Abstract

A software solutions company out-sourced a ground-up application build to a team of globally distributed independent consultants. The project eventually exceeded time and cost expectations and failed to deliver production ready tested software according to a planned market release window. The asserted reasons for this planned versus actual delta varied with time including failure of leadership, failure of project oversight, to design and implementation flaws and so on. Out of the box Scrum was implemented by a newly contracted consultant and the project delivered 153% of plan by the end of four two-week sprints. This report discusses the problem, the implemented solution, and the results while challenging what really made the difference.

1. Introduction

Knowing it was time to evolve to new technology in terms of scalability, reliability, technology and platform agnosticism, as well as, increased functionality, and lower cost of acquisition and ownership as primary drivers, a software services company outsourced the design, generation and implementation of their future money-making software product with the hopes of having something new moved into production at a cost of less than two years.

As progress was communicated across time in terms of technological milestones, the customer/sponsor (funding the project) began to feel as if she did not really know what her money was buying, what was done and how much was left, and whether she should be making provisions to change over the company this year, next or thereafter. Converting all customers from the old product to the new product, with all of the associated customer management, marketing blitz, and sales and operational changes would be its own work of course; but understanding when investors would be done spending money and would begin making money back on the initial investments was a black box. As the project continued on, the customer increasingly felt as if she were in a helpless limbo — knowing full well she needed the new product now, but not able to get it; needing to shut down the old product line immediately thereafter; but not knowing when any of this would actually occur.

As time and money were expended, the project sponsor eventually chose to change out the leadership and structure of the project with the goals of a) gaining complete transparency into the project and product, b) preparing the internal operational teams to own and manage said product (versus the external team members), and c) gaining a clear definition of ‘done’ and production readiness associated to remaining money.

A third party consultant was brought into the mix, existing leadership moved on to other work, and teams were restructured to be multi-skilled, self-encapsulated, self-directing delivery teams. In essence, Scrum was implemented out-of-the-box within two weeks using user stories, acceptance criteria, a single corporate prioritized work repository, daily scrums, sprint planning and review meetings, inclusion of business proxies, and weekly transparency status meeting with all executive leadership, including those funding the project journey.

The revised project framework planned to deliver all identified and corporately prioritized software elements (user stories) in four consecutive two-week sprints. As measured by user stories and associated effort hours (as relative sizing and velocity association had not yet been implemented at this point), the new teams and structure enabled the delivery of 103% of expectations in the first three two-week sprints and ending the project with the teams delivering 153% of plan at the end of four sprints. Liking this process, transparency and progress to such an extent, the customer requested the project and scope to be extended with more funding so that this newly experienced tangible progress could be further practiced and the benefits further realized.
The secrets to this project’s success can be articulated as having three primary elements: a) out of the box Scrum, b) a conversion to user stories, and c) complete project status transparency frequently, often and all of the time.

2. Putting the Framework in Place

After the project sponsoring company concluded they needed to make a change, it was decided that a third party consultant should come in and take over the project conditional upon two objectives: a) at the end of this interim transition period, all work should be owned and driven by the internal house team, not the external group; and b) they needed a completely overhauled method of delivering software and it needed to be evidentially valuable and transparent – all to be achieved during the interim transition period.

The teams were trained up on a blend of delivery patterns built on Scrum project management [1], software development [2], and extreme programming [3]. This repeatable delivery pattern consisted of a) a sprint/iteration planning meeting to determine what work would be picked up in the sprint; b) a sprint/iteration demo purposed to show tested, working software to stakeholders at the end of each sprint; c) a sprint/iteration review meeting purposed with reviewing what went well, what didn’t, and how teams should improve for the next sprint; and d) daily scrum/stand-up meetings to verbally review progress, impediments and general group evolution.

A very important change to the environment occurred as a result of implementing user stories and acceptance criteria [4]. Used to define clear units of work and associated definitions of done that would be delivered and validated in a single two-week sprint, user stories were the difference between a team saying they were working hard, and a team delivering something that all stakeholders understood prior to work starting, and understood after work was delivered. After user stories were created, they were then prioritized by the corporation to identify which ones should be delivered and in what order according to business value. It is here, during this prioritization exercise, that the idea of blending business value priority with technical risk and complexity valuation was additionally introduced — and it is during this exercise it became apparent that prioritizing work on a software project required purposeful, multi-pronged focus — different than practiced historically. Then, with a clear definition of work, a clear definition of done, explicit testable statements and a clear priority for delivery filtered through business and technical lenses, the work queue came online in a matter of days.

