Capacity to Execute

Waterfall to Agile Transition Considerations - a Case Study

John Cvetko

John.Cvetko@TEKasc.com

Abstract

Transforming your enterprise to an Agile SDLC can be an arduous journey. Transitioning any large organization to a new state is a multifaceted problem and takes thought and planning to ensure the desired “intent” is achieved. The goal is to minimize disruptions to business operations and increases the probability of success. Whether the desired approach is incremental or a sea change, understanding the “capacity to execute” is crucial when evaluating the organization’s ability to achieve the desired goal. By assessing the capacity of the existing organization, it is possible to determine best how to plan strategically and systematically to move the organization to an Agile framework. In this paper we will discuss the assessment of an organization relative to the Agile principles, and how to develop a strategy that maximizes the Agility of the organization in the new paradigm. Using a case study, this paper will explore common issues with the transition to an Agile SDLC, and how, if not done with care and forethought, could result in significant risk for the organization.

Biography

As a Principal of TEK Associates, Mr. Cvetko works with companies and government agencies to improve their organizations by helping them manage the IT challenges they face. He applies state of the art solutions to evolve business processes, creating more efficiency and productivity, all while improving quality. His previous work as a CIO was enabled by a variety of program, system and product management positions held throughout his career. The last 12 years have been primarily focused on assessing and transforming large enterprise software systems for state governments. He has worked with the state governments of Colorado, Washington, Oregon, North Carolina, North Dakota, Mississippi, Utah, Kentucky, and Oklahoma. Earlier in his career he worked as a management/technical consultant for firms such as NIKE and Boeing, and in product development and program management for Tektronix, PGE/Enron and ASCOM.

1 Introduction

All too often, outside consulting firms or internal Agile champions position Agile to be a solution to deeply rooted organizational issues, or worse, as a philosophy that will bring dramatic flexibility to a rigid system. Overemphasizing Agile as the solution to all problems does a disservice to the organization being served, and erodes the Agile brand when expectations are not met.

The Agile SDLC has some unique qualities that can be useful for organizations to varying degrees; however, it should be thought of as a tool in a company's tool bag and scrutinized using the same return on investment rigor applied to other business initiatives. Developing a business case that outlines the purpose, business context, perceived limitations and culture is paramount to understand the problem being solved and establishing pragmatic expectations. The coupling of realistic expectations with rigorous business analysis, enables the leaders to envision and articulate their intent (a.k.a., the end-state). Understanding the end-state will then help to establish the key performance indicators (KPIs) needed to monitor the progress being made towards the business objective. By having this level of visibility, the leaders can help remove obstacles and course correct as needed to improve the chances of success. Using a thoughtful and purposeful approach to transformation will, in the end, create a more robust Agile implementation that will propel the business toward its objective. [Brosseau 2019].

This paper is structured in three distinct sections: The Case Study will provide information on an organization that is struggling to implement Agile. The Transition Considerations section will discuss fundamental elements important to organizations implementing Agile. The third section is the Transition Analysis, which will review the Case Study implementation relative to these considerations.

2 Case Study

The case study presented here is of a large IT enterprise in an organization that will be referenced as the XYZ company in this paper. The XYZ company has five main lines of business (Programs) that are dependent on each other. Each Program has complex workflows and many regulatory and compliance obligations. The enterprise has 6 feature releases a year that are delivered on a fixed schedule and with a fixed capacity, thus creating a "release train" [Leffingwell 2007]. Each release contains upwards of 30-40 intertwined Waterfall and Agile projects that ranged from 100 to 8K development hours and are from all Program areas.

The IT senior management structure has more of a facilitation function, connecting governance, Programs and IT executive management to IT functional managers and vendors. This approach could be considered a laissez faire management style, with a high degree of decision making done by consensus.

2.1 Shared Resources and Competing Priorities

The Program areas compete for development resources and often have their projects bumped from a planned release, due to a higher priority project. Often a lower priority project is rescheduled multiple times until the organization risks being out of federal or state compliance, which then elevates its priority by default. Once a project becomes a high priority, additional scope is added by the Program because of the uncertainty of future development resource availability. The constant scheduling challenges for shared resources and competing priorities create dysfunctional behaviors that are a continued source of friction between the Programs and IT.

