The Green Book:

Collection of Independent Essays

by Gene Gendel

Last updated: August, 2019

Table of Contents

About the Author

Gene Gendel is Agile Coach, Trainer and Organizational Design Agent. Gene is a proud member of the small community of Scrum Alliance Certified Enterprise Coaches (CEC). Gene's goal is to help organizations and individual teams with improving internal dynamics, organizational structure and overall efficiency.

Gene strives to engage at all organizational levels: senior and midlevel management, teams and individuals. In his work, Gene uses various methods, tools and techniques to strengthen learning of others and to ensure that teams and individuals gain autonomy after he "coaches himself out of the job." Throughout his long career, Gene has served small, mid-size and large companies domestically and abroad. Gene is a well-recognized member of global and local Agile communities where he influences people via open-space Agile collaboration workshops, coaching retreats, group events and presentations. Gene strongly supports Scrum Alliance (SA) in its efforts of "transforming the world of work." He is an active member of SA working group of coaches and trainers that have been involved in improving SA certification/education programs by aligning them with natural career paths of Agile professionals: Enterprise Level Coaching Certifications (CEC) and Team Level Coaching Certifications (CTC). (Gene is also one of co-creators of the program.)

Gene's additional credentials include the following:

- Certified Team Coach (CTC)
- Certified in Agile Leadership (CAL)
- Certified in Large Scale Scrum (CLP)
- LeSS-Friendly Scrum Trainer
- Certified in Scrum @ Scale (S@S)
- CSM, CSPO, CSP, PMP

Here is the list of Gene's additional focus areas:

- Organizational/System design
- Enterprise-wide
- Scaling Agile solutions/frameworks
- Coaching Leadership, Scrum Masters, Product Owners, Teams

Gene's website: www.keystepstosuccess.com

About the Book

This book is a collection of independent articles, written by Gene Gendel, over the course of five years, from 2014 to 2019, reflecting the author's real-life experience of working and consulting for various companies as an organizational design agent, trainer, coach and mentor.

The articles cover the following areas and domains:

- Organizational Agility
- Organizational Design
- Corporate Psychology
- HR & Finance
- Agile @ Scale
- Agile Tools and Techniques
- Agile Budgeting and Finance
- Agile Metrics
- Agile Teams Dynamics
- "Wagile" Patterns
- Agile for Non-IT
- Product Ownership
- Other Miscellaneous

To provide some continuity of reading independent articles, they are logically grouped, based on subject, irrespective of original publication dates.

The articles appear in exactly the same format and with the same content as they were originally written and published on the web—with some minor editing and graphic modifications to fit layout and publishing requirements.

Each article is accompanied with web reference (URL), indicating where the original publication appeared.

About Coaching

Guidelines to Hiring a Professional Coach

Originally published on July 10, 2019 | Location: http://www.keystepstosuccess.com/2019/07/guidelines-to-hiring-a-professional-coach/

Let's face it. Today, finding an experienced and credible Agile coach, is not easy. If you disagree with this statement, you are either very lucky and have special access to some great talent (e.g., referrals or networking) OR your perception of the role may need to change.

There is no need to be ashamed of not being able to find a good coach. You are not alone. Many companies face the same challenge.

Truth be told, *unfortunately*, the industry has changed significantly over the last few years and become the source of many problems. (Some very classic problems are described here). Today, the term "Senior Agile Coach" has been grossly diluted.

But *fortunately* there are still great standards and guidelines you can follow when looking for an Agile coach, irrespective of industry trends. Please consider the dimensions below when looking for a professional Agile coach for your organization. The original sources of these requirements are listed at the bottom of this page, and you are encouraged to explore them for additional details.

Please, do not reduce, simplify or trivialize some of the key expectations of a professional Agile coach, because if you do, the following two problems will follow:

- Industry coaching quality (average) will be further decreased. Even if you don't care about this fact as much, you will care about the next fact.
- Quality of service to your own organization will be also low

With that…

"Must-Haves" for Professional Agile Coach

Quantitative assets:

- Has significant hands-on experience in at least one of the roles on a Scrum Team.
- Has coached multiple organizations, departments or programs.
- Has *at least* 1,000 hours of experience coaching at the enterprise/organizational level or a combination of enterprise and multi-team level coaching.
- Has diversity of coaching experiences that can be demonstrated using different client engagement examples and which include experience at the organizational level.

Demonstration of deep knowledge:

- Has formal and informal education about coaching and strong mentor relationships.
- Has good working knowledge of Agile and Lean values, principles and practices.
- Has helped individuals, teams and leadership to understand and apply Agile and Lean values, principles and practices effectively.
- Understands the dynamics, patterns and development of multi-level teams and how they interact at the organizational level.
- Knows the difference between consulting and coaching and knows when to apply each.

Ability to clearly articulate and substantiate one's own:

- Coaching career overview (coaching, Agile history and how a person got to where he/she is today, including key milestone years).
- Coaching focus (summary of a person's professional self today, including a coaching approach and/or philosophy to coaching).
- Coaching goals (personal development goals in coaching).
- Formal coaching education (formal education activities that have contributed significantly to one's coaching journey. This includes a wide range of courses on topics, including facilitation, leadership, consulting, coaching, process, tools, techniques, frameworks and other related activities which have influenced a person's coaching practice).
- Formal mentorship education (coach mentorship and significant collaboration activities where a person has DEVELOPED a skill or technique or RECEIVED guidance to his/her coaching approach and mindset).
- Informal coaching learning (significant topics a coach has studied outside of the Scrum literature which has impacted his/her coaching approach or coaching philosophy).
- Agile community participation (Agile community events, such as user groups, gatherings, retreats, camps, conferences, etc., in which a coach has participated).
- Agile community leadership (leadership contributions to the Agile community (e.g., writing, publishing, presenting, facilitating, organizing, training and other activities) through events, publications, courses, blogs and forums).
- Agile community collaborative mentoring and advisory (significant collaborative Agile mentoring, advisory activities, where a person was mentoring, advising other individuals to increase their competency or in development of a specific goal).
- Coaching tools, techniques or frameworks known (coaching tools, techniques or frameworks which one has implemented, customized, co-developed or developed in one or more client engagements).

Skills, tools and techniques:

- Has contributed to significant improvements in organizations or departments through coaching techniques.
- Has helped organizations and teams beyond the basics of Scrum theory and practice.
- Has enabled organizations to find their own solutions to business problems through the application of Agile principles.
- Is familiar with, promotes and embodies the mindset of Servant Leadership.

- Uses a rich set of facilitation, training and coaching tools and models.

Personal Qualities:

- Coaching mindset-coaching skills/practices and frameworks.
- Evidence that the coach has taken both their experience and learning and synthesized these into definitive practices, frameworks, approaches and strategies).
- Self-awareness: able to reflect on their own contribution to the coaching by virtue of their own "being."
- Constant learning: has and continues to acquire coaching-oriented learning through multiple dimensions.
- Diversity of experience with different types and sizes of organizations
- Participation in the Agile community.

Note: Your company needs to have internal expertise to validate a person's ability to be a coach, based on the above.

