Travelogue - To Leanville

Harish Krishnaswamy (Software AG)

2013 December 18

Abstract

At Software AG webMethods (wM) R&D labs, we are a team of over **380 engineers** who build enterprise products focused on integrating applications, business processes and partners. Circa 2010, the wM Suite was a salad of about **30 products** developed by more than **20 teams** across **9 locations**, several of them brought together by M&As. We shipped outstanding products that are proven industry leaders but it was painful challenge to integrate, test and ship them together on time. One of our major releases originally planned for **18 months** was delivered **9 months late** and had more than **11,000 known defects** when released.

It was then that we embraced Scrum as our development model. But we still needed a framework to synchronize individual teams and products and deliver as a single Suite. We found an answer in the Lean Software Developmental model with its focus on value, flow and progressive elimination of waste. We chose to take a trip to Leanville to *accelerate the flow of value to our customers* on time, with verified quality and doing more with less.

The Value Stream

THE CENTRAL IDEA

All teams pull their work from a **SINGLE BACKLOG** creating value that **flows continuously** from idea to delivery and delivering **ONE** integrated and tested **Suite build every 4-6 weeks** – everyone marching to the **beat** of single drummer.

Suite Backlog

- Product Owners (POs) sift through the ideas collected from various sources and shape them into minimum marketable features spanning one or more products and add them to the Suite Backlog.
- An algorithm prioritizes features from multiple POs in the backlog. It assigns a rank based on relative rankings from POs, budgeted allocation and effort already spend on a given investment area (broader strategic focus to which a feature belongs to). Suite-wide and cross-components features are favored over others.
- This allows a systemic control on what features can be picked by teams and when. It also ensures that the teams spend their effort in line with the budget guidance from business.

Every team has a filtered view of the prioritized backlog. The team sees only what it can and should work on. Each team works on a limited number of features at a time and cannot take up new features even when its in-flight features are blocked for any reason. This ensures that the overall system WIP is in check. Any stories beyond the team's average velocity are automatically hidden in the team backlog view.

Single Piece flow - Build Promotion Process

As the newly created feature flows down the value stream, every source change gets tested at the product level for regression daily (build verification tests). If the change passes, it gets tested at an integration level (product interfaces, system integration) and then for stabilization. Teams may start small but must test daily. They strive to get into a cadence (takt) of successful daily promotions. Over time, each team grows a rich test bed of functional, regression and integration tests.

Stop the Line - Andon

This practice flows straight from the Toyota Lean Assembly line. When a defect is found, all production stops until it is rectified. Anyone can pull the cord to stop the line once a problem is found. In our system, we try to protect yesterday's investment before delivering today's value. Every time an existing test fails, the code must be fixed before the new change can flow further. Any test failure results in a Blocker.

- When there is a blocker, the responsible teams) immediately try to fix the problem and provide a new build as soon as possible. All dependent components then repeat their tests against the new build and restore the Promotion flow downstream. A delay in blocker resolution results in a source-code lockdown for that component (local lock). Or in extreme cases, on all components across the suite (global lock). This prevents new changes from getting into the system until the existing blocker is resolved.
- After restoring the flow, teams huddle to do a Root Cause Analysis (RCA) and learn how to prevent similar issues in future and get better at detecting /resolving them. The blockers are monitored and

discussed on a daily basis at all levels in the organization. Their trends are the leading indicators of suite stability, quality and help us find the sustainable pace of delivery.

Continuous Improvement

Our processes are in a state of continuous evolution with changes driven by the teams themselves, with support from the Change Agents team. Everyone, individually and as teams actively design and shape the processes iteratively and empirically. They are encouraged to experiment - introduce changes to current practices locally, test and analyze them with the intention of *going Global*. The details and results of the experiment are shared in a central knowledge base, setting the stage for new experiments, diffusion of ideas and in some cases, a wider adoption of the practice.

In sprint retrospectives, teams actively identify what work is wasteful, how it can be avoided or improved closely resembling the Kaizen sessions in Lean practice.

