A RESCUE OF A COTS PUBLIC SECTOR PROJECT USING AGILE, SCRUM, LEAN, AND SAFe PRINCIPLES, PRACTICES AND FRAMEWORKS

BEN BLANQUERA, Pillar Technology

In this paper, we'll chronicle the turnaround of a failing enterprise scale (200+FTEs) Oracle - COTS (Commercial Off The Shelf) Implementation in a public agency. A decision was made to do a hard reset from a waterfall to an agile methodology. A delivery methodology was designed and implemented which included practices at enterprise, program, and team levels. This methodology encompassed elements from agile, scrum, lean, and SAFe [1] frameworks. While we did not have a specific intent to use the SAFe framework it turns out that we ended up using some key elements of the framework at the different levels. We'll make explicit the connection between our practices and those in SAFe framework. We'll also examine the transition from a bureaucratic, with slow-decision velocity, siloed culture to one of fast moving, transparent, trusting, and delivery-focused culture. From hard reset, it took 15 months to the first release with 1/3 of the previous staffing and the team is now on a six-month release cadence. Lastly we'll review our key learnings/success factors focused on:

- the right people with the right attitude and sense of urgency,
- program governance and refactoring,
- release , features, and stories,
- transparency and trust,
- continuous improvement and learning,

1. INTRODUCTION

In June of 2012 our firm received a call from a large systems integrator asking if we'd be interested in leading the methodology and delivery of a large agile rescue project. After four years and over 800 man-years of effort the systems integrator had decided to do a hard reset of the project. The project was a waterfall implementation of a large Oracle COTS system to modernize and re-platform off of old mainframe systems.

What was meant by a hard reset was that all but 10 of the original 200 member team were being let go and that we were going to start fresh using agile principles and practices.

Going into the project some of the key challenges were:

- How quickly could we define and implement a delivery process that levered agile principles in the midst of a high-pressure project rescue environment?
- Could we shift the culture to high trust mode?
- Could we break up a large monolithic COTS system release into a incremental delivery model?
- What mechanisms would we need to put into place to manage the complexity of the 40+ integrations?
- What could we do to incorporate traditional QA phases (system integration testing, performance testing, user acceptance testing) into an agile process?

This paper will be broken up into the following sections:

- Team structure
- The story (each section will have a summary chart, narrative and lessons learned)
  - The big ugly - first 90 days
  - It still hurts - days 90-180
  - Hitting our groove - days 180-360
  - Countdown to release - days 360-450

Authors Address: Ben Blanquera, 5606 Barney Drive, Dublin , Ohio 43016 email: bblanquera@pillartechnology.com

- The practices
- Keys learnings and success factors
- References

2. TEAM STRUCTURE

The customer was a large public sector agency who provided the product owners, business analysts, testing team, and part of the infrastructure support. Program leadership was a large systems integrator with overall accountability for the project. Program leadership provided a whole complement of developers, QA, infrastructure, and reporting resources. My company, Pillar Technology, was accountable for methodology and leading the multiple teams. Figure 1 illustrates the team structure with responsibilities of subteams.