Resources:

- SCRUM ALLIANCE® CERTIFIED ENTERPRISE COACH℠ READINESS CHECKLIST
- SCRUM ALLIANCE® CERTIFIED TEAM COACH℠ READINESS CHECKLIST
- Certified Enterprise Coach (CEC) Application – SAMPLE
- Scrum Alliance Certified Team CoachSM (CTC) Application — SAMPLE
- Summary of (CTC & CEC)

Centralized vs. Decentralized Coaching

Originally published on May 22, 2018 | Location: https://www.infoq.com/articles/centralized-decentralized-coaching

Key Takeaways

- There is a frequently seen confusion with respect to the definition of Agile coaching: coaching focus (e.g., enterprise vs. team) is confused with coaching alignment (centralized vs. decentralized) within an organization
- Centralized coaching departments run the risk of turning into single-specialty organizational silos that are locally optimized for their own expansion and personal success; they are also removed from real action—the reasoning behind this: standardization has its weaknesses
- Centralized coaching is often limited to being "responsible for introducing KPIs, documentation of "script-style-one-size-fits-all best practices and cookie-cutting approaches." This leads to system gaming by other departments and organizational silos that must "meet numbers goals"
- Centralized Agile coaching makes sense only when it takes place within an organization that is small enough to be effectively managed front-to-back (including all its organizational layers) and is genuinely supportive of its own coaches by providing them with "organizational immunity" and operational safety, enabling them to perform their challenging duties
- The main advantage of decentralized coaching approach is that coaches are close to real action: deeply engaged with products/services, and they are intimately engaged with senior leadership. Decentralized coaching is deep and narrow (as opposed to being broad and shallow) and takes time to cause meaningful and sustainable organizational changes

"The old rules no longer apply...

"When General Stanley McChrystal took command of the Joint Special Operations Task Force in 2004, he quickly realized that conventional military tactics were failing. Al Qaeda in Iraq was a decentralized network that could move quickly, strike ruthlessly, then seemingly vanish into the local population. The allied forces had a huge advantage in numbers, equipment, and training—but none of that seemed to matter....

"A new approach for a new world...

"McChrystal and his colleagues discarded a century of conventional wisdom and remade the Task Force, in the midst of a grueling war, into something new: a network that combined extremely transparent communication with decentralized decision-making authority. The walls between silos were torn down. Leaders looked at the best practices of the smallest units and found ways to extend them to thousands of people on three continents, using technology to establish a oneness that would have been impossible even a decade earlier. The Task Force became a "team of teams"—faster, flatter, more flexible—and beat back Al Qaeda."

Original source: Amazon summary of the book "Team of Teams," by General Stanley McChrystal (Author), Tantum Collins (Author), David Silverman (Author), Chris Fussell (Author)

Note: This writing was inspired by the discussion among LeSS trainers (CLT) and candidates, LeSS-Friendly Scrum Trainers (LFST), Certified Enterprise Coaches (CEC) and Certified Scrum Trainers (CST). The main influencers of this writing are Rowan Bunning, Josef Scherer, Greg Hutchings, Michael Mai, Robin Dymond, Viktor Grgic, Bas Vodde and Gene Gendel. Points of view

are drawn on experience of individuals' consulting at various companies in the following industry sectors: Global Telecommunications, Finance/Banking, Insurance and Department of Defence.

Resolving the Confusion: Focus vs. Position

While this writing is about the differences in effectiveness of centralized coaching vs. decentralized coaching in complex organizational settings, first, we need to clear up one important misunderstanding: mistakenly, enterprise-level coaching is confused with *centralized*, while team-level coaching is confused with *decentralized*.

This is a confusion of two different coaching aspects: *focus* and *position*. For example, within an internal organizational structure of the same CTO (could be different from other CTOs of the same organization), there could be both levels of coaching: enterprise- and team-level coaches—they would be working concurrently while complementing each other's work in many ways. What makes them different is their coaching focus (enterprise dynamics vs. team dynamics). But at the same time, their placement/sense of belonging is the same: they are decentralized from an enterprise apex and fit into a more local area (sphere of influence of one CTO).

To summarize the definitions of two coaching aspects:

Coaching focus:

- **Team** coaches are primarily focused on tools, frameworks and dynamics of multiple teams, with less emphasis on organizational transformation.
- **Enterprise (organizational)** coaches are more focused on organizational dynamics and more abstracted elements of transformation, with emphasis on senior leadership, upper management, organizational policies (e.g., HR) and multiple organizational domains.

Both focus areas, enterprise- and team-level, are equally important and required for transformational success, irrespective of where a coaching discipline is placed: centrally or decentrally. Note,, many experienced coaches are able to operate equally effectively at team- as well as enterprise-level, as they "travel" up and down an organizational vertical. (

Note: for more detailed definitions of coaching focus areas, please refer to authentic and credible sources describing this profession: Scrum Alliance and LeSS.works.

Coaching position:

- **Centralized**—a distinct organizational unit (e.g., Agile Center of Excellence or Agile Global Centre) that drives Agile transformation across an entire organization by introducing best practices, tools, techniques, standards, benchmarks and scorecards, against which everyone else is measured. Such organizational unit is loosely coupled with any specific product, service or line of business. It is primarily supported (and sponsored) by an organizational structure that has been selected and "put in charge" by more-senior leadership. With this selection, other organizational

structures (e.g., operations or product groups) usually remain less vested in the effort, and even if follow along, do so with noticeable complacency.

- **Decentralized**—a less rigidly structured team of like-minded coaches that align themselves with a clearly defined product, service or line of business. Transformational focus here is much narrower and requires much more genuine support/vesting (with sponsorship!) from multiple organizational verticals involved (e.g., business, operations, IT, HR, finance, etc.). In order for decentralized coaching to have a meaningful organizational impact, an organization must be of manageable size (no Big Bangs), as it is defined by an organizational sushi roll that contains elements/instances of multiple organizational structures involved.

Let's take a closer look at two distinct types of a coaching position:

Centralized Coaching

Usually, this approach is preferred by organizations that treat Agile transformation as a trend, accompanied by inspirational slogans, PR and town hall talks with senior organizational leaders rendering support mainly "in blessing and in spirit" rather than by real actions. This is frequently done without real understanding of deep system implications of this important undertaking. There is no true gemba by senior leadership. Actual transformational efforts are delegated downward-and-downward, to lower echelons of power where the original purpose is diluted and focus is lost. With With Agile coaching, being a centralized organizational function that owns transformation, one of its main deliverables becomes setting of standards and measures of success, by which the rest of an organization is measured.

A usual justification of centralized coaching function is the need to standardize and define "best practices" for others by claiming that "...too many independent adoptions would be hard to measure/compare against another…so we need consistency." That justification is something that is very important for organizations, where individuals' rewards and compensation are based on individual performance, scorecards and KPIs. To fulfill organizational mandates for measurements, centralized coaches are tasked with introducing agile maturity metrics (AMMs) that are usually composed of a wide array of maturity indicators, bundled together in some arbitrarily created maturity buckets/levels. **This approach very quickly turns into a box-checking exercise for other organizational units, whereas everyone tries to claim higher maturity in order to meet goals. Not surprisingly, this is accompanied by system gaming and unsubstantiated claims of success.** Since with centralized coaching approach, there is a higher volume of demand for coaches and it often exceeds a supply, quality is frequently compromised, and it manifests itself as follows:

- Larman's Law of Organizational Behaviour # 4 kicks in—it describes individuals whose past roles have become less needed in flattened/leaner organizations, and now "...coaching seems like another thing these misplaced folks can do successfully." These individuals perceive coaching as an opportunity to stay busy and fast-track their own careers (Agile coaching for them is just a

"hop-on-hop-off" bandwagon) until they secure another comfortable position in an organizational structure.