Next, a single Scrum Master was designated to facilitate a constantly updated/constantly published clear plan of attack, as well as, to ensure all obstacles impeding or stopping progress were prevented from existing at all or removed altogether. And teams were reconstructed to be self-encapsulated and homogenized by including developers, testers and product stakeholders from the original third party team, as well as, developers, testers and product stakeholders from the internal corporate team.

A single backlog and work queue, single repository for defects, and a single, one-page, project status composed of planned versus actual sprint burn-down data were all made accessible to anyone directly and remotely involved in the project. Furthermore, a single weekly meeting was put in place with Executive Management and corporate defined stakeholders to review the plan and the actuals with brutal transparency using sprint burn-down charts. The results of these restructuring efforts yielded one method of doing work, one repository to store it, one method of defining and prioritizing work, and one method of communicating plans and status within and outside the project.

3. Executing to Plan

All of the work for this project was done when reconstructing the teams, putting in place a predictable, repeatable delivery pattern, having one backlog and one corporate priority, developing user stories and acceptance criteria as work unit and testability definitions, and implementing the weekly status meeting with executives was completed. After that, all the group really had to do was deliver tested, working software.

A baseline master user story list was created for the application, both existing and planned functionality. Said user stories were role based and included acceptance criteria defining ‘done’ which were leveraged as testability statements. As stories were created, they were prioritized by the corporation, relative sized by the development team, re-prioritized by the corporation, and then built out and picked up at sprint planning meetings. As this transition occurred at a hyper-quick velocity, purely moving from estimates by effort hours to estimates by relative sized story points occurred in parallel as the leap of logic proved too great amidst so much change. Sprints occurred in ten day iterations where Day 1 included the sprint planning meeting, all days started with a stand-up meeting, and Day 10 concluded with a demo of tested, working software and a review of activities to start performing, stop performing, or continue performing.
A clear definition of ‘start’ was accompanied by a clear definition of ‘done’. This pattern enabled clear visibility, clear statusing, and provided stakeholders and executive staff insight into what it might actually take to deliver working software.

A critical element in execution of this plan lay in having weekly meetings with executive staff to discuss sprint/iteration burn-down charts, as well as, informally meeting with everyone all of the time. Team members, stakeholders, executive staff and nearly anyone who showed interest in or had influence upon delivering working software to the benefit of the business and the customer base were all invited to ask questions, challenge priorities, become directly involved, or otherwise help eliminate opaque software development and delivery practices by becoming a unified company with unified priorities and methods of achieving them.

4. Biggest Hurdles

The two largest hurdles addressed during this experience included: a) stale, incomplete requirements composing the definition of done for the application’s usefulness; and b) getting the company to focus on the single most important priority for the application first, followed by the second most important priority for the application and so on.

4.1. Acquiring a clear definition of done

At the beginning of the project, requirements were drawn up in the form of uses cases with all the expected elements, most importantly including the pre- and post-conditions giving the teams a closed loop definition of done per requirement. Two challenges existed with these requirements: a) they were drawn up independently based upon 3rd party industry research and analysis without the direct benefit of anyone from the funding company’s teams who knew the market, the business, the problems, or the customer base first-hand; and b) as the project grew further behind and efforts to catch-up rewarded heroism over systematic behavior, the requirements themselves became non-versioned stale documents. Quite quickly, the use cases turned into history books clearly articulating the vision and direction at the time of the writing, but woefully out-dated, non-useful, and non-system reflective. Regardless the reasons or motivations, not only were the end-users and end-user service providers excluded from construction of the requirements, the evolution of a system solution soon out-paced the relevancy of having a set of requirements at all.

4.2. Achieving a clear corporate priority

At the beginning of the project, the direction to produce a new system consisted of two primary concepts as articulated by the outsourcing company: a) make the new system do everything the old system does, but better; and b) make sure the new product not only uses new technologies, but includes all the most current industry trended widgets and solutions. To exacerbate this two-prong directive, it was assumed, based upon the reputation and expertise of the outsourced team, that things would be delivered in some particular order that made sense for everyone involved. As no priority was provided by the company to the outsourced team, a little bit of everything and all of nothing was delivered during this period. Status meetings primarily consisted of technical team mates discussing how hard they were working on things they asserted the business needed - while executive staff wondered when they would see something tangible they could relate to and sell.