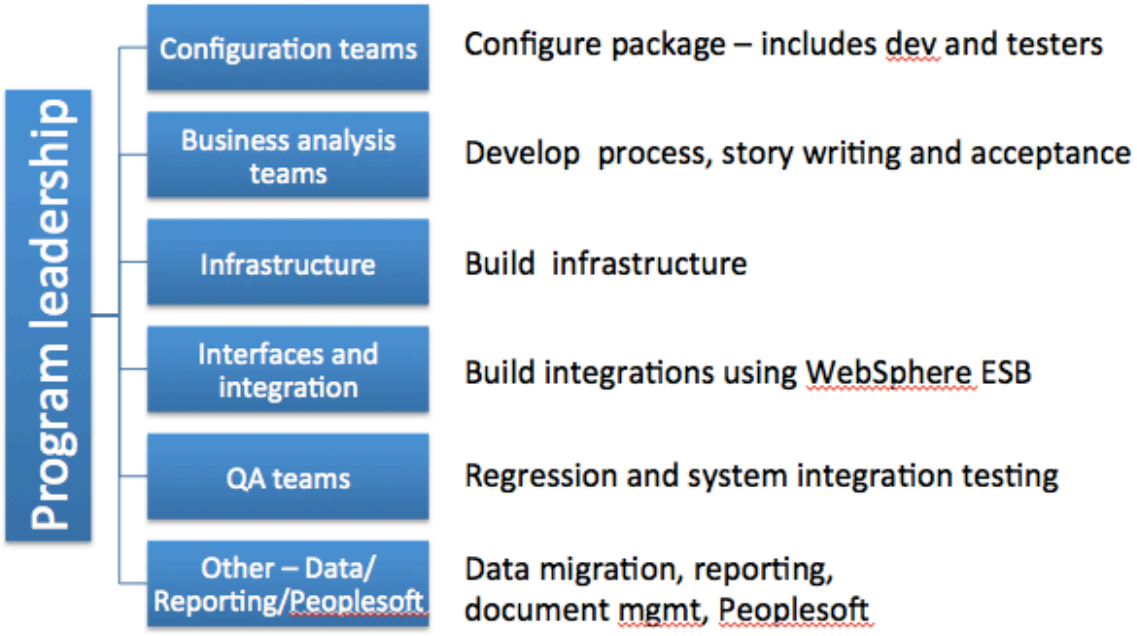


Figure 1 High-level team structure

3. THE STORY

3.1. The big ugly - days 0 - 90

The team parachuted into what seemed to be a software development war zone. We called this phase the "big ugly" because no matter where you turned, it was ugly. It was ugly due to the climate of mistrust, broken governance, dysfunctional teams, dysfunctional processes, and frustrations associated with nothing having been delivered in four years of building.

Table 1: Phase summary with challenges and tactics for the first 90 days

Challenge	Tactic	Comments
Build a "winning" culture	• Apply lean software thinking [2]	Lean provided a holistic approach.
Building the right team members	• Get experts in the COTS package on the team • Insist on senior leadership engagement • Align vendor contracts to align with the program goals	Our COTS experts came with domain expertise. This was important to align as much of the out-of-the-box functionality with the implemented solution. Close to full-time product owners are a necessity in a large complex COTS implementation.
Getting a team space	• Get whatever room(s) are available to co-locate the team	A key to building a team is to increase the velocity and frequency of communications. An enabler is to force everyone on the team in one room
Building a backlog [3]	• Identify solution "architect" to "own" how to solution into releases, features, and epics	Early in the program we identified "easy" features to build confidence in the process.
Creating appropriate development and test environments	• Create a cloud environment early so internal infrastructure doesn't become critical path	We found that the existing client infrastructure was not ready to support multiple agile teams.
Getting team effectively using Scrum [4]	• Experienced Scrum leaders • Train the entire team on "day in the life of agile"	It was very difficult to get everyone used to breaking up the work into stories and then going into the test and approve cycle.

The first order of business was to build a winning culture, get the team organized and functioning as a team. The challenges and tactics associated with this task are listed in Table 1. We also wanted to start delivering working software in the first 90 days.

Within the first week we took 25 core team members through a "Day In The Life" of agile to start creating a common vocabulary and view of what the process was going to look like. During the class we were met with a healthy amount of skepticism.

We knew that a critical success component was strong product ownership. We ended up with four strong product owners who were equivalent to division heads. The four product owners ended up dedicating a minimum of 50% of their time to the project.

We also commandeered two large conference rooms. In conference room A, we housed the configuration team that configured and developed the Oracle COTS product. In conference room B we housed the infrastructure and platform teams. The job of this team was to build the infrastructure, integrations, reporting, and data migration.

While we were getting the logistics and training going, we entered into a sprint-0 phase where we worked out basic technical architecture and started creating a backlog of stories for the team to start working on.

We used a Scrum framework to guide the team process.

No matter how messy the backlog was, we were bound and determined to start building software sooner than later.

Lessons learned during days 0 - 90 were:

1. Assign overall tech lead over all the teams to drive direction setting and decision velocity. Due to lack of ownership and leadership of the infrastructure, interface, and integration team, key architecture directions were not set, which created significant amount of technical debt.
2. Don't bring developers on until you have a minimum of two sprints worth of backlog groomed. We did not have adequate backlog groomed before the development team arrived. Frustration was created when developers were ready to build and the backlog was not in a state to be built.

3.2 It still hurts - days 90-180

We had significant progress during the first 90-day "triage". We called this phase "It Still Hurts" because we were still struggling with velocity and our program governance processes.