- Coaches "Centaurs"—describes low quality external consultants that are hired temporarily from outside at low/wholesale cost through third-party vendors, conveniently listed in preferred vendor applications of client-companies. (**Note**: usually, these vendors have challenges supplying high-quality intellectual assets in general, let alone Agile transformation consultants, as the latter are relatively specific professional niche).

Since centralized coaches are responsible for setting the tone for the rest of organization, they are also tasked with producing large volumes of supportive documentation: standardized training materials, audios, videos, etc. Hundreds (at times, thousands) of internal wiki pages are created to host information that, for the most part, **comes in the form of copy-pasting what is already available publicly on the internet (easily accessible commodity)**. This consumes many man-hours and produces a false sense of information ownership, internally. Such information also becomes outdated frequently and requires lots of internal manual rework to be kept up to date.

As demand for coaching comes from various organizational areas, centralized coaches get temporarily deployed to offer assistance. But since for many internal customers, Agile transformation is still a meeting-numbers game (to comply with enterprise-wide organizational mandates), and the demand for coaches often exceeds its supply (it comes in spikes). As a result, often coaches are spread thin across multiple organizational areas and their ability to truly make long lasting, meaningful impact is hindered. Coaching becomes broad and shallow. In a long run, as a delayed result of demand-spiking, there is an accelerated growth of centralized coaching groups, as described above.

… And now, the artificially inflated group of centralized coaches takes on the form of a single-functional specialty department that is governed by "local optimization": they are optimized to maintain their own increased size and the need to stay busy.

Here are some quotes about centralized coaching from the influencers of this writing:

From Viktor Grgic of Odd-e:

> Organizational tree of Agile coaches who commonly force upon others their "services" is quite serious dysfunction. If organization is very much into this, one might choose to limit scope of adoption, show real result while others are extremely busy with programs, etc. In other words, there is not much that can be done when KPIs for Agile transformation are set at the very high level of organization and everyone is busy complying with them.

From Greg Hutchings, of Amelior Services:

> I would discourage those who think that the best use of Agile coaching and training budgets would be to create an Agile center with people primarily aligned with and focused on belonging to and spending

time with a separate, specialist group, as this is just about the exact opposite of go-see, working at gemba, and inspecting and adapting with people in the main value creation part of the organization.

Leaving your men behind

One of the most painful examples of centralized coaching dysfunction is having coaches internalized/commoditized by an organizational structure that does not genuinely support Agile transformation, because it neither understands it, nor sees any personal benefit of it. Furthermore, there is a fear that organizational agility at-large (at scale) might be viewed as a threat to the very purpose and usefulness of an organizational structure itself. For example, placing a coaching function within an existing Business Analysis group, or an existing Governance CoE, or management CoP/PMO or Architecture department would be a disservice to an Agile coaching initiative altogether, as these organizational verticals would not provide coaches with necessary support and safely to perform challenging duties. They will leave their coaches behind. For example, if coaches reveal organizational dysfunctions that may lead to an increase of political tensions and unsupportive organizational structures, then Taylorian managers of modern days will readily sacrifice their own coaches ("throw them under a bus") to regain political recognition to better fit an organizational landscape.

Decentralized Coaching

With a decentralized coaching approach, coaches are locally aligned/dedicated with teams, their customers and products, and they immediately involve senior leadership.

This approach is usually preferred when a specific organizational area (e.g., IT, product development) makes a **conscious decision** to improve its agility/adaptiveness. Decentralized coaching is typically sponsored/supported by a real end-consumer, with enough organizational power to protect autonomy and authenticity of original transformation goals (e.g., CTO/CIO and respective senior business partners). In this scenario, people that consume services and people that pay for services **are the same people that are really vested in success.**

With decentralized coaching, fewer but more experienced and dedicated coaches are required. Coaches are more carefully selected by an organization. Coaching seasoning/experience is being viewed as the most important factor and becomes a natural remedy against a "wholesale" approach: highly qualified coaches will not work for a discounted pay, while genuinely vested clients are willing to pay a fair price for high quality service.

Decentralized coaching is deep and narrow: it is focused on fewer people (in total) but on a wider gamut of organizational elements/domains (e.g., IT, business partners, HR, finance) by taking a holistic look at the whole organization. With this approach, it is much easier to trace effectiveness of coaching "from concept to cash" by chasing not just superficial indicators/outputs (e.g., adherence to

Scrum events, increased velocity, stable/collocated teams) but also by seeking true business outcomes (increased ROI, improved customer satisfaction, beating external competition, team happiness).

Once engaged deeply, dedicated coaches go through a few important steps of a coaching cycle: assessing, delivering structured training, coaching and gradually disengaging while giving autonomy back to a client. There is no rush to meet year-end target numbers with coaching. Dedicated (local) coaches and organizations they support, are much less preoccupied with KPIs, metrics, scorecards and meeting numbers. AMMs are treated solely as a barometer of local improvements (please see how). Throughout an entire engagement period, coaches remain deeply embedded with development teams and respective product teams. This is all accompanied by many observations, short feedback loops and frequent retrospectives. Any issues or observations that have systemic implications and require attention of senior leadership are addressed together with senior leadership.

Given autonomy and sovereignty of an organization and its dedicated coaches, there is a higher chance of running experiments, inspecting and adapting without fear of failure or being prematurely judged and becoming a subject to repercussions.

Why False Dichotomy?

Sometimes, we hear a concern that because of a decentralized coaching approach, there will be no adequate shared learning across the whole organization. But why should this be the case? Why should decentralized coaching and shared learning be mutually exclusive (**false dichotomy**)? Could there not be some other effective ways to ensure that coaches succeed in both: to remain dedicated to their own, distinct organizational areas, for the reasons described above; and still be able to collaborate, synergize, learn from one another and create full transparency for their individual methods and styles?

All of this could be very effectively achieved by forming self-organized/self-governed coaching communities of practice where coaches from different organizational areas of different focus (team, enterprise) and with different skill set (technical, career, process) consistently share their knowledge and experience in a safe, reporting-free environment.

Conclusion

If an organization is relatively small and centralized coaching does not turn into a PR showcase with a small group of privileged individuals trying to set a tone for thousands of others by enforcing KPIs, metrics and best practices, **AndIf** there is a way to prevent centralized coaches from rushing towards year-end target numbers, **AND** instead engage deeply and narrowly with clients while offering continuous support and conducting safe experiments, **THEN** centralized coaching approach is worth a try. **Endlf**

However, if the above conditions are not possible because of historic organizational malfunctions, then dedicated coaches that are deeply embedded with vested customers is a better choice.

Here are some more quotes from this writing's influencers:

From Bas Vodde, of Odd-e and the co-founder of Large Scale Scrum framework:

> If you have a centralized team that can *truly* go to products and coach them, they're very valuable as they see a lot of cross-product dynamics. If they can't do that, then a decentralized group is a better chance of getting at least some value out of the coaching.

From Rowan Bunning, of Scrum WithStyle:

> If the goal is for Agile thinking and practices to be disseminated throughout an organization in a way that everyone feels that they "own" their ways of working, then coaches should be co-located and deeply embedded with teams and business units they support. Furthermore, if a message from coaches to developers is to move from single function groups to cross-functional teams, then having coaches centralized into a single-function group may seem hypocritical. It may also be perceived as a pursuit of control over how Agile coaching services are procured and disseminated—potentially to the benefit of those in a centralized group.