To address the friction, the IT management recommended moving all projects in the releases to the Agile SDLC. This recommendation was touted as providing a faster, more flexible development approach that could provide higher quality at a lower cost, all while reducing risk for the organization. The underlying assumption was that the software development was inefficient and that by implementing an Agile SDLC, the Programs would increase their chances of getting a project done more quickly.

2.2 Stalled Agile Transition

At the time of this writing, the transition to Agile by the organization was in its third year. On initial implementation, the organization had conducted Agile training, employed coaches, identified tool replacements, and delivered Agile projects successfully alongside of the Waterfall projects in the same release. This moved the organization from a pure “Waterfall release train” to the “hybrid release train” mentioned above. The Agile projects were considered small in scope and isolated, i.e., not a critical path for other projects, free of dependencies and could be pulled from the release at any time. Conversely, the large Waterfall projects crossed many programs and could not be pulled from the release without a significant delay in the schedule.

The hybrid releases worked well initially, however the need for more flexibility was required. For example, a large Waterfall project had a few moderate changes that were added one month prior to the release date. These late changes significantly increased the risk of defects being introduced into production. The risk was mitigated through additional testing, which ultimately delayed the schedule. The delayed schedule did not affect the project with the last-minute changes; however, it did impact the delivery of a time sensitive, critical project in the same release.

The need for the late changes was legitimate, and the organization held a series of meetings to discuss available options to mitigate this issue in the future. The first option suggested was to move the larger projects to Agile. It was at this point the Program heads noticeably baulked at the idea and their general skepticism of Agile became known. It was clear the Programs had not realized the value they were promised and had developed a level of distrust of IT management. The expectations of Agile were set so high and in such a shallow way, that the skepticism and reluctance with using Agile for the larger projects is understandable.

The final solution implemented is known as the “soft requirements” approach. The idea being that 99% of the Waterfall requirements are correct, but occasionally the business needs the flexibility to make moderate changes to a project after the initial requirements cutoff date. To support this approach the process was changed to allow for two requirement cutoff dates; the first date was set at four months prior to the release (soft change cutoff), and the second date at one month prior to the release (final cutoff).

The “soft requirements” approach was a change of the Waterfall SDLC to compensate for the perceived deficiencies in the implementation of the Agile SDLC. This effectively stalled the implementation of Agile for the foreseeable future.

3 Transition Considerations

3.1 Business Case

It is often the case that when moving to Agile, several adjectives are used to sell the idea, e.g., Agile is more flexible, provides faster delivery, improves quality, etc. While these statements may be true, it would be foolhardy from a business perspective to base such a transition on adjectives alone, especially if your organization is not delivering well with its existing SDLC. If the enterprise is managed poorly using Waterfall, why would one believe that it would be any better with Agile? Why would Agile make bad management good? Sadly, it is not uncommon to see organizations make changes based on superlatives alone.

Agile consultants are sometimes inadvertently tarnishing the Agile brand by not addressing the underlying organizational issues first. It is not uncommon to hear management consultants touting the magic of Agile to fix” insert whatever the company issue is here”. For example, in the XYZ case study above, the IT management did not understand the cause of the Program’s dissatisfaction of the IT department pre-Agile. Albert Einstein once said, “If I were given one hour to save the planet, I would spend 59 minutes defining the problem and one minute resolving it.” Presenting Agile as a solution to an ill-defined problem may create unwanted second and third order effects in the organization.

3.2 Management Intent

All too often, Agile is implemented in organizations without an understanding of the business context, structure of the organization and culture [Ramesh 2018]. In post-Agile implementations it is common to hear staff complain that their company did not implement “true” Agile and they consider themselves “Agile-ish”, “Agile-fall” or worse, they have implemented “fake Agile”. This in part, can stem from a lack of appreciation by the staff of the business context limitations, i.e., highly regulated industries, unions, existing contract obligations, organizational risk tolerance, centralized management structure, etc. These conditions can be hard boundaries for implementing an Agile framework and can result in limiting the degree of flexibility desired.