Table 2: Phase summary with challenges and tactics for the days 90 – 180

Challenge	Tactic	Comments
Build out the team spaces and onboard team members	• Build out team spaces as soon as possible. Doesn't need to be pretty just functional • Have a explicit onboarding plan and accountability	
Break up work into deliverable "chunks"	• Use features and epics	
Develop forecasting and estimation models and metrics for velocity and release timing	• Configuration/ development teams worked off of one backlog • Develop consistent estimating across teams • Use proxies to develop forecasting models • Build a master release plan	Estimating was done only by the configuration teams since it was determined that they were the critical path. Having a transparent model is critical to the "refactoring" sessions.
Build credibility by committing to and hitting velocity	• Team commits to velocity • Use lean and Kanban thinking to identify and clear bottlenecks • Make sure your Scrum masters are velocity adders	Flow of stories across the board is a critical success factor. It was important to make clear to the team what the cycle time across the board for cards and how important it was to reduce that cycle time.

Build a "we" mentality where everyone had a mutual accountability to deliver the program [5]	• Insist on product owners available in the room frequently • Have a daily leadership standup • Build effective big and visible board • Build and execute a ongoing communication plan • Leverage online information stories such as "Sharepoint" and "Rally" to make detailed information accessible and transparent • Don't forget team building activities	Education of the product owner and their role was critical to having the users buy in. We also used the end of sprint retrospectives as a to develop a climate of continuous improvement.
Think through a QA strategy	• Source QA teams early • Testers should be on each team • Automate as soon as possible • Get end users on teams as manual testers	There are multiple tiers of testing including unit, story, and sprint testing. The sprint testing was automated.

After the first 90 days, we were quickly outgrowing our small conference rooms. Based on high-level estimates we knew that the small core team was going to grow into multiple teams with 6-10 members on each team. We eventually grew to 12 teams with a total staffing of 125.

Based on those estimates, decision was made to relocate to a much larger area to accommodate the growth The primary attribute of the area that we cared about was that we could be 8-10 team members seated at a set of tables with lots of white board space.