Note: By way of illustration, in LeSS adoptions, a LeSS coach would focus on two to eight Teams (about 50 people) that work on the same product for the same Product Owner out of the same product backlog. A LeSS coach would also focus on additional organizational layers that are in immediate proximity to the IT-side of a LeSS organizational construct, specifically, on a product/product-people side and their respective senior leadership. LeSS coaching engagement is meant to be deep and narrow (as oppose to broad and shallow), and is focused on a small "sushi roll slice" of a whole organization. For LeSS coaching to be successful, bigger does not mean better, and it naturally supports the idea of organizational descaling.

"Who are the Judges?" Who Decides Who Is Gonna Coach?

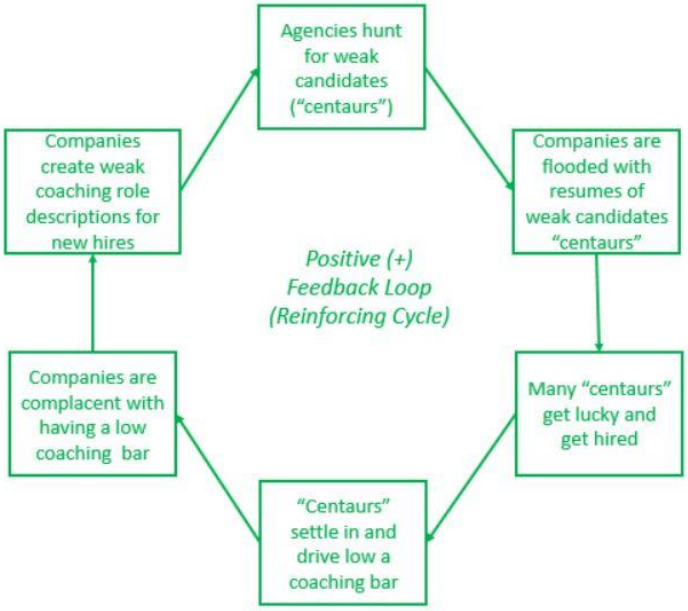
Originally published on August 7, 2017 | Location: http://www.keystepstosuccess.com/2017/08/who-are-the-judges-who-decides-on-who-is-gonna-coach/

Lets kick off this post with the quote from another recent discussion that generated a number of strong comments from experienced professionals:
"…as long as companies remain complacent and reliant on outlived staffing/head-hunting approaches, cold-calling techniques, and ineffective HR-screening processes, performed by people that poorly understand the ***essence of an agile coaching profession, while trying to procure cheap "Agile" resources (using "preferred vendor lists") or treat seasoned professional coaches as "requisitions to be filled," a coaching bar will remain low, and companies will be getting EXACTLY what they have paid for" (coaches-centaurs (p.17)).***

To summarize, the purpose of the above referenced discussion was to increase awareness about implications of ineffective coaches and coaching that exists in abundance today. Here, let's look at some root causes why this problem exists.

Who defines the role of Agile coach?
For the most part, organizational understanding of a coaching role is weak. Definitions of a coaching role that flow around suggest that companies are still confused about what coaches do. Definition of a coaching role is frequently lumped together with the role of a project manager, team lead, business analyst, Jira/Rally/VersionOne administrator, etc. While some of these *other* roles could represent potentially relevant past experience for a coach, lumping all of them together in one all-inclusive role description, delimiting them by commas or forward slashes is ironic, to say the least. Many of these "ace pilot/submarine captain/NHL star" roles create a conflict of interest not just for people that step into them but for everyone else who gets affected by interaction. Very often, inaccurate definition of a coaching role leads to inappropriate behaviors by a coach, such as attempts to seek authority and organizational power, exhibition of command and control behavior, competition with people being coached for ownership of deliverables, monetary incentives and other perks.

Once a poorly defined coaching role description hits the street, it enters a vicious cycle—reinforcing feedback loop (described in detail here).



Note: The above illustration excludes other system variables that may have an effect on the variables and the variables' relationships shown above.

This vicious cycle usually leads to one inevitable result: over time (usually months, sometimes a few years) companies realize that Agile coaching did not bring about enough sustainable organizational improvements, as it was expected. This further leads to two outcomes, both of which dependent of senior leadership vision and goals:

- Companies seriously re-assess their own initial actions, acknowledge mistakes made, and then improve coaching standards and elevate the bar in favor of real, experienced coaches

- Companies try to water down mistakes they have made, trivialize a coaching role for a lack of its benefit and, and by doing so, further reinforce the loop above

Who really makes decisions and why?
Rarely, senior executives take an active role in a coaching hiring process; exceptions exist but they are rare (usually exceptions are seen when things become very urgent—page 14). But even when they (executives) do engage in the process, it is usually more the act of a formality to ensure that a hired person "fits the culture." Of course, and very ironically, one of the key expectations from an experienced coach should be to challenge an organizational structure (both at enterprise and team level), and since culture is corollary to structure (Larman's Law # 5), the latter would change (would be challenged) as well. But this is not something that too many senior executives would like to hear. For the most part, a hiring process is delegated to first- and sometimes second-line management, as well as internal Agile champions that oversee and *own* Agile transformations. While Larman's Law # 1, historically, has defined the attitude of middle

management towards fundamental changes that challenge a status quo, the recently added Law # 4 neatly describes "contribution" by some internal Agile champions. And while exceptions do exist, trends and statistics speak louder.

Let's imagine the process by which an organization wanted to hire an Agile coach (as employee or consultant—no difference):

In this process, the interviewers are individuals described in Larman's Law # 1 and # 4. On the other hand, an interviewee is a seasoned Agile coach with a long enterprise- and team-level track record: she is a system thinker, dysfunctions challenger, a real organizational change agent.

Impact on a hiring process by Larman's Law # 1-type interviewers:

At an interview, a coach-candidate meets with first- and/or second-line managers that also expect that a coach will report to them when she joins a company. During a discussion, interviewers hear from a coach certain things that coaches usually bring up, uninhibitedly:

- Simplified overall organizational structure where developers receive requirements and communicate on progress by interacting directly with end customers, not middle-men
- Flattened team structure, where developers self-organize and self-manage. Overall reduction of supervision and resource management in favor of increased autonomy, mastery and purpose by individuals that do the work
- Harmful effects of individual performance appraisals and subjective monetary incentives, especially in environments where team commitments and team deliveries are expected

Unsurprisingly, the biggest question that many interviewers walk out with after interviewing such a candidate is this: "*What will my role be like if this coach is hired and brings about above mentioned organizational changes?*"

Impact on a hiring process by Larman's Law # 4 -type interviewers:

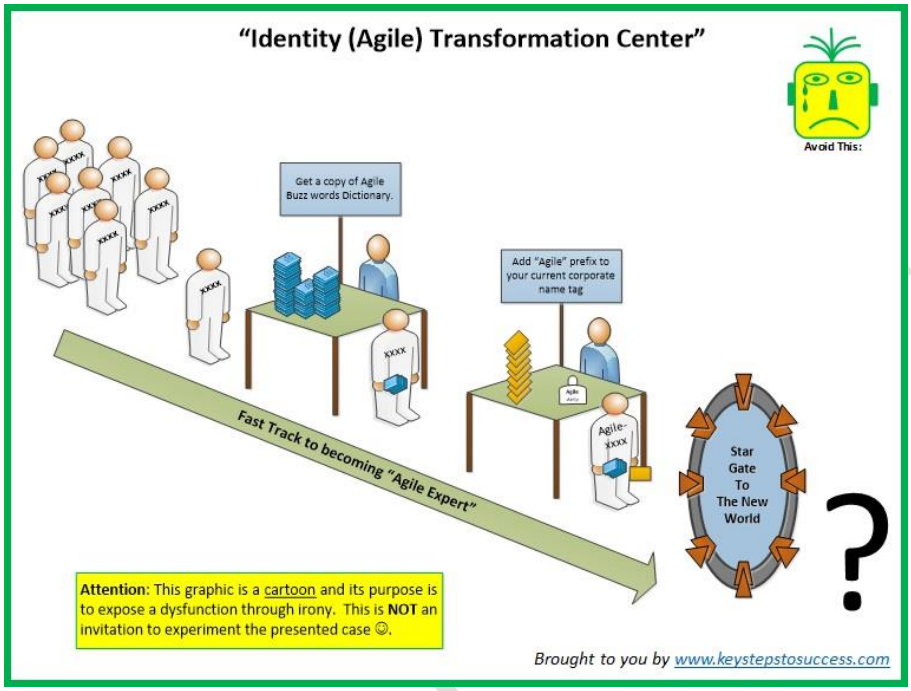
Knowledge and experience of a coach-candidate supersedes that of internal Agile champions and *process owners*. Some of the discussions a coach elicits—and answers provide—far *exceed expectations* (not to be confused with a term used in a performance appraisal process!) of her interviewers. Some suggestions and ideas shared by a candidate are great food for thought for senior executives but not at a level where Larman's Law # 4-type coaches are authorized to operate. Interviewers clearly see that a coach-candidate, if on-boarded, soon may become a more visible, influential contributor than the interviewers themselves. A coach may also bring about some organizational turbulence that will take out of comfort zone some individuals that are resistant to changes.

What are the odds that this experienced coach-candidate will be given a "pass"? What are the odds that she will be even given a chance to speak to senior executives involved in a hiring process, to attempt to influence them, to open their eyes, to offer a deeper system perspective on a situation, to make them think and talk about the forbidden?

Slim-to-none

And this is one of the ways, in which organizations that are complacent about Agile improvements, shoot themselves in a foot: they very effectively disqualify qualified Agile coaches and by doing so, reinforce the feedback loop illustrated above .

You Get What You Ask For: Agile Coaches-Centaurs



Originally published on July 9, 2017 | Location: http://www.keystepstosuccess.com/2017/07/you-get-what-you-ask-for-agile-coaches-centaurs/

Why are there so many failed Agile "transformations"? We frequently hear the following answer: "Because companies lack senior leadership support". **True**. And let's not trivialize this: Without strong and genuine support by senior leadership (beyond slogans and "support in spirit"), without selecting a deep, systemic approach to problem resolution, companies can only expect localized, peripheral and, most likely, short-term improvements.

But is there anything/anyone else that can be conveniently held accountable for failed agile transformations?

How about ineffective agile training and coaching?

Note: If you are interested in learning more about some of the most common challenges with Agile training, please visit this page. This post is about coaching.

There is a vicious cycle that hurts so many companies (and this can be also considered as a self-inflicted wound).

- Initially, companies set a low bar for coaches, based on poor understanding of a coaching role.
- Low quality coaches are hired (most of them are not even coaches, but rather people that have mastered Agile jargon and know how to impress HR and uninformed hiring managers).
- Weak coaches (most of whom have minds of conformists, not challengers) cannot effectively guide companies to fix systemic weaknesses and dysfunctions.
- Teams and departments don't really improve; rather they create a superficial appearance/illusion of progress (often, to impress senior management).
- Companies lose faith and stop seeing value in coaching.
- Companies start trivializing a coaching role.
- Companies decide not to spend more money on high quality coaching.
- Cheaper, even less effective coaches are hired (or internal, misplaced people are *refurbished* into coaches, overnight, as per Larman's Law # 4). Initially, the low-set coaching bar is lowered even further…and so on….

As a result, what was initially meant as a strategic organization- improvement effort, now takes on a form of just another system-gaming *change management fad* that ultimately leads to a failure and responsibility/blame-shifting.
What are some of the reasons why the above happens? Here are some suggested reasons:

- Companies don't understand the essence of Agile coaching role: it is viewed as another "turn-on switch" management function.
- Leadership does not feel a sense of urgency (p. 14) to make changes and exempts itself from being coached: people are too busy and too senior to be coached; they find coaching trivial.
- Certain organizational pockets are genuinely resistant to changes, afraid that changes can be brought about by real coaches (as per Larman's Laws 1-3).
- Market over-saturation with unskilled recruiters that *hunt* for low-quality coaches and contribute to the above cycle: this further lowers a company's chances to find a good coach
- *This list can be extended….*

Who is responsible for initiating this vicious cyclic dysfunction? Does it really matter if we identify guilty ones? Maybe it does, but only as a lessons-learning exercise. What probably matters more is how to break out of this cycle. Where to start: discontinue low-quality supply (coaches) *or* raise a bar on demand (by companies?). Usually demand drives supply and if so, a coaching bar will remain low **for as long as companies keep relying on body-staffing, head-hunting agencies and untrained internal HR-screeners to procure cheap "agile" resources or treating real professional coaches as "requisitions to be filled," and companies will be getting EXACTLY what they have paid for: coaches-centaurs (p.17).**

Big question
"*What should companies be looking for when hiring a coach?*"

An organization should be looking beyond of what is typically presented in a resume or a public profile of a candidate: usually, a chronological list of an employment history. Much more attention should be paid to the following important ***quantitative*** characteristics of a coach:

Coaching focus: What is an approach and/or philosophy to coaching does a coach have? This will help a company understand an individual mindset of a coach.

Coaching education AND mentorship: What active journey through education, mentorship and collaborative learning in coaching and related activities over significant period has a coach taken?

Formal coaching education: What has contributed significantly to a person's coaching journey, including courses on topics of facilitation, leadership, consulting, coaching, process, and other related activities which have influenced a person's coaching practice? Such education may not have to be degree-related. (Training and/or certification from any recognized institution could be sufficient.)

Coaching mentorship and collaboration: How has a coach developed a skill/technique or received guidance to a coaching approach and mindset? Respect and recognition of mentors matters here.

Informal coaching learning: What important topics outside of Agile/Scrum literature have impacted a person's coaching philosophy? This increases chances that a coach is well-rounded, beyond standardized book learning.

Agile community engagement and leadership: Does a coach engage in Agile user groups, gatherings, retreats, camps, conferences, as well as writing, publishing, reviewing, presenting, facilitating, training, mentoring, organizing and leading Agile events? An active participation and leadership in the agile community is a good demonstration that a coach has not developed herself within a unique organizational silo by self-proclaiming and self-promoting, but rather has diverse and "tested" industry experience.

Agile community collaborative mentoring and advisory: Does a coach mentor or advise other individuals (not for pay) on how to increase their competency or development? Is a relationship on-going, purposeful and bi-directionally educational?