Unlike Waterfall, Agile couples a set of prescriptive qualitative characteristics with quantitative development techniques. The Agile manifesto core values and the 12 principles are collectively an intent statement. The difference between a goal versus an intention is that a goal is focused on achieving a specific objective, whereas an intention is more concerned with the mindset and end-state [Mattis 2020 44-45].

Consider the following Agile principle: “The best architectures, requirements, and designs emerge from self-organizing teams” [Agile manifesto 2001]. The end-state of this principle is “the best architecture”. The mindset of “self-organizing teams” in this principle is vague and can be interpreted in a thousand different ways. The intent of this principle, if read by a line employee, functional manager, and executive staff will be interpreted based on the fears, desires and expectations that are top of mind in the context of their current environment. As an example, a line employee in a highly regulated industry with a strict management hierarchy, may interpret this as freedom from micromanagement and the ability to select the development team of his choice. However, management may interpret this principle in a much more conservative way. This interpretation gap can have the unintended consequence of increasing frustration and lower morale in the organization, i.e., it could do harm.

Taking time to examine the organization relative to the Agile principles will help identify potential issues and limitations of the implementation. This analysis will help articulate the contours of the Agile end-state and can be used as a feedback loop to test the level of decentralized decision making desired by executive management.

Once the end-state is defined, then developing a “management intent” statement is useful for broad communication to the organization. Using the language of the organization to communicate how Agile will fit into the environment can reduce ambiguity that can cause unnecessary cultural friction and frustration. An intent statement consists of three distinct elements; it describes the purpose, method, and end-state. It is intended to be a living statement that will help shape decision making of the staff when dealing with the obstacles they will address throughout the transition. This statement should also be tested periodically, this will be described in the next section.

3.3 Testing Management Intent

The testing of management intent can be thought of as a feedback loop for managers that is useful to course correct, adjust the tempo of the implementation, or coach teams when needed.

The scope and the modality of the testing can be done in a variety of ways. The testing approach depends on the leadership style and the tolerance of management for criticism. It is important to have in person, face to face discussions at a minimum, this reinforces and models a core tenant of Agile for the teams. Some managers prefer skip level interviews or management by walking around, others may prefer staff huddles. In addition to the face to face discussions, it is also highly recommended to conduct an anonymous survey. An anonymous survey will provide unfiltered feedback that may be useful to detect hidden, non-obvious issues. Additionally, depending on the size of the organization, it will provide a much larger sampling which can provide an understanding of the general morale of the staff.

3.4 Implementation Performance

The only way to determine if a return on investment or some level of success was achieved, is by measuring the performance of a previous state to the new state. If performance measurements cannot be conducted, then the investment should be considered high risk.

How do you measure two seemingly very dissimilar things, in this case, Waterfall and Agile performance? A starting point is to separate the qualitative from the quantitative components of the SDLC. Agile is ladened with co-mingled quantitative/qualitative statements which can make it seem difficult, however, if it can be counted or expressed using empirical data then it is a quantitative element. The table below shows a crosswalk of common performance metrics for the SDLC's. Not all elements in the table need to be measured, but at a minimum, unit of work, time, and money (scope, schedule, and budget) must be measurable and measured as key performance metrics (KPI's).

Table 1 SDLC Metric Crosswalk

Measurement Description	Metric Terminology*	
Quantitative	**Agile**	**Waterfall**
The proposed features, enhancements, upgrades.	Backlog	Requirements
Ranking of the most desirable features.	Ordered backlog	Requirement prioritization
Incomplete or inconsistent requirements.	Recidivism rate	Changes to requirements
The completion of projects to schedule and budget.	Agile Earned Value[4]; Epic and release burndown	Earned value, Schedule variance[3]
The average work a team does over time.	Velocity	Earned value
Consistency in workflow across the teams.	Cumulative flow	Resource leveling
Overall bugs/defects found in user testing	Defect rate	Defect rate
Defects found in production.	Escaped defect rate	Leakage rate
Average time it takes to fix a defect.	Defect resolution rates	Defect resolution rates
Indication of technical debt (application level) [2]	Increasing defect resolution times; reduced accuracy in time estimation, increasing time estimation, etc.	Increasing defect resolution times; reduced accuracy in time estimation, increasing time estimation, etc.
Cost to deliver all requirements.	Total epic costs	Total project costs
Qualitative Measurements		
User experience	UX studies, surveys	UX studies, surveys
Client satisfaction	Surveys	Surveys

1. Not exhaustive

2. Technical debt can be difficult to accurately measure easily. There are a variety of tools that can simplify the task in a more precise manner. For the purposes of this paper however, some high-level measurement approaches are listed.