5. Addressing the Biggest Hurdles

5.1. A clear definition of done, acquired

The problems of knowing when software would be fully developed and delivered, when it would be validated and accepted, when the corporation would be done spending money to get the solution, and when it could then be marketed and installed in the marketplace problem first and foremost addressed by the implementation of user stories and acceptance criteria which provided clear start and stop boundary that everyone could understand. In particular, user stories were written up for all existing functionality in the system (admittedly in arrears), as well as, all pending/queued up functionality yet to come. These user stories were then expanded to include definitions of done in the form of testable statement acceptance criteria. The definition of done with regards to working on any particular idea was very surgically defined as: a) when the system performs user story ‘X’; and b) when all acceptance criteria associated to user story ‘X’ pass, this particular user story has then been implemented completely and ‘done’ has been achieved. The benefit of this approach became apparent when all people, executives, stakeholders, analysts and technical staff could understand the role-based user story and the role-based definition of done without fanfare.

Second to this, synthesizing or inter-mingling the team members from the external consulting staff with people from the company who knew the business, the challenges, the market and the customers proved a big
win after multiple design disagreements immediately revealed themselves. The immediate observation suggested that industry data used to design the system did not take into account current customer usage behaviors; and likewise, some current usage behaviors as recommended by the corporation and practiced by the customer base were inefficient and needed retraining. It became apparent the sum of teams was more powerful than the individual parts as originally designed.

5.2. A clear corporate priority, achieved

The problem of getting the corporation to define priorities, in order of importance, proved to be quite difficult. Of particular note so the reader is sure not to overlook this aspect of the changes, achieving an executable priority statement from the corporation would never have occurred had there not first existed user story and acceptance criteria statements that everyone could understand in plain, daily language. Since the external team had written the first version of requirements, had the requirements remained in use case format as originally constructed by the first team, momentum would have quickly been lost on simply making sure everyone in the newly homogenized teams even understand what use cases needed to be for this project, how they were used to deliver software to date, and what they meant for delivering software into the future. Had there been no effort to create a new baseline of functional requirements for the system built in user story format, this project overhaul would have failed at the outset due to perception, impression and interpretation issues surrounding requirements, software and customer need gaps. User stories and accepted criteria unequivocally provided a clear list of work upon which prioritization could occur thereafter.

Thereafter, the company recognized the need to have a single voice, rather than the panel of voices currently practiced. As a result, a product manager role was created within the company responsible for prioritizing what elements should be put into the software and in what order. One product backlog using one method of communication (user stories) gave way to the company giving single voice to prioritization and expectation setting. Stakeholders spent time reviewing user stories, discussing them, and providing their input to the product owner. Thereafter, the product owner provided a single, prioritized list of elements that should go into the product.

In reality, the product owner narrowed the list down to a number of elements that absolutely must be in the product on the first day. This was an immense improvement in collaborative development of the product solution. The remaining challenge lay in the fact that the product owner said all sub-100 elements were first priority and could not be further prioritized. Agreement was then achieved that software would be delivered by starting at the top of the list and working down until such time as this plan no longer made sense.

6. Results Review

Originally planned to be an interim overhaul project transitioning leadership from the external team to the internal staff, as well as, to completely change how software was being specified and delivered to the company, time went by very quickly. The project delivered 103% of plan in the first two ten-day sprints/iterations, and 153% of plan by the end of four ten-day sprints/iterations.

A third party consulting firm at risk of having their contract terminated delivered at and above expectations for every sprint/iteration thereby enabling them to have their contract extended with the sponsoring company indefinitely. Complete functional, tangible, tested software was demonstrated at the end of each sprint/iteration to anyone in the company with an interest. Weekly executive meetings shifted to discussing user story priorities and progress, not dissatisfaction with not understanding progress and value.

Independent consulting staff stayed (rather than abandoning), internal staff integrated and learned to lead and contribute, and internal staff took over leadership of the project.

7. References


