	Y	<test data filename>
NavigateToAccountsSummaryPage	Y	<test data filename>
PerformBillPay	Y	<test data filename>
Logout	Y	

Table 2: High level execution sequence

With the development team involved in the design and development of the keyword-driven script-less framework, the development team begins to ‘own’ and support the product. The developers could also run these regression scripts as some of the agile projects have reported doing it (Ambler, 2010). Lately, tools support Domain Specific Language (e.g., Gherkin) for test definition breaking down the barriers of niche-skills and alien programming languages that traditional commercial tools brought with them (Pettichord, 2000). With a language like Gherkin, you can specify the behavior in unambiguous terms and develop code that can verify in the language of implementation. For example, with Cucumber which is in plain text, the test harness can be generated to verify the feature and steps within the test as shown in table 3.

Feature: Overdraft amount withdrawal exception management
If a user attempts to withdraw funds that exceeds account balance to throw an error that indicates insufficient balance unless the user has an over-draft account

Scenario: Insufficient account balance exception handling during withdrawal attempt
Given the user is a regular account holder is logged in
And has selected the withdraw amount option
When the user requests an amount exceeding his current balance
Then the system should inform the user that the current balance is insufficient to honor the request

Scenario: Over-draft account withdrawal subject to an upper limit
Given the user has logged into his margin based account
When the user has selected to withdraw an amount exceeding his current account balance
Then the user should be allowed a one-time over-draft withdrawal subject to an upper limit

Table 3: Example of a behavior description using Gherkin DSL

The behavior definition as described in table 3 translates to a test runner in the specific implementation language.

5. Cultural Challenges and Agile Adoption

Agile estimation is very inclusive and iterative as opposed to the traditional top-down command-and-control style that traditional development lifecycle entails. This is another challenge faced by teams making the transition to agile. From the cultural perspective, outsourced quality assurance engagements

involving external teams find it difficult to break-free from the 'being-told-what-to-do' frame-of-mind to a highly engaging, self-organizing and collaborative participation in the development process. Some of these traditional off-shore teams find moving from the top-down estimation which they are habituated to give absolute numbers for a set of requirements thrown 'over the wall' to an iterative and relative sizing that agile practice recommends difficult. And the myth that agile estimates are absolute has to be broken. That different teams can weigh same stories differently is perfectly acceptable in agile estimation (Kelly, 2013). The concept of continuous learning and retrospectives is also an alien practice to traditional off-shore teams who rarely get a place in the table to provide inputs that materially affect the practice of application development. Teams that are beginning the agile transition journey may err on the side of caution by estimating more than the actual efforts needed. This is ok and over time as the team gets into a cadence; the team is able to learn from the sprint velocities, getting better at estimations.

In some of the cultures, it is believed that a good process can make a great product as long as you can demonstrate rigid adherence to the process, the outcome will turn out to be good. Unfortunately, with software development this is far from the truth (Hartman, 2010). Agile methodology describes *principles* when followed through *practices* that are contextual result in addressing most of the issues that stem from other software development methods. It is to be realized that Agile is a culture rather than a software development process. This acknowledgement for some of the traditional teams is hard to come-by. In highly metrics-driven leadership cultures, this can be a chicken-and-egg problem. Without adoption, you cannot show the statistics and without statistics, you can't get the traction and support for the agile transition. In organizations that are run by metrics and measures, the teams inherently imbibe the culture of process efficiency at the expense of the product.

I have resorted to building the mind-share within the organization, in such situations, by constantly deliberating and giving my points-of-view backed by industry reports, thought leadership articles and references to business values and outcomes for agile adoption and transition.

6. Team Velocity and Cadence

A common challenge that teams face in the early formative stages of agile adoption is in finding the right cadence. This is driven by the urge to start working on newer tasks before completing the ones committed to for the current sprint. Teams often lack this discipline of starting many tasks and activities without committing to complete them. While inability to complete the ones started is not unusual for even the more mature agile teams, the mature teams realize the importance of completing them before starting to work on newer ones. The Kanban approach to limiting the work-in-progress and driving to completion the stories that are started is the most effective way to manage the velocity of teams. *Kanban* (Japanese) literally means a signboard that was first used in the Toyota Production System by Taiichi Ohno to control production between processes and just-in-time manufacturing. They were effectively used to minimize work-in-progress and reduce cost related with holding inventory (Gross & McInnis, 2003). Even today Toyota continues to use the system not only to manage cost and flow, but also to identify impediments to flow and opportunities for continuous improvement

7. Conclusion

In this paper, I have tried to articulate some of the major challenges to agile transition with a propensity to agile test automation. This challenge for test automation teams are to think of the value automated testing effort brings to business and development teams continuously look to optimize the test coverage by

vectoring the automated tests and balancing the manual exploratory and automated tests. I have also touched upon some of the key challenges such as shifting from a command-and-control team structure to a self-organizing style of functioning, coming to terms with relative sizing and estimation, limiting work-in-progress to focus on completion of user stories, value realization through working software etc. that I have faced in working with teams that have struggled with the transition.

References

1. Ambler, Scott, 2010. *How agile are you? Survey,* www.ambysoft.com/surveys.
2. Beck, Kent, 2000. *Extreme Programming Explained: Embrace Change*, Addison-Wesley.
3. Crispin, Lisa & Gregory, Janet, 2009, *Agile Testing: A Practical Guide For Testers and Agile Teams*, Addison-Wesley.
4. Glass, Robert, 2002. *Facts & Fallacies of software engineering*, Addison-Wesley.
5. Gross, John M. & McInnis, Kenneth R. 2003, *Kanban Made Simple*, AMACOM books.
6. Hartman, Bob, 2010, *Doing Agile Isn't the Same as Being Agile*, PHXSUG presentation.
7. Hunt, Andrew, 1999. The Pragmatic Programmer, Addison-Wesley.
8. Kelly, Allan, 2013. *Top Twelve Myths of Agile Development*, Agile Connection – online community, www.agileconnection.com/print/article/top-twelve-myths-agile-development.
9. Kniberg, Henrik, 2010, *Kanban vs. Scrum – making the most of both*, lulu.com.
10. Martin, Robert J., 1996, http://www.objectmentor.com/resources/articles/ocp.pdf
11. McCabe, Thomas J., 1982. *Structured Testing: A Software Testing Methodology Using the Cyclomatic complexity Metric*, NBS Special Publication, National Bureau of Standards.
12. Pettichord, Brett, Jan/Feb 2000. *Testers and Developers Think Differently*, STQE Magazine.
13. Sahota, Michael, 2012. *An Agile Adoption and Transformation Survival Guide: Working with Organizational Culture*, lulu.com.
14. Wileden, Jack C, et al., 1983. *Behavior specification in a software design system*, The Journal of Systems and software, 3, 123 – 135, Elsevier Science Publishing Co., Inc.