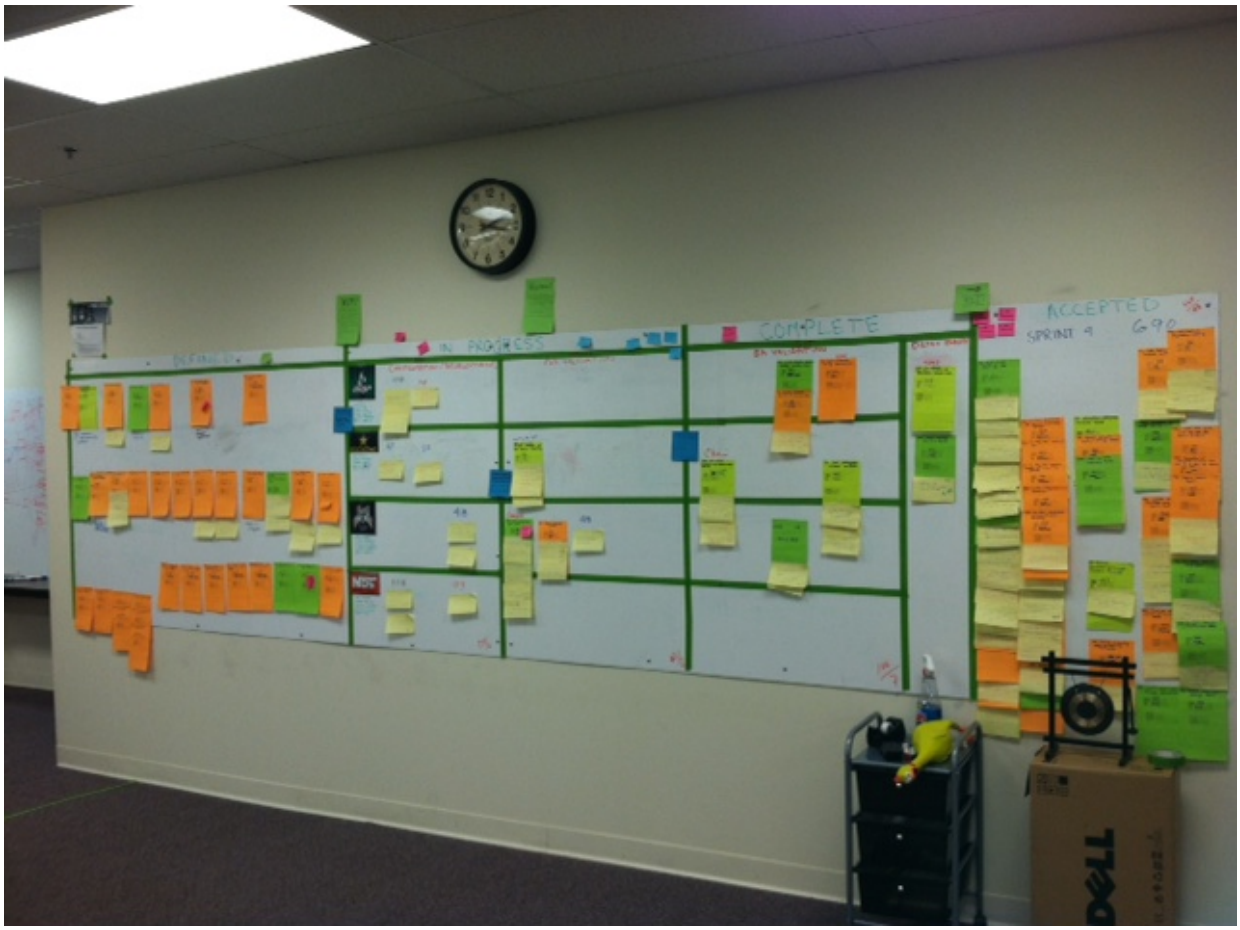


Figure 2 - The team board

The development / configuration teams worked off of a common backlog seen in Figure 2. The far left of the board was the backlog arranged by feature. In the middle of the board were the steps through development including in process, in testing, ready for BA review. On the far right was ready for product owner acceptance and approved stories. Every time a story was accepted by the product owner, the gong you see on the bottom right-hand corner was rung.

A major question we tackled during this phase was how could we approach rolling out incrementally an incredibly large COTS program into multiple releases. A majority of COTS/ERP applications are not designed to be rolled out incrementally. Much of this is attributable to the nature of the business processes, complicated integrations, and data challenges that make it difficult to slice the system into portions that can be deployed incrementally.

Due to regulatory and customer challenges as well as the nature of the data migration, integrations, and legacy system retirement, it was determined that there could be multiple releases in 6-month increments. The first release, which carried the weight of the system core, would take 15-months.

Our approach was to complete features each sprint, which could then be fully string tested [7]. Integrations would be sequenced to coincide with feature development. In this light we were delivering functionality incrementally, but could not release the product feature by feature.

The configuration team's velocity was measured when code was accepted by product owners in the QA environment. At the end of every sprint,, the QA team would then construct regression tests for what was built in that sprint. We leveraged the regression tests as smoke tests for the various environments.

During this phase we developed an estimation model that estimated the average number of stories per feature and the average number of points per story. These averages were based on forward projecting the actuals from previous sprints. The mode was then cross validated/adjusted based on comparisons with "expert" opinions and data from other implementations of this COTS package.

Part of building the trust with the end user base was to be able to hit velocity commitments. We challenged our teams to find ways (while working at a sustainable pace) to find improvements to our work processes. There were several iterations whose commitments were not met, which caused severe concern from the user base. Fortunately because we were showing "working" software during the end of sprint demos to the larger user base, trust was earned by evidence on incremental progress.

We were also diligent with team building activities within and across teams to build rapport. A special effort was made to build rapport between the team leads. This was done with weekly social activities. "All" team lunches were held to establish personal connections among team members.

Lessons learned during days 90-180 were:

1. Get a big visible board and Scrum process around backlog grooming. Set WIP limits for stories waiting for acceptance. There were several times that the team was running out of work due to the backlog not being well groomed. In order to keep up our velocity, it was important to get fast feedback from the product owners. At one point, the product owners were not approving stories in a timely manner. We held our line in terms of WIP for stories awaiting acceptance and told the product owners that it was better to send the dev teams home until acceptance caught up than to create new WIP. This got their attention.
2. Get leads biased for delivery and COTS package experts on each subteam. There were three issues that led to inconsistent team velocity. One was the lack of experienced COTS package developers on some of the teams. Another was that the Scrum masters were not "leading" their teams in a way that conveyed urgency. The last issue was a lack of clear technical direction on our infrastructure and integration team.
3. Define the QA strategy and source that team early. QA strategy and regression team was not sourced early, leading to a large amount of QA debt. The team had to scramble to get the automated regression testing caught up.

3.3. Hitting our groove - days 180 -360

In this phase, team velocity continued to improve and the overall program churn settled down and the team felt like they were "hitting their groove".