Coaching tools, techniques and frameworks: Does a coach develop awareness and understanding of tools, techniques and frameworks while engaging with organizations? Has she customized or developed anything that was client/engagement-specific?

In addition to *quantitative characteristics*, here are ***qualitative*** characteristics of a good coach:

Coaching mindset

- How does a coach react when an outcome of coaching was different from what she had desired? In the past, how did a coach address this situation?
- How, based on clients' needs, has a coaching mindset needed to change? In the past, what compromises did a coach make? What was learned?
- What new techniques or skills did a coach learn to meet a client's needs?

Coaching competencies

- Assess—Discovery and direction
- Balance—Coaching and consulting
- Catalyze—Leadership and organizations
- Facilitate—Focus and alignment
- Educate—Awareness and understanding

Coaching specialties

- Lean/Kanban
- User Experience/Design
- Scaling Agile/Enterprise Agility
- Technical/Quality Practices
- Organizational Structures
- Lean Startup
- Product/Portfolio Management
- Organizational Culture
- Learning Organizations
- Non-Software Application
- Business Value/Agility
- Technical/Product Research
- Multi-Team Dynamics
- Organizational Leadership
- Organizational Change

Note: The above, is based on guidelines provided by Scrum Alliance application process for CTC and CEC.

While running some risk of sounding self-serving (very much NOT! the intent here): please, be mindful and responsible when you select guidance-level professionals in our Agile journey.

Agile Coaching: Lessons from the Trenches

Originally posted on: Aug 14, 2015 | Location: https://www.infoq.com/articles/agile-coaching-lessons

Note: This article was written with participation of my dear friend and colleague Erin Perry.

High performing organizations, high performing teams and high performing people do not often happen organically. They are a return on investment.

We've spent time in the trenches, both giving and receiving coaching at organizations of all sizes: from small startups to large enterprises. In this article, we will use our hard-fought experience to shed light onto Agile coaching. First, we will take a step back, helping define what being an Agile Coach means and what skills are necessary to be successful in an organization. Then, we'll examine patterns and anti-patterns for both in-house coaches and coach-consultants. We will shine light on how to enable coaches to be successful in your organization.

What is a coach?

Agile coach is an overloaded term. It's applied to advanced scrum masters, trainers, and leaders who aren't sure where they fit in an Agile organization. Agile coach is not a role mentioned in Scrum, Kanban, XP or any other Agile framework or practice. It's grown organically as larger organizations have realized the benefits of agility, and their appetite has increased for long-lasting change. Coaching can reap amazing rewards if done skillfully. What does a skillful coach look like?

Companies that rely on external Agile consultants want to know if they are acquiring good coaches with a proven track record and broad industry experience. Companies that prefer raising their own coaches want to identify the people with coaching aptitude. Individuals that pursue the career of an Agile coach wonder if they have what it takes to become a coach. Individuals that have established themselves in the role of Agile coaches wonder where the industry is taking the role. What is the future of Agile coaching as it becomes a broader role with a more diverse definition?

Definition through comparison—coaching or training?

Coaching and training are not mutually exclusive. Though many Agile trainers can also coach and many coaches frequently train as a part of coaching, the difference between the

two should be clear. In the comparison, the goals and role of a coach vs. trainer are highlighted.

Training

- Primary goal of training is to impart knowledge.
- Training is usually much shorter in duration: time-box is fixed and so is, typically, agenda.
- Training relies on hierarchical relationship, with Trainer being in a position of authority over the Trainees (holding expert and positional authority).
- Trainer is expected to be a subject matter expert in a given domain.
- Training is largely directive, providing Trainees with ready answers and solutions.
- Training provides short-term influence by Trainer on Trainee.
- Training can be done virtually/remotely, though it is less effective than classroom training.
- Training imparts a discrete set of skills. If more skills were required, additional training would be required.
- Comfort zone of Trainees is not frequently breached by Trainer. Training is mainly done agnostically of Trainee's personal experience (barring some interactive training).

Coaching

- Primary goal of coaching is to guide coaches toward self-improvement through observation and guidance.
- Coaching is usually much longer in duration. It is not strictly time-boxed. Coaching sessions may be shorter or longer, depending on how communication between coach and coachee continues.
- Coaching agenda is rarely fixed. Instead, it is responsive to the current needs of the coachee. Experienced coaches frequently use situational/opportunistic coaching to flex their style as needed.
- Coaching does not stress hierarchical relationship between coach and coachee
- Success of coaching is frequently dependent on Coach's ability to establish rapport and trust with coachee.
- Coach is not expected to be an expert in any one skill or subject area, but instead have broad experience in coaching as a skill.
- Coach's primary goal is not to provide coachee with final answers and complete solutions but rather enable coachees to derive their own answers and solutions by steering discussions and thinking.
- Past experiences and examples can be used by coaches in a non-prescriptive fashion, to help coachees develop their own associations and see analogies.
- Effective coaching is always done in person. Remote coaching is very ineffective as it does not have a personal element to it—a fundamental aspect of coaching.
- Coaching must be bi-directional. In most effective coaching sessions, coaches speak less and listen more. They reflect on what they hear/learn from coachees.

- Coaching has more impact on coachees; they develop their own ways/means to find solutions and address problems.
- Comfort zone (personal space) of coachees is frequently entered by coaches as the latter need to relate to experiences and sentiments of the former, to deliver more effective coaching. Experienced coaches either know how to enter personal spaces. delicately, or explicitly ask coachees to grant them permission. This alleviates negative effects of unwanted intrusion.

Coaching styles

A coach is constantly assessing where directive ("commanding") style coaching is required, balancing with supportive and reflective coaching. Finding the right situation for each style and properly transitioning between them are critical skills for healthy coaching.

Here are some of the typical conditions under which coach selects one style over another:

- Directive coaching
  - Coachee exhibits low ability and inadequate subject matter expertise for contextual learning; the coach has strong expertise in the subject matter.
  - Coachee has low motivation and morale.
  - Coach leads by example and expects coachee to follow.
- Non-Directive coaching
  - Coachee exhibits high aptitude, strong skillset and subject matter expertise, regardless of coach's skillset and expertise.
  - Coachee has high motivation and morale.
  - Coach reflects on what coachee thinks and says and makes coachee come to his own conclusion.

This approach is based on Control-Experience Tool (modified from Canadian Forces Leadership Doctrine by Alan Okros). The Coaching Style Dashboard is another valuable resource to balance the two styles.

It can be tempting, especially for naturally directive leaders, to fall too often into the directive route. It is, by far, the easier form of coaching. It is also less likely to leave a lasting impact on the coachee. A parent tells a young child not to run into the road and expects them to obey. So long as the parent is watching the child, they can reinforce the rule and ensure compliance. At some point, though, we must properly coach children to understand the impact behind a rule and to instill inherent motivations of safety and responsibility.

Taking a purely directive route will insure compliance, not engagement. The goal of any coach that begins directive should to be to move, as quickly as possible, the coachee on an axis that allows supportive, non-directive coaching.

Style behaviors

A directive coach

- Tells
- Provides answers
- Teaches
- Gives examples
- Offers advice

A reflective coach

- Asks
- Provides guiding questions
- Creates an environment for self-learning
- Gives learning resources
- Helps Coachees find their own vision and goals

Coaching specialties vs. coaching competencies

Coaching expertise can be measured using *Specialties* and *Competencies*. The Certified Scrum Coach (CSC) application by Scrum Alliance serves as a guide to defining these dimensions.