3. PMBOK 2004

4. Sutherland 2014

3.5 Quality

Unlike scope, schedule and budget, quality is more difficult to measure since it can be both quantitative and qualitative. Quality means different things to different people, and time should be taken to understand what is most important to the organization. Using a car as an example, quality could mean luxury, style, and prestige for some, while reliability, cost and longevity are valued for others.

Understanding the complete breadth of software quality characteristics with a common language for communication is important. The software quality standard ISO 25010 systematically outlines software quality from 8 different dimensions, see figure 1 below [ISO 25010].

The ISO standard is comprehensive, and it is only presented here as a reference to depict the breadth of elements that can be considered for measurement. The organization would select the highest priority elements to be monitored before, during and after the transition. Selecting a few elements to monitor does not mean that all other elements are not important or will not be tracked, it only means that these are the key elements that must not be affected negatively by the transition. The measurement could be at the high level, i.e., Usability, Security, etc., or it could be a specific element like that of Operability and Fault Tolerance. Measuring the high valued quality characteristics provides the ability to identify issues, and to determine if the Agile implementation is having the desired effect on the final product.

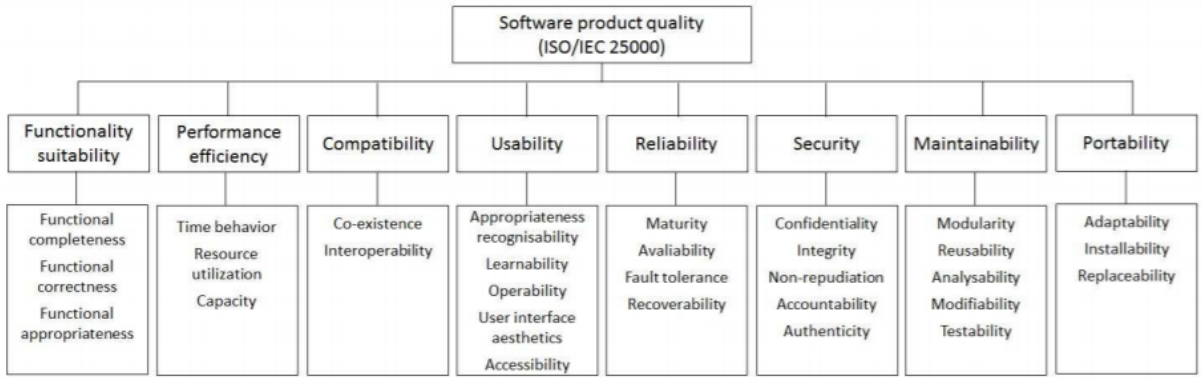

Figure 1. ISO 25010, Quality Software Standard

3.6 Value

Value is often mentioned as a focus of Agile and is touted as a differential between Agile and other SDLC's. Some organizations that employ Agile, struggle with the concept of value, to the point of it being overthought and hindering performance. Software value is not a new concept; the only difference between the two SDLC's with respect to value, is that Agile heavily emphasizes this characteristic, keeping it top of mind.

Like quality, value means different things to different people at different levels, which is why some may struggle with it. Using the basics of scope schedule and budget, i.e., the "iron triangle", coupled with the fourth element quality, a case can be made that these four elements determine software value. See figure 2 below.

In layman's terms, this means that the software development group is delivering what is desired, when it is desired, within budget and with the quality characteristics expected. By identifying and tracking metrics along these four dimensions, value is inherently being measured.