Table 3: Phase summary with challenges and tactics for the days 180 - 360

Challenge	Tactic
Continue to hit velocity targets and drive continuous improvement	• Think "flow" identify and resolve bottlenecks quickly • Experiment with ideas from retrospectives

Successfully navigate the integration "jungle"	• Sequence integrations to be played • Fake and mock and stub [6]
Figure out what we need to do to release	• Develop a big and visible master plan • Look for ways to do things in parallel not serially

We were committed to working at a sustainable pace and avoid any death marches. To that end, we had developed a velocity schedule that was reviewed at the end of every sprint and that reconciled with the targeted release dates. Additionally we took retrospectives very seriously and challenged the teams to come up with suggestions to reduce the cycle time to releases. Anecdotal observation indicated that we improved the cycle time by 20% -30% by implementing suggestions made during the retrospective process.

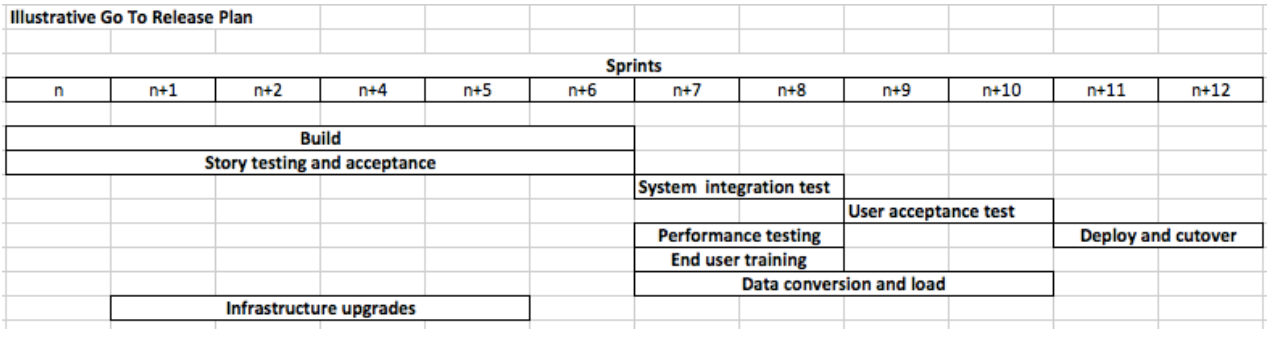
As with many COTS systems, there existed multiple integrations with this implementation. They included:

- batch and real time data input systems,
- correspondence generation systems,
- document management systems,
- PeopleSoft revenue management system, and
- reporting systems.

In all there were 40 different integrations. We used Websphere ESB to integrate all the different applications and services. The work was sequenced so that as functionality was built out on the COTS system for a specific feature, the corresponding integrations were being developed so that we could initiate string testing as soon as a feature function was completed. We knew that the biggest risks in the project were in the integrations and consequently string tested as early as possible to expose issues.

It was important that we could absorb changes while still making sure everyone knew what were the critically important elements of the release and when decisions needed to be made. To that end we developed a macro release plan that was "refactored" at the end of every sprint (see Figure 3). As we drove towards release, we practiced the notion of "Last Responsible Moment" [8] which delays commitment until the last responsible moment, that is, the moment at which failing to make a decision eliminates an important alternative.

Figure 3 - Illustrative example of release plan showing critical elements



From a QA perspective, we used embedded manual testers on each team for story testing. Additionally we had an automated regression team that created automated tests. During this phase we identified a bottleneck around testing and our business partner provided expert end users to add capacity to story testing. The QA

professionals then focused on getting ready for and executing system integration test. We also increased testing efficiency by focusing on test data management. We found that 40-50% of the testing cycle was allocated to prepping the test data. An additional side benefit of using end users as manual testers was increased end user buy in and a set of experts that could grow into end user trainers.

Lessons learned during days 180 -360 were:

1. Address data conversion early and minimize the amount that needs to be converted. Issues were found when we started testing with converted data. We found that in some cases, we needed to define different business rules for converted vs "new" data.
2. Allot 2-3 times more time than you think you need for integration testing.. Determine an integration testing strategy early. Testing integrations were painful, as we needed to coordinate with trade partners from other organizations.
3. Fully automate builds and deployments. Get an experienced build engineer on board early. Since the deployment processes from development to QA environments were manual, we found it painful to promote twice per day in a consistent and predictable manner.

