Putting Agile to the Test:

A Case Study for Test Agility on a Large IT Project

Abstract |

Agile best practices can be applied to a variety of situations. Most prevalent is IT projects, often for software development. But can you apply agile to a 'portion' of a software implementation project? This paper will provide an overview of a successful example where agile techniques were applied to a multi-year, multi-million dollar IT program – specifically to the testing phase. The following topics will be explored:

- Scenario (what was the scope?)
- Challenges (how did we get here?)
- Approach (why was agile chosen? what was used to monitor progress?)
- Benefits (what did we do right?)
- Lessons learned (what could have been done differently?)

After examining this case study, you will have a greater appreciation for the testing complexity of a large-scale software implementation as well as techniques that can be applied quickly to your current project.

Key words | Enter three to five key words for your paper here.

Agile, Scrum, Iterative, Testing, Program

Scenario

After multiple years of planning and development, a regional insurance carrier's program to replace the existing system for processing claims was falling behind schedule. Unit testing of individual modules was reasonably effective, at least enough to gain confidence to proceed with full-scale integration and system testing. However, there was limited ability to verify interconnectivity of the components, let alone integrations between the software package and third-party data providers.

For months, multiple test leads worked to translate volumes of requirements into business scenarios, which were actually complex use cases meant to exercise features, functionality, and screens within the application. These business scenarios spanned across more than a dozen insurance products. The test leads were in the process of decomposing the end-to-end scenarios into thousands of test cases to validate all functionality and integrations.

Challenges

The challenges were many. At the original planned testing start date, development was not complete for all expected deliverables or for the third-party vendor data integrations. An additional constraint with the vendors was that most did not have a dedicated test environment and could only expose one external entry point into their system. These environmental constraints compounded the coordination for testing and threatened to elongate the schedule.

Internally there were multiple test environments, each configured slightly different, as dictated by the sub-team that was using it. A new integrated test environment, created exclusively for System

Originally published as part of the PMI® Global Congress 2016—North America proceedings.

PMI Global Congress 2016 North America

Integration Testing (SIT), was not fully functional or stable.

Business requirements – and therefore system (or technical) requirements – changed throughout the development and testing cycle. Since the central system was package-based, the decision was made early on not to document 'out-of-the-box' requirements; only customizations and changes to base functionality were documented in the requirement set. This made traceability from requirements to test cases cost-prohibitive (effort-driven). Although thousands of test cases were written, this covered only a portion of the full functionality. Test cases continued to be written and updated throughout the entire project life-cycle.

Testing was a core competency within the IT department, mostly internal employees with functional, performance, and automated testing skillsets. Purposefully, the project testing team was composed of IT and business resources. The key challenge was that the IT testers were expert testers (mindset and tool skillset) but knew very little about the business process. Business staff were allocated to assist with testing; this was great from a subject matter standpoint, but the individuals had no testing background and did not think like testers.

Communicating testing progress was a challenge from the start. Although the testing team maintained a standard format for delivering test status, it contained too much detail and was not appropriate for management level updates. Additionally, there was no established cadence for updates, leading to wasted time and inconsistent messages – mainly delivered in ad-hoc conversations.

The last, and certainly not the least, challenge was schedule-related. At this point in the project, the planned testing start date had been missed and the expected launch date was less than nine months away. This was a critical transformation for the company with Board-level visibility.

Approach

Given these challenges, what would you do? A project manager (me!) was brought onboard, specifically to organize and manage the system integration testing (SIT) phase.

Although the entire program was not executed with an agile methodology, agile principles were applied to complete the testing phase:

- ***Customer satisfaction*** was the highest priority.
- ***Adapting to changing requirements*** throughout the testing phase drove success.
- ***Working software was deployed regularly*** (almost daily) for business validation.
- Developers, testers, and business sat and ***worked together during the entire testing phase***.
- Testing team was ***motivated and empowered to deliver***.
- Majority of the ***testing team was co-located*** in the 'SITuation Room.'
- ***Progress was measured by delivering software*** free of critical defects.
- Team ***sustained constant pace*** for the duration of the SIT phase (work hard, play hard).
- Business scenarios and test cases were prioritized. ***Lower priority work was not done*** until later.
- Testing team ***reviewed and re-grouped at the end of each iteration to adjust*** to changes.