Coaching specialties

Coaching specialties are a core skillset, expertise and knowledge that coaches possess. To a large extent, they are based on a focus area of coach's paid and unpaid work (prior or present). Here are some examples of coaching specialties:

- Lean Principles, Lean Startup
- Design, Product/Portfolio Management
- Technical/Product Research
- Scaling Agile/Enterprise Agility
- Distributed Agile, Multi-Team Dynamics
- Technical/Quality Practices
- Development Operations
- Development/Process Tools
- Organizational Structures/Culture
- Organizational Leadership

Coaching competencies

Coaching competencies are proficiencies that coaches are expected to demonstrate in their interactions with individuals and their organizations. Here are some examples of coaching competencies:

- Ability to serve as an organizational mirror, by accessing and surfacing the underlying system problems. Ability to look below the surface, expose challenging symptoms and perform root cause analysis.
- Ability to facilitate client Agile adoption, implementation, and alignment. Ability to engage and facilitate stakeholders in controversial conversations and alignment-building activities. Ability to maintain non-biased views and facilitate collaborative decisionmaking.
- Ability to balance coach's own Agile expertise with coachee's (client's) goals and intent. Ability to understand and respect the nature of a client-consulting relationship whether as an employee or consultant. Ability to ask powerful questions, lead by example and guide client self-discovery.
- Ability to educate and guide coachee's (client's) agile learning through application and discovery. Ability to focus on stabilizing principles and varying practices to situationally align coachee's (client's) maturity with effective application of Agility.
- Ability to function as a catalyst and change agent for coachee (client) organization. Ability to engage in with the whole organizational system and the leaders who guide them. Ability to connect interdependencies and catalyze organizational reflection, learning and growth.

Levels of coaching

Agile coaching can be administered at various levels: organizational/enterprise level and local level.

When coach is involved *organizationally (systemically)*, the focus is on the following:

- Become more agile across an entire organization, trying to influence/educate senior leadership and executives.
- Assessing team(s) and organization(s) for effectiveness of applying Agile principles and practices.
- Advising and consulting with organizations and leadership on various Agile practices, such as Scrum, Kanban, Lean, XP.
- Facilitating team(s) and groups to achieve higher quality collaboration, enabling a culture of continual learning and knowledge dissemination.
- Developing team, leadership and organizational agility through guided self-discovery and growth.
- Advising teams on careful adoption of scaled Agile frameworks as mechanism for organizational descaling (e.g., LeSS, SAFe, RAD).
- Challenging the organizational and leadership status quo and enabling an Agile (Kaizen) culture.
- Analyzing systemic patterns, including norms, standards and behaviors

- Educating senior leadership on interconnection of various organizational elements within one Organizational Ecosystem. (If you cannot access the document please contact Gene for access.)

When coach is involved *locally*, the focus is on the following:

- Supporting single or multiple teams in improving their dynamics and maturity.
- Coaching individual team members, Scrum Masters, product owners.
- Assisting to establish Agile roles, ceremonies, day-to-day interactions.
- Focusing on engineering practices, coding standards, test quality.
- Advising teams on Agile requirements, living documentation, metrics, communication.
- Advising teams with adoption of basic Agile frameworks (e.g. Kanban, Scrum, XP).
- Challenging inappropriate locally manifested (in isolation) behavioral patterns.
- Balancing local optimization with team growth.

Coaching individuals vs. coaching groups

Individual coaching

Individual coaching is one-on-one. Such coaching sessions are typically conducted in privacy; the coach works with a single person on a very personal level. Individual sessions may address personal adaptation, happiness, job satisfaction, problems with management or subordinates, embracing roles and seeing career growth opportunities, dealing with personal challenges, reservations or fears. Individual coaching is often used to engage and support a Scrum Master or product owner as an individual.

Coaching is more conversational and personal and often takes a great deal of trust and camaraderie.

Group coaching

In Agile settings, group coaching is typically focused on entire feature teams or Product Owner teams, where people are expected to have shared beliefs, norms and goals. Group coaching addresses team dynamics, roles, day-to-day interactions, metrics, reporting, etc. Coach can set up a dedicated session for group coaching or leverage existing group ceremonies (e.g., retrospective).

Group coaching is often more structured and requires expert authority to be successful.

Both individual and group sessions can be prescheduled or situational/opportunistic (at moments, when coach finds ad-hoc appropriate moments to administer coaching).

Rules of coaching engagement and disengagement

Every coach (internal or external) needs to define and discuss with coachee (individual or company client) rules of engaging and disengaging. This is done for a variety of reasons:

"Rules of Engaging" between coach and coachee—it is important to define certain conditions under which coaching experience will be conducted. Sometimes, certain topics or points of contention can be avoided or negotiated. It is also a good Agile coaching practice to conduct a change readiness assessment prior to engaging. This helps identify certain risks or hard blocks for effective coaching and disclosing them to coachee (client). Every coach must be willing to walk away from a client that cannot establish agreeable rules of engagement, including readiness for transformation.

"Rules of Disengaging" between coach and coachee are no less important as they help with identifying appropriate time to discontinue (or lighten) coach-coachee relationship. This is done to avoid prolonged co-dependency, excessive transactional activity (compensation) for a diminishing value. Disengaging from coachee can be either done because coachee achieved a desired maturity or because of running into unresolvable obstacles that make continuous coaching ineffective and impractical. In the former case, coach may periodically conduct an Agile maturity assessment to gauge progress. In the latter case, coach may (and should) prematurely disengage from coachee but be clear about his reasons with coachee.

We recommend applying Agile principles to coaching engagements as well. Build an initial vision for the product (coaching). Regularly refine the backlog, taking time to reflect on the engagement and how it should be adjusted. Working in coaching sprints using Scrum or through regular delivery using Kanban help enforce the values and demonstrate Agility through example. It also provides regular touchpoints to determine if a new style is required or if the engagement should be halted. XP practices such as test-driven development can also be applied to coaching. Establishing testable criteria for coachee readiness first will help form the engagement activities towards demonstrable results.

The goal of any coaching engagement should be to bring the teams to a healthy state where learning and self-improvement are happening organically. The coach should be attempting to become unnecessary. Daniel Mezick, in his book *The Culture Game*, very effectively describes a coaching profession in scope of best coaching standards and coaching ethics.

Coach-consultants vs. full-time coaches

In his article "Unspoken Agile Topics," (section *Challenges with Agile Leadership*) written in 2013, Gene Gendel—one of the co-authors of this article—describes some of the most commonly known challenges faced today by companies as they rely on Agile coaching support.

1. Here are some challenging questions that have to be answered by companies that rely on help of external Agile experts:

- a. How will engaging with an Agile coaching consulting firm guarantee a top-notch Agile coach-consultant? (There is *no* guarantee that any company will deploy a good coach.)
- b. How will I ensure the coaching I'm receiving is working towards our independence and growth, not prolonging dependency to ensure further employment?
- c. How will an external coach address challenges unique to our culture and establish rapport and empathy with our teams?

2. And here are some no less challenging cases that arise when companies rely merely on internal resources:

- a. How will we avoid myopic views of internal coaches who may have limited experience with other cultures and companies and the creative ways they've solved problems?
- b. How will we support internal coaches to truly remain independent, with the freedom to challenge and question internal leadership without fear of jeopardizing their employment?
- c. How will we establish our internal coaches as experts, while avoiding the "prophet in his own land" perception?