3.4. Countdown to release - days 360 - 450

During this phase the build phase was completed, and the focus was completing the testing required to "go-live"

Table 4: Phase summary with challenges and tactics for the days 360 - 450

Challenge	Tactic	Comments
Entering and exiting the system integration testing (SIT) and user acceptance testing (UAT)phases	• Make explicit SIT and UAT entrance and exit criteria • Don't forget about performance testing and tuning	We worked with the QA to develop the integration test scripts to make sure that we had clear acceptance criteria. Prior to performance testing, we arranged for "tuning" experts to be on site.
Effective communication and decision making	• Appoint a go live leader • Appoint a leader for the SIT and UAT phases • Daily standups for SIT and UAT	While SIT and UAT phases sound very "waterfall," they are at this time a necessary evil for a large COTS effort.
Manage the transition	• Get users involved as early as possible as testers • Develop training early • Build a go-live plan • Rehearse the go-live plan	Getting end users involved early allowed us to build capacity during the build phase but also had the nice fringe benefit of creating a pool of trainers used to the new system.

In this phase we went through system integration as well as user acceptance testing. There were a set of predefined system integration acceptance scenarios that had to be passed prior to entering the user acceptance phase. Concurrent with system integration testing was performance testing. The system integration testing was led by the QA team. User acceptance testing was led by the user community and revolved around multiple "day in the life" scenarios. Entrance and exit criteria were present for both the system integration and user acceptance testing (see Table 4)

As with most go-lives, the weekend before the go-live Monday, there was a countdown of activities. The most time sensitive part of the cutover was the window required for the data cutover. As we ran through the weekend, we started picking up steam on the schedule and finished cutover activities eight hours ahead of schedule.

We all came in at about 5am on the day of go-live. Users would start logging into the system at about 6:30am. From 6:30 to 9:00am, user volume grew. Our command center was strangely quiet during this time period. We were prepared for the worst in terms of defects. To our great satisfaction the number of defects found could fit on one sheet of paper and any issues found during the next few post go-live weeks were quickly resolved. Everyone on the teams indicated that this was the smoothest launch they had ever been part of.

Lessons learned during days 360 - 450 were:

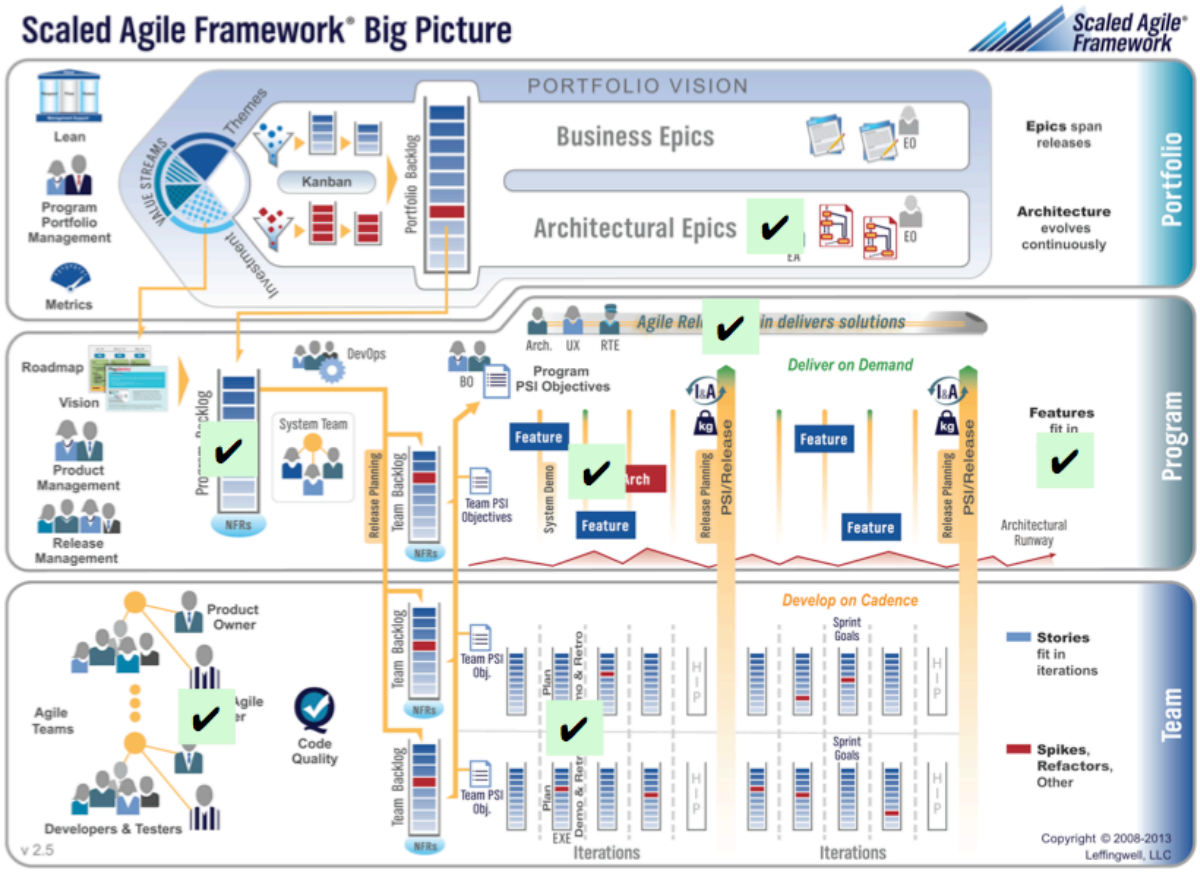
1. Build early performance testing of features into plan. Performance issues surfaced during System Integration and User Acceptance Test phases that we had to scramble to resolve.
2. Assign operations lead to be focal point for planning and coordination of go-live. There was confusion early on regarding readiness of infrastructure and operations for go-live.