A three-pronged approach was devised: optimize the environment, organize the team, and orchestrate the work.

Optimize the Environment

In order to control the chaos of the test environments, a detailed spreadsheet was created to track software versions, service components available, database connections, and connectivity to third-party data integrations. Due to cost constraints, the environments could not be made identical, but standardization was achieved as much

Originally published as part of the PMI® Global Congress 2016—North America proceedings.

as practical; variations were tracked. Dialogue with each of the external vendors confirmed available test environment connections. The decision was made to point our project-level test system to the vendors' test system for initial unit & system testing. Once this was certified the link would be moved to point our SIT test environment to the vendors' system – and stay that way. This ensured testing moved into the SIT phase only when the integration was first proven and drove overall stability of the SIT environment. It did hamper testing in the project test environment(s), but it was a worthwhile trade-off that was made to increase stability and guarantee full-functionality for SIT.

New code was integrated into the environment regularly – typically multiple times per day in the project test environments and daily in the SIT environment. The release management function was strengthened, swelling to three FTEs at one point, to oversee the code movement process and control all of the test environments: pre-production, SIT, performance, training, and three parallel test environments. (Developer sand-box environments and production were not included.)

Organize the Team

The IT testing team was divided amongst the projects with at least one test lead and one test analyst. Initially they were assigned to the various projects to learn the system and validate test scenarios until the SIT test phase began. Team members were assigned to project testing teams, and eventually the SIT test team, based on current knowledge, the ability to learn, and skills growth over the duration of the project. For example, IT testers, with strong testing skills translated requirements and wrote test cases, while those with strong communication skills mentored the business users who were assigned to testing.

Business staff were extricated from their day jobs and allocated full-time to the project. It is important to note that these 'business testers' were hand-picked for their critical thinking skills, adaptability to change, and willingness to learn. Business knowledge (related to specific deliverables) and critical thinking skills determined placement on particular projects.

The SIT testing team was co-located in an open environment, affectionately known as the 'SITuation Room,' to facilitate collaboration. Knowledge sharing between the IT and business testers was a critical success factor – it fostered open communication so that the IT testers learned the business process and the business testers learned how to test.

Although the SIT test lead provided guidance to the business testers, the testing team was self-organizing – test scripts were self-selected to be 'checked out' for editing and executing each day. In order to facilitate cross-functional collaboration, a daily stand-up meeting was held each morning attended by the test leads, the project manager, and other interested (but silent) stakeholders. This set the tone for the day and was a great method to share key information. At the height of the project, the release manager attended daily to provide environment status and work with the test leads to schedule code builds and any other interruptions. And yes, we did occasionally sit during the stand-up meetings!

The project manager cultivated strong relationships with the program leadership, functional managers, development project managers, release management, and the testing team members. Since he was truly functioning as a scrum master, this was crucial for removing obstacles, insulating the team from management, and escalation of critical issues when necessary. For the duration of the testing effort, the testing team maintained an elevated, but realistic pace. In addition to working hard, they made time to 'play hard' with afternoon snack breaks, milestone celebrations, after-hours outings, and even going to exercise together. (These were all optional but well attended.) Team members were also recognized and rewarded using an internal HR-driven awards program – presented at monthly 'all-hands' program meetings – and a large celebration event at the conclusion of the first set of releases.

Originally published as part of the PMI® Global Congress 2016—North America proceedings.

Orchestrate the Work

Since not all functionality was ready to test when the system integration testing phase was planned to start, the key use cases were prioritized along with the most important products to release. Following practices from iterative software development and Scrum, the SIT testing effort was aligned into four iterations, where products were grouped logically for testing and phased for rollout after the third iteration. The fourth iteration was used for lesser important products, features, and defect clean-up. Each iteration was tracked as a separate 'release' – even though functionality would not be released externally.