Learning

The most challenging phase during the journey was to keep the build promotion flowing continuously *while* developing new features. At early stages of Build Promotion, we frequently gridlocked into inter-component blockers and source locks. And we struggled to find a sustainable pace for delivering new features while adding tests and resolving issues.

Prior to the adoption of the Lean Assembly line model, developers would focus on adding a lot of new features quickly. Any regressions/defects reported by Testing were deferred to a separate, long stabilization phase where they could be addressed. This leads to a 'fast development' illusion in the early stages of the release while in reality we had only accumulated a huge amount of technical debt to be addressed at the tail end of the release.

In the new model, any regressions found in daily testing had to be fixed immediately and the product could not ship a new build until all tests passed. Without a steady stream of builds (flow), developers working on new features could not share their daily code changes with colleagues across locations and other dependent products. Any delays resulted in source code lockdowns and no further changes were permitted unless the blocker issues were resolved. This broke the rhythm of new development and much of the developer's time was spent on fixing issues and adding tests to their build verification suite. Continuous Integration exposed broken interfaces and caused some products to block on others that were fixing their blockers. Soon, the entire system gridlocked with teams focused on running tests and fixing defects rather than writing new features and delivering new changes to other products. On the surface, it appeared that we were turning out features slower than before.

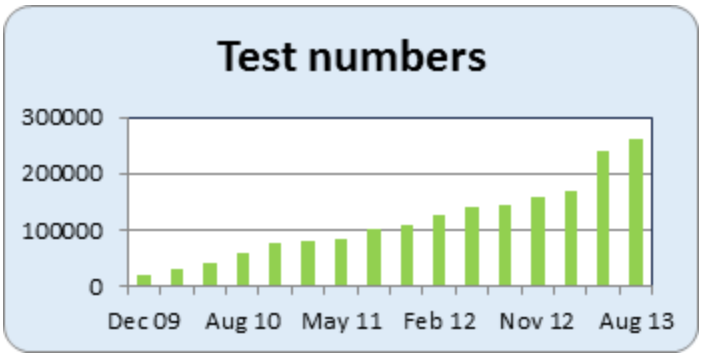
To avoid source code locks and resume feature development, teams began focusing their energies on preventing cross-component, cross-location blockers that were costlier to resolve. Teams evolved new practices like notifying dependent teams of upcoming changes, sharing tests and test environments to catch issues early and cheap. They voluntarily synchronized their build cycles,

check-in times and sprint schedules such that each product could promote its builds smoothly (achieve flow). They added new automated tests to fish out the problems earlier and verify changes quickly.

In time, the test beds grew and the daily tests began to catch issues that earlier were caught only in downstream stages weeks later. Automation allowed developers to test their new changes locally, find and fix issues quick and cheap. Fewer defects found downstream meant that there was lesser rework and re-testing in the tail-end. New features did not require the long stabilization phase anymore. Suddenly, we figured that we were delivering features with high quality earlier than expected. The teams were now more confident in predicting and hitting the delivery dates.

- The daily challenge of running time boxed tests and resolving blockers across time zones from IST to PST led to the birth of special-focus virtual teams like the Suite Promotion Coordinators. This team closely resembles sprinters passing the baton in a relay race and ensures the build testing, promotion and trouble-shooting activities continue uninterrupted. This has also helped product teams across centers come together and collaborate at new levels. We have adopted common tools, CI systems and collaboration tools. We have also customized our bug tracking system and reporting tools to optimally support our processes. This helps teams deliver green builds daily and get out of blockers as quickly and cheaply as possible.

- The relentless focus on reducing cycle times and preventing blockers has contributed to a steady growth of automated regression tests. This allows us to detect issues early and cheap. It also eliminates wasteful activities like rework, rebuilding and repetitive testing driven by late integration and associated overheads in communicating, documenting and tracking inter-component issues. Management overheads like creating status reports, Issue triaging etc. have also come down drastically.