4. THE PRACTICES

While we did not go into the project contemplating using the SAFe framework, in retrospect we used many of the same concepts in our implementation. The SAFe framework practices used are indicated in Figure 5.

Figure 5 - Chart with relationship of practices to SAFe

✔ - Indicates where our practices mirrored what is described in SAFe



While we used many of the SAFe practices, there were several significant practices that we used that were not in the SAFe framework, including: program refactoring, program modeling, and program estimation.

Program Refactoring

Refactoring was a practice where every few weeks we would:

- review the assumption sets regarding the release plan (i.e planned vs actual velocity, capacity, backlog , testing/QA stats),
- use a Theory of Constraints [9] mindset to determine accelerators and decelerators, and
- define a plan to keep us on track to the targeted release date.

This refactoring was done with full participation of the steering committee, product owners, and delivery team leadership.

Common backlog and estimating

All of the configuration teams worked off of a common backlog. Estimation was done using poker planning using Fibonacci sequence ratings on a scale from 1 to 55 by team leads. We moved to Kanban system for point estimating. After the first four iterations, we had enough data points to estimate the average point size for each story, which would then get reestimated once the card was played out of the backlog.

Kanban

We layered "Kanban" [10] thinking on top of the Scrum processes. Within the development team processes we used visual workflow and Work In Process (WIP) limiting to identify bottlenecks and improve "flow" across the process.

Grooming the backlog

Due to the complexity of the business process there were 15 business analysts charged with developing the stories. There was dedicated BA Scrum team whose work product was stories that were ready to be pulled by the team. A story was deemed groomed and ready to be pulled when:

- it was written in the format: As a <type of user>, I want <some goal> so that <some reason>;
- its required reference business rules were identified and put into a Sharepoint repository; and
- behavior-driven acceptance criteria in the Given/When/Then format were written and put into the Sharepoint repository.

During the feature definition process, we used the concept of "process spikes" to identify areas where it was unclear as to how a particular business process would be implemented. We worked to prioritize the "process spikes" early in order to take risk and uncertainty out of the project.

Product owners would be responsible for prioritizing the work. We had a weekly sprint planning session to review that had be done in the previous week, prioritize the next week's work, review the backlog readiness, and accept and new stories/feature into the backlog.

Metrics and models

We developed a release model with a few key aspects:

- Velocity as a function of staffing and expected learning.
- An integrated plan that included infrastructure builds, QA activities, data migration, cutover and post production.

Key metrics we looked at included:

- Velocity
- Burnup
- Average points per story
- Average stories per feature
- Cycle time through the process
- Number of days of backlog
- System availability and uptime

These metrics were reviewed on an ongoing basis as feedback loop to our release model and refactoring sessions. There was a daily discussion regarding the metrics at the morning leadership standup to create an opportunity of shared understanding and timely adjustments as needed.

5. KEYS TO SUCCESS AND LEARNINGS

Following is a summary of key learning and success factors for the project. These should be helpful for anyone contemplating a rescue implementation of a COTS project.

The right people with the right attitude and sense of urgency

- Get as many domain experts on the project team no matter where they come from. Start with an "A" team. A rescue and/or COTS project is not the place for anything less than the 'A' team. Make sure the COTS vendor has on-site presence.
- Replace the mentality "let's schedule a meeting to figure that out" with the question "whom do we need to get to resolve this issue right now?".
- Clock speed matters - think of progress in terms of four-hour blocks, not in terms of iterations.