Business requirements fluctuated as more was learned about the functionality of the out-of the-box system, necessitating real-time adjustments to the test cases that composed the business test scenarios (use cases). Much of this work fell to the SIT test lead so that the other testers could continue to make progress. Regular touchpoints were scheduled with the business analysts to communicate updates and adapt testing based on the changes.

In addition to the daily stand-up, the product owner facilitated daily defect triage meetings where defects, enhancement requests, and any other changes could be discussed and prioritized. This cross-functional meeting was attended by program leadership, the project manager, the SIT test lead, the lead business analyst, and as needed, development project managers, business analysts, and test leads. Each day, a 'Top 40' list of defects was published with the goal of remediation within one business day. At the end of each iteration, this meeting became more important as decisions were made about what scope remained and what was pushed to a subsequent iteration.

After each iteration, and sometimes at key milestone points, the team held a retrospective, so that lessons learned could be applied quickly for maximum benefit. Although this was primarily for the testing team, input from other areas was included to drive improvements more broadly across the organization.

Many agile artifacts were utilized to guide the testing team, monitor progress, and provide report-outs to program stakeholders: Program Roadmap, Release Plans, Iteration (sprint) Plans, Daily Stand-up, Product (test case) Backlog, Product (defect) Backlog, Burn-up Chart, Velocity Tracking, Burn-down Chart, and Review & Retrospective Meetings. Some examples will be shown during the presentation and can be shared with participants.

Results

The three-prong approach worked. Despite schedule slippage beyond original customer expectations, the prioritization of critical products first enabled benefit realization to begin more quickly which afforded the team the time needed to complete the remaining products and functionality. Stakeholders were satisfied, team member engagement was high, and the organization was better positioned for the next set of releases (2.x) and subsequent large-program initiatives.

Benefits

Stakeholders of the program recognized the benefits of utilizing an agile approach. Primarily, the customer appreciated our effort to deliver critical products as close to the original schedule as possible without sacrificing quality. (Therefore, additional time was granted to develop and test remaining products and deliver them in subsequent iterations.) Organizing the testing into prioritized iterations increased the involvement of business staff, built confidence with key stakeholders, and realized business benefit earlier than projected.

As expected, IT testers gained significant business knowledge regarding the business process while the business testers gained an appreciation for the amount of work behind the scenes to develop a new business solution. An unexpected benefit was that the best business testers were able to write test cases/scenarios in addition to execution, so much so

that some even applied for open test analyst positions with the IT department.

Often a long and fast-paced project can decline into a 'death march' where creativity and innovation are overtaken by repetition and blindly following 'orders' from the project manager. This did not occur, even though the testers were working longer than normal hours for an extended period of time. The team ownership that agile expects empowered the testing team to continually look for ways to improve the process, implement changes, gain efficiencies, and improve the final deliverable. The functional manager of the testing practice recognized the need to change and vocally supported the team, further advancing an agile approach to testing.

The streamlined agile structure made it easier to report on progress and escalate obstacles that arose during the project. Test leads owned the testing metrics and tracked status in ways that flooded into the status dashboard nearly automatically. Testing status was updated daily and was pushed to stakeholders on a weekly basis – prior to the weekly leadership meeting – which allowed for an informed discussion of progress and barriers. Obstacles were openly discussed by the team and owned by the scrum master; the leadership team was alerted and engaged as needed to accelerate remediation.

Lessons Learned

Using an agile approach highlighted the importance of mid-course correction – identifying and implementing minor modifications as quickly as practical to realize small gains that add up over time to large efficiencies. The team held end of iteration retrospectives and mid-iteration impromptu reflective workshops – both of which highlighted changes that could be made to improve throughput, increase quality, or simplify the work. Whenever possible, the change was corrected immediately; otherwise, process changes were made in the next iteration. Here are a few of the most noteworthy:

More consistent application of risk-based testing to prioritize most important work. One of the testing team's core practices was risk-based testing – a process of interviews of business SMEs and project team resources to identify the most important and most complex functions that should be tested earlier and with greater rigor. Although the various project teams applied the practice, it was not always done consistently, which led to quality issues when certain functionality was not exercised completely prior to moving into the SIT testing phase. It caused re-test late in the life-cycle. Once this was identified, the test leads worked collaboratively to ensure that the risk-based process was applied similarly and that results were reviewed by the SIT test lead as input into the SIT test planning.