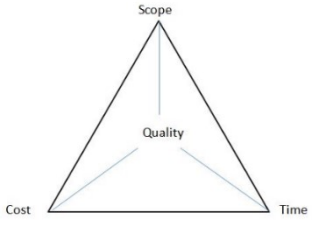

Figure 2. Value Triangle

4 XYZ Agile Transformation Analysis

When a plane falls out of the sky due to a design issue, the subsequent investigation and analysis will often identify systemic failures across many complex processes. Like a plane crash analysis, a failed, or in this case, a stalled implementation of Agile has many bad assumptions, missed opportunities and failures across several functional areas. A more detailed understanding of some of the issues and failures of the transition are provided below.

4.1 Business Case Assessment

A formal business case with an analysis of the existing organizational issues and how they would be mitigated using Agile was not conducted. While it is hard to imagine that an initiative of this importance would not have some rigor supporting the effort, it is not uncommon. The responsibility for this error lie with both the IT and Program senior managers.

The primary method utilized by IT to convince the Programs to move to Agile were anecdotal examples laden with superlatives. The Programs accepted this information at face value and did not request any due diligence on the claims being made. This was a missed opportunity to formally address the heart of a structural issue of having to reorder projects in the backlog because of the shared development resource as mentioned in section 2.1.

4.2 Planning and Implementation Assessment

There are three recognized approaches to transition an organization to Agile [Mckinsey]. The first approach is the big bang, ironically it is a Waterfall approach to implementing an Agile framework. The thinking behind this approach is that the transition will be disruptive using any approach, so it is best to minimize the disruption time and do it all at once.

The second approach is to transition in waves, tackling key horizontal and vertical components iteratively over time. For example, wave one might be to reorganize a specific isolated product line to Agile. Wave two may be to implement new tools that will better support Agile processes, etc.

The third approach is termed as “emergent” and is a bottom up approach, meaning that the transition is largely driven organically by the line staff and functional managers. This approach minimizes the risk of business interruption; however, it can take much longer to fully implement. Given the laissez faire management style, consensus decision making, and low risk tolerance, the XYZ organization naturally gravitated toward an emergent approach.

Staff was provided with formal outside training, in conjunction with an Agile coach to assist the teams through their first set of projects. Small, relatively easy projects were selected to limit business risk and ensure early success. During this process, discussions occurred about upgrading to more modern tools that better supported the Agile processes. Tools were identified but would only be implemented when time in the development schedule permitted.

In lieu of any extensive planning, it was assumed by the IT senior managers that once Agile was introduced, all the projects would naturally want to move to the Agile methodology. The pace of the transition, along with the limited realized value that Agile was providing, left the transformation initiative open for comparison and constant criticism by the Programs.

4.3 Development Vendor and Program Experts

The software development was outsourced to a capable vendor with all development staff residing onsite. The contract could handle both Waterfall and Agile development, this was accomplished by providing a “wrapper” around an Agile development to integrate it with the organizations project structure. This approach is referred to as an “adaption layer” and is common for organizations that do not have a high degree of contracting flexibility.

Even with this contract protection there was still reluctance by the vendor to fully embrace Agile for the following reasons:

1. The Programs had a limitation of providing knowledgeable resources on a consistent basis for requirements elicitation and demos. Given the system had five major intertwined Programs, it was often required to have all five Program Subject Matter Experts (SMEs) available for decision making. The Programs could sustain scheduled, short bursts of intense requirements elicitation and then minimal contact for follow-up if needed.

   This fire-and-forget environment was more easily accommodated by the traditional Waterfall process. In these specific types of systems, the Program SMEs were often the final decision makers and they were stretched thin for a variety of reasons. This is an issue that is commonly known in this vertical market. Finding and training a SME to a point where they are effective could take years. The development vendor, not being assured of having the right business resources in a reliable and sustained manner, viewed this as a risk that could be better mitigated by using Waterfall for large projects.

2. The services delivered by the organization were heavily regulated, and fiscal accountability and transparency was required. All projects had a 10% cost overrun flexibility against the final estimate given by the vendor.

   For Waterfall projects, change orders could be managed by the vendor in a more formal and traditional fashion. With Agile there is an inherent expectation of broad flexibility, where changes are continually expected and embraced for the duration of the project, however, providing this level of flexibility without additional money was a financial risk for the vendor. This was a source of continual frustration for the Agile teams.