Program governance and refactoring

- Put in place a strong steering committee that understands what's going on, is active in direction setting, has a high decision velocity, and clears blocks.
- Develop a transparent model that makes explicit assumptions regarding velocity, capacity, and target release dates. Use that model for discussions at the end of every sprint to remove things that are slowing down the team, implement items to accelerate, and set mutual expectations for release.

Release, features, stories

- Develop a release plan and them develop a priority order of features and stories to fit within that release.
- Identify someone to be "solution architect" who is an expert on the domain and the package. This helps speed up things by aligning as much of the implemented solution to what is possible without customization
- Identify spikes and prioritize them early to reduce risk.

Transparency and trust

- Educate everyone on the process.
- Have a communication strategy and plan. Some elements of that plan to include:
    - Daily standups for teams, for program leadership and for team leads
    - End of sprint demos and program review
    - Weekly steering committee
    - Sprint planning meetings
    - Informal out of work gatherings
    - Celebrations
    - Extensive use of big and visible charts
- Earn credibility and trust by:
    - Telling the team of good news fast but bad news faster.
    - Committing to velocity and more often than not hitting that commitment.

Continuous Improvement and learning

- Continuous improvement matters - velocity should increase over the duration of the project.
- Retrospectives need to be taken seriously - they should be done on both the team and program levels

- Cross train and pair across all functions.
- Use experiential learning [11] models to train team members.
- Assign a mentor to all new team members.

6. CONCLUSIONS AND DISCUSSIONS

When we first arrived on site it was unclear as to what specific principles and practices would be required to rescue the project. Compounding the challenge was the complex monolithic nature of the COTS product and integrations. As the project progressed it was clear that a hybrid approach of agile, Scrum, lean, SAFe was required.

Above all rescuing this project would not have been possible without senior product owners who were involved in a day-to-day basis, experts in both the COTS product and the business domain on the team, a transparent release model that was refactored often, and a culture of continuous improvement.

We did not employ engineering practices like continuous integration (CI) and test driven development (TDD). This was done due to the perceived technical challenges with implementing those practices in a COTS environment. Most of our practices centered around scaled agile management practices. It is felt that in subsequent releases, increased utilization of agile software engineering practices would further improve the velocity. This remains an open question for future releases.

By no means was this a pain-free project. There were many bumps during the road. The primary things we would do differently next time would be to:

- spend more time architecting the solution path through the package to improve backlog grooming before we bring a large development team on site,
- develop the quality assurance strategy early and resource automation activities early,
- focus on and resource dev-ops [12] early to enable rapid and frequent deployments into environments, and
- decide early on data conversion strategy and do mock data conversions early to use as test data

7. ACKNOWLEDGEMENTS

I'd like to thank my colleagues at Pillar Technology for all their help and support. I'd especially like to thanks Tony Nguyen, Mark Price, and Mike Evans for great work on the project and to Daryl Kulak, Matt VanVleet, and Bob Myers for their strategic insight and guidance. I'd also like to thank Hakan Erdogmus and Joseph Yoder for all their help shepherding this paper through the editing process. Last but not least I'd like to my wife Sandy Blanquera for her help and support as I worked on the project and the paper,

REFERENCES

1 . Scaled Agile Framework website - http://scaledagileframework.com/
2. Lean software thinking wiki page- http://en.wikipedia.org/wiki/Lean_software_development
3. Dean Leffingwell, Building requirements and backogs - Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise (Agile Software Development Series), 2010
4. Scrum guide - https://www.Scrum.org/Portals/0/Documents/Scrum Guides/2013/Scrum-Guide.pdf - zoom=100
5. Peter Senge, Building a "shared vision" - The Fifth Discipline: The Art and Practice of the Learning Organization, 2007
6. Martin Fowler, "Faking, mocking and stubbing," http://martinfowler.com/articles/mocksArentStubs.html
7. String testing - http://www.geekinterview.com/question_details/34491
8. Mary and Tom Poopendieck, "Last Responsible Moment", Lean Software Development: An Agile Toolkit, 2003
9. Theory of Constraints wiki - http://en.wikipedia.org/wiki/Theory_of_constraints
10 . Kanban wiki - http://en.wikipedia.org/wiki/Kanban_(development)
11. Experiential learning wiki - http://en.wikipedia.org/wiki/Experiential_learning
12. DevOps wiki - http://en.wikipedia.org/wiki/DevOps