- After a major release, the development tests are turned over to support teams and used as regression safety nets. Stage-wise automated testing ensures the fixes do not cause regression in the product, does not interfere with other components or other fixes shipped in the suite. Like in feature development, it is now possible to release quality fixes early and in a predictable manner. Instead of pushing fixes to individual customers on separate timelines, we now publish a Fix release schedule and allow customers to pull their fixes on designated dates. Since our suite testing resembles our customer environments closely, we have very few regressions caused by new fixes. The quality and predictability of both development and sustaining deliverables have resulted in more than 80% reduction in customer escalations, fewer support requests and overall

number of fixes.

- We also observe that the processes have become simpler and leaner over time - lightweight RCAs, simpler Sprint Planning Reports. The drive to do more with less cause teams to closely look at the documents and reports they generate. They do just enough to operate effectively and eliminate anything that does not directly contribute to help them achieve quality, prevent defects or reduce their delivery cycle time.

- Real time dashboards and automated reports have increased the transparency in operations at all levels. The teams measure what they want to control and quickly respond to trends and information that the metrics give them. The data and transparency allows for better planning, coordination and provides continuous feedback on what works and what is not working. Build Promotion Reports and Blockers are tracked across the suite daily. Teams track their backlogs, impediments and progress in Daily Stand Ups and Retrospectives. Cross-component and Suite wide features are tracked at the upper levels of management on a weekly basis. All reports and dashboards are visible to everyone and can be pulled on demand.

- Cycles are shorter with progressive elimination of idle time and non-value adding activities like rework, repetitive testing, excessive reporting, late integration and testing etc. Teams improvise to simplify and automate repetitive operational work whenever possible. Predictable rhythms of activities and WIP limits minimize wait times for features/code.
- The change process is now self-sustaining as teams have imbibed the habit of constantly learning and looking for opportunities to do things better and with less. Teams self-organize to eliminate bottlenecks and focus on global optimizations ahead of local ones.

The Scorecard – *Three years down the path*

Our first **6-monthly release** rolled out on November 15, 2013 exactly as planned at the beginning of the release cycle. We have just **1200 issues** now across all versions and products. The customer situation has improved with steadily decreasing customer incidents, response times and hot escalations. We have a growing test bed of more than **250,000 tests. 50,000 automated tests** verify the suite daily. We have a potentially shippable Suite build set every **4-6 weeks**. Significantly, this has been achieved while the organization continue to add new products, teams from new locations and features to the mix.

PRE-LEAN		CURRENT SCENARIO
Planned 18 months.	**Release cycle**	6 months
Delivered 9 months late	**Predictability**	On the planned date
11,000	**Known Issues (All Releases cumulative)**	1200
Manual during defined phases.	**Test execution**	50000 tests daily and counting
Late Big Bang integration	**Integration**	Continuous. Automated
Few close to the end of release	**Shippable Integrated Build**	Every 4-6 weeks
2-3 months	**Stabilization phase**	2 weeks with overlap on next release preparations
Service Packs and Individual fixes	**Support Fixes**	No Service Packs. Fixes released on a scheduled calendar.
Pushed upwards, largely manual compilation	**Status Tracking / Reports**	Data/Information Pulled real time from tools and dashboards
Product Development, Testing and support teams	**Teams**	Cross-functional teams, Suite Coordination/Execution teams
Management driven, Phase gates	**Planning/Milestones**	Team-driven,time-boxed end-to-end deliveries
Compliance oriented, Top-down	**Processes**	Value/Waste driven, Both bottom-up and top-down adopted by experimentation and measurement.

Conclusion

If you belong to a large, distributed and multi-product organization, chances are that your challenges and goals are similar to ours. Some of what worked for us may work for you. Or save you some false turns. Your best answers, however, will be different and unique to your context. But I believe the core principles and processes would still work for you regardless of your context and where you are in the agile journey. Pick an idea that resonates with your from here or elsewhere. Test it in a controlled time boxed experiment. Measure. Adopt what works and discard the rest. Rinse. Repeat.

The journey of Lean Adoption and Continuous Improvement is challenging and unique to each context, organization or culture. It is also immensely rewarding and valuable to the business and for personal and organizational and the business growth. Bon Voyage!