4.4 User Testing Analysis

For these types of systems, User Testers began workflow testing after sufficient software functionality was delivered to the test environment. One would expect that Agile would be more advantageous in this model, however User Testing was only conducted on the Waterfall and not Agile projects during the development process. While counterintuitive, the Sprint incrementalism did not provide sufficient end-to end workflow to make it efficient for the User Testers until all the Agile project code was delivered to the test environment.

The Waterfall projects on the other hand, were being developed in an iterative fashion with major pieces of code delivered into the test environments at predetermined times. This approached enabled the testing to be done efficiently during the development cycle. This enabled the User Testers to conduct all or portions of the end-to-end tests in parallel or slightly lagging the developer's system integration test (SIT) team, prior to the final release. It should be noted that the inefficiency of testing the Agile code was not due to a limitation of the Agile SDLC, but rather, it could be attributed to the length of the Sprints (2-3 weeks) and the size of the Agile projects. A larger project with longer Sprints could easily be structured to deliver a greater amount of functional code for User Testing during the development period.

4.5 Desired Quality Metrics

Using the ISO 25010 terminology, the XYZ organization identified Functional Completeness, Modifiability, Usability and Security. Many more metrics were monitored as a matter of course for all projects; however, the organization highlighted these elements for the reasons described below.

The definition of these terms by the standard, and the reason for the organization's focus on these metrics are described as follows:

"**Functional completeness** - Degree to which the set of functions covers all the specified tasks and user objectives."

The organization focused on functional completeness because they were unsure if user stories could accurately capture the intertwined program requirements to the level needed. Functional completeness was measured and monitored in User Test by tracking the number of changes or updated user stories required by the Programs.

"**Modifiability** - Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality."

This was selected because they were interested to see if the defect rate was comparable to their existing defect rates, which was currently monitored by the test team.

"**Security** - Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization."

Security was highlighted because they were unsure how the security requirements were captured using user stories. Security was measured by the number of changes requested during the Security Testing, and if the documentation was sufficient for the governing compliance body.

"Usability - degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use."

The XYZ organization, as a matter of course, conducted quarterly surveys to their internal staff with respect to user satisfaction. This existing measurement tool was adjusted to delineate between project types without identifying the SDLC utilized. This blind survey approach was intended to mask the SDLC used from the client to not skew the results.

4.6 Quality Analysis

The post Agile implementation quality measurements above, indicated that Agile did not significantly degrade or improve the organizations output. Below are some notable improvements and issues found by the quality elements that were being tracked.

4.6.1 Usability and Functional Completeness - The Programs valued the periodic demonstrations provided by the Agile process; they were presented not only to Program SME's but to the line staff as well. This approach fostered a level of inclusion that was well received. It allowed the organization to get feedback from line staff which was not done with the Waterfall projects until User Test. The feedback provided typically resulted in minor changes that would impact efficiency and usability. It should be noted that the discussions that occurred during the demonstrations were informative to the engineers beyond just the features that were presented. This increased communication flow was noted in some of the user survey comments.

As mentioned previously, the surveys were adjusted to elicit feedback on specific projects, this enabled the comparison between like sized Agile and Waterfall projects. The scores were reviewed pre and post implementation and indicated a slight increase in user satisfaction with the Agile projects.

4.6.1.1 Documentation - The level of initial documentation was considered insufficient and was updated to reflect more details in the user stories. When the handoff occurred to operations, the documentation for the project was lite, however it did not contain key information that was considered essential by the organization.

Example: An Agile project tasked with the creation of a dashboard that monitored the processing of a large amount of documentation flow for the Programs was commissioned. After the release to production, an operations supervisor noticed some anomalies that were thought to be from incorrect calculations. When operations asked for design documentation for review, the user stories were provided which were inadequate to determine the data and calculations that were used to derive the dashboard information. To further investigate the issue, engineers would need to be consulted, while meeting with engineers is not an issue per se, it did highlight an increased dependency on the vendor.