Expanded usage of automated testing (grew from build testing to full scenario automation). During the initial testing phases, limited automation testing was done – mainly due to the instability of the code and test environments. Early automation included 'smoke' testing new builds as they were applied to the environment prior to manual testing. This was effective, but not overly productive. Once this was recognized midway through the test effort, a test engineer was brought on-board to review test scenarios, recommend automation opportunities, and implement. By the end of the SIT testing, nearly all of the 'Day 1' test scripts were automated and could be executed within about an hour and results were posted to a dashboard with drill-down capabilities. Very early in the second release, most of the 'Day 2' test scripts were added to the automation and status dashboard.

Prescribed order for type of testing: functional, then end-to-end, then business scenario. Early in the test execution phases, testers selected work randomly – based on his/her area of expertise (especially the business testers). This worked well for the individual testers to show progress, but it was not effective from a test management or coverage standpoint. As the testing effort evolved, it became more evident that there was greater efficiency to be gained by ordering the test

execution appropriately – progressing from stand-alone functional test cases to end-to-end test cases. The team learned that once these cases had (mostly) passed, then the full business scenarios could be executed without issues or restarts. Testers still selected their own work on a daily basis, doing so in a more organized fashion to optimize throughput. This significantly increased the confidence of the sponsor/stakeholder.

Consistency of test cases; more normalized size of test cases to effectively track progress. In the rush to translate requirements into test cases, each tester documented in his/her own style with varying degrees of detail in the steps and expected results. As long as the same tester executed the test cases, this was acceptable. However, when another tester tried to execute them, it led to confusion and frustration. Additionally, it was nearly impossible to accurately track progress and predict velocity due to the varying size (effort). The agile concept of story points was adapted to test planning – as an attempt to normalize the size of the test cases to be roughly equivalent. Over time, this did increase accuracy of planning for the execution effort and tracking progress of execution.

Daily cross-functional defect triage (separate from daily stand-up) provided instant perspective of the quality and stability of the total codebase and test environment. As the team began shifting from predominantly test planning to test execution, the daily stand-up was utilized to discuss actual progress (i.e. work done yesterday), planned progress (i.e. work to be done today), and blockers to progress – obstacles that were preventing test cases from completing (i.e. fatal / critical defects). Due to the magnitude of the effort, there were too many defects to be discussed in a 15-minute meeting AND the right people were not involved in the daily stand-up. This led to the creation of a daily cross-functional defect triage meeting. As discussed earlier in this paper, the defect triage meeting focused on scope control for each customer release. Separating the two allowed the daily test lead stand-up to remain focused on quality control and managing the test effort. Key issues were escalated from the daily stand-up into the defect triage meeting for visibility and key stakeholder assistance in remediation when required.

Conclusion

Agile techniques can be applied selectively to portions of a software implementation project to improve quality without sacrificing prioritized scope. Attendees of the presentation will gain a greater knowledge of agile best practices and increased confidence regarding the application of these principles to their current project. Sample artifacts will be reviewed and can be provided to attendees to jumpstart the learning process and benefit realization.

About the Author

My entire career has been with Westfield Insurance - as a developer, architect, testing leader, and Project Manager, having led dozens of business & technology projects with a proven track record of satisfied sponsors and team members.

As a member of the PMI Northeast Ohio chapter, I currently serve on the Operations Board, where I oversee a portfolio of projects focused on delivering increased member value. I am a current participant in the Leadership Institute Master Class (2016).

References

Agile Alliance. (2001). *The agile manifesto for agile software development*. Retrieved from: http://agilemanifesto.org/

Project Management Institute. (2013). *A guide to the project management body of knowledge (PMBOK® guide)* – Fifth edition. Newtown Square, PA: Author.

Stenbeck, J. (2013). *PMI-ACP and certified scrum professional exam prep and desk reference*. La Mesa, CA: GR8PM.