This also indicated that the accuracy of User Testing may be affected, meaning that the testers were testing the dashboard functionality to the level of the user stories provided and not to the underlying business logic. Lastly, when the issue above surfaced, the development vendor became concerned that the reduced level of documentation had increased their risk and potential liability.

While this documentation example may seem insignificant, the dashboard was court mandated due to a death of an individual, caused in part by incorrect system processing. These issues were addressed by providing additional pertinent information within the user stories. This had the effect of increasing the overall number of user stories.

4.6.2 Security - The organization implemented a variation of Scrum called “Secure Scrum” (S-Scrum) [Pohl 2015], which provided a technique for identification and tagging of user stories that have a security component. This approach not only provided security requirements management and traceability for testing; it also was helpful in raising general security awareness among the team.

4.6.3 Modifiability - The Agile teams could handle minor last-minute changes easily, in some cases too easily, i.e., during the final release certification period. This had the potential to disrupt testing, training, and since the projects were part of a project release train, a change this late in the process could be a risk to the build process. The teams agreed that no changes would be done prior to the last Sprint.

4.7 Value Analysis

In general, the XYZ organization at the highest level is receiving significant value because each project being delivered a has fixed scope, schedule, budget, and quality requirements. As the needs shift for the organization, the projects are reordered in the backlog to reflect the changing business circumstances. This natural structure provides a macro level of value attainment for the organization.

At the individual project or micro level however, the reshuffling is perceived by the Programs as a value deficiency. When individual projects are reordered in the backlog, the time component of value is not being satisfied for these groups, i.e., the project is not delivered when it is desired.

Implementing Agile will not fix this fundamental problem, however the act of transitioning to an Agile SDLC was an opportunity to review existing issues within the organization, as mentioned in section 4.1. Identifying this issue prior to the Agile implementation would have highlighted the source of frustration and, more importantly, set the correct expectations with respect to the limitations of Agile in the context of the organizational structure.

5 Conclusion

This paper has demonstrated some of the many different aspects to consider when implementing an Agile framework in an enterprise. These aspects, if not understood or addressed prior to the implementation can result in a stalled, chaotic, or even failed Agile implementation. This case study highlights the confluence of misaligned expectations for Agile, lack of preplanning analysis, and laissez faire management style. These conditions, in combination with management’s decision to utilize an emergent Agile implementation approach, resulted in an unintended negative consequence of creating a level of distrust between the Programs and the IT department.

Understanding the problem to be solved by conducting analysis of the context and structure of the organization relative to the Agile principles, will help characterize the desired end-state which is key to setting expectations. Further, this form of modeling can be useful to determine where structural changes can increase the probability of success, and ultimately, a more Agile organization.

References

Brosseau, Daniel, 2019. “The journey to an agile organization” Mckinsey May 10, 2019 https://www.mckinsey.com/business-functions/organization/our-insights/the-journey-to-an-agile-organization#

Leffingwell, Dean 2007. Scaling Software Agility: Best Practices for Large Enterprises. Addison-Wesley.

Mattis, Jim N., and Bing West. 2019. *Call Sign Chaos: Learning to Lead. New York: Random House.*

Beck, K., et al. (2001). “The Agile Manifesto.” Agile Alliance. http://agilemanifesto.org/

Project Management Institute. 2004. *A guide to the project management body of knowledge (PMBOK guide)*. Newtown Square, Pa: Project Management Institute.

Sutherland, Jeff, 2014. “Agile Defense - The transformation of how wars are fought, how logistics are delivered, and how the Department of Defense does business.” Scrum Inc 2014.

International Standards Organization (ISO). 2011. ISO/IEC25010 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva, Switzerland: International Organization for Standardization.

Ramesh, Balasubramaniam, et. al. 2018. “Consider Culture When Implementing Agile Practices.” MIT Sloan Management Review posted Oct. 03, 2018 https://sloanreview.mit.edu/article/consider-culture-when-implementing-agile-practices/

Pohl, Christopher, Hof, Hans-Joachim, 2015. “Secure Scrum: Development of Secure Software with Scrum”. The Ninth International Conference on Emerging Security Information, Systems and Technologies 2015.