An ideal situation would be for each organization to strike a happy balance by building out internal Agile coaching practice by mixing up external and internal Agile coaches. While external coaches bring to the table experience of other organizations and industries, holistic and uninhibited views, internal coaches contribute with deeper knowledge of their own organizational structure and culture.

Another challenge that organizations must face with regards to in-house coaches is how to give them an "**Honorable Discharge**" from duty when their service is no longer needed. This is less of an issue for employees that became coaches by transitioning from another role; once their coaching service is no longer needed, they may simply fall back into their previous roles (developers, Scrum Masters, etc.). This is much more of an issue for professional Agile coaches that were asked to join a company full time to help a company go through challenging times of Agile transformation.

When a company engages with a coach, it must have a strategy in place for how it will gradually progress from active coaching to self-direction and autonomy. A company must resist the temptation of having a coach take an authoritative, long-lasting position with a department or a team and becoming a long-term "doer," problem solver, and solutions provider. A company should also refrain from trying to "mold" a coach into a manager or auditor, rather than an organizational change agent.

Discontinuation of a coaching relationship must be done honorably, in a way that ensures that a company gradually builds up its own internal excellence. A coach must continue to be an asset throughout the entire engagement, even as coaching intensity starts to diminish naturally.

To avoid confusion and misunderstanding about what a coaching engagement is and what it is not, both coach and coachee (company, LOB, department, team, etc.) must thoroughly discuss and mutually understand the essence of a coaching role before engaging, as well as properly set each other's expectations.

Coaching solo vs. a coaching team

An organization may engage a single coach or a group of coaches that join as external consultants or company's employees. Some of the cases described below specifically address situations that arise when a company relies on its own internal Agile expertise (in-house coaches). In general, when external coaches engage with a company-client, they are perceived as representatives of an external coaching entity and "shielded" from some of the challenges described below.

Both solo-operating F/T coaches and team-playing F/T coaches are perceived by a company as employees first, coaches second. This means a company applies the same values, norms and evaluation standards to coaches as it does to the rest of company's employees. This creates a conflict of interest: Coaches must highlight challenges and impediments that may reflect poorly on their own employer. They are in the unique situation of constantly jeopardizing their employment and livelihood through the very responsibilities they have been given. How can a coach or a team of coaches perform their jobs effectively if there is an inverse relationship between quality delivery and their own safety?

Further, this situation is even more challenging for F/T coaches that operate as a team than for those that operate solo.

When a group of coaches operates as a team (shared goals and purposes, shared efforts, shared strategy and vision, collective ownership), but a company perceives and evaluates each coach as an individual ("I am a star") performer, it frequently causes a conflict of interest. As team players, coaches are expected to peer/mob-coach, cross-learn and cross-train each other, swarm (work together)—effectively, practice everything that they preach to an organization that they coach.

But there are strong forces that pull coaches apart. Rewards and incentives systems based on individual performance and achievements make them more preoccupied with their own well-being, with their own ability to advance within an organization. At times, there could even be an invisible competition between coaches that is caused by their

organizational positioning and relationship to each other. This produces silos and dysfunctions as it makes coaches turn away from each other, compete for span of influence, be unwilling to share and support each other as is expected from team players: morale deteriorates, transparency goes down and so does productivity. (See here more on alternative ways to offer incentives and rewards, based on collective performance. If you cannot access the document please contact Gene for access.)

This compounded effect of intra-coaching team dysfunction that sits below typically observed challenges that organizational coaches face day-to-day in the line of duty, significantly lowers value that coaches bring to an organization.

In the light of what has been described so far, here are some guidelines for organizational Agile coaches that are full-time employees:

- De-couple your organizational position and authority from your role of a coach. Organizational leadership does not equate to organizational coaching.
- Do not over-emphasize personal promotions: becoming a coach should not be treated as a "fast track" for personal career advancement.
- Offer objective guidance without personal or political considerations.
- As a coach, act as servant-leader, enabler and facilitator, not command and controller.
- As a coach, try to view an organization as an outsider; do not "take sides," based on a part of organization you belong to: this will completely compromise your impartiality and objectivity.
- Be resistant to micro-management and intolerant to wasteful processes, activities and roles.
- Finally, if you coach as a part of the team of coaches, remember about values and principles of collective ownership that you coach to Agile teams—always practice what you preach.
- If you are working on a coach team, apply the same principles you would to an Agile development team. Apply Kanban or Scrum, for example, and work as a team on the same goals. Make it clear that accomplishments and delivery belong to the *coaching team*.

Coaching "bad smells" and why they should be avoided

Below are some of frequently observed "Bad Smells" that are associated with bad coaching:

Bad Smell	**Why Does it Smell Badly?**
Continuously resolving clients'/coachees' problems for them. Engaging as "doers" for too long and continuously giving complete solutions. Exhibiting command and control behavior.	Initial "leading by example" is OK. Part of teaching comes through initial training. However, any prolonged engagement as a "doer" puts a client/coachee into a comfort zone and prevents learning, independence and autonomy.

	Commanding and excessive directive telling also implies that a coach has an authoritative position with a client. This makes a client less independent in his or her decisions.
Assuming authoritative, long-lasting position with client/coachee that does not translate timely into tangible results. Establishing dependence.	A coach ends up generating financial benefits, whereas a client gains very little value.
Publicly criticize/scrutinize individuals of any level, especially in presence of their superiors.	This behavior creates hostility and mistrust between a coach and individual clients/coachees. If a client/coachee feels comfortable to be openly coached in front of others, without becoming defensive, they should explicitly invite a coach to do so before it is done in public. This is a key sign of coaching immaturity.
Using team-based metrics to judge individual team members.	Excessive use of metrics is a simplistic false dichotomy. While certain health checks/indicators can be used as a way of reflecting to individual teams, using the same metrics to judge individuals is counter-productive and misleading.
Using team-based metrics to compare teams to each other, establishing competition between teams.	A prime example of local optimization, establishing cross-team competition will not optimize the performance of the entire organization and will instead encourage information hoarding and closed communication patterns.
Getting involved in activities and feedback that influence individual performance appraisals, incentives, compensation, bonuses, promotions.	Being able to explicitly influence compensation/financial well-being of an individual is a violation of an individual's safety space. A coachee's desire to become autonomous and independent in making his/her own decisions will be significantly diminished. Individuals will feel "obligated" to follow recommendations of a coach. Recommendations will be perceived as instructions/mandates
Quantify/numerically estimate ("status/checkbox") things that cannot be quantified. Applying a	Similar to the above: there is a huge human factor that is responsible for successful Agile

numerical scale (metrics) to human/cultural element of Agile transformation.	implementation. A human factor is not quantifiable and cannot be easily plotted on a scale. Checks and balances produce a false sense of completion (or lack of such). This diminishes opportunity for continuous improvement. Examples may include "Using Scrum" or "Converted to Agile" checkboxes for teams.
Tracking individual recommendations given to clients/coachees and assessing if recommendations are followed, reporting to senior management when they are not.	This is an indication of Command and Control, micro-management and lack of trust. Policing individuals and enforcing things to be done contradicts Agile principles and prevents Kaizen adoption. This also erodes relationship between a coach and client/coachee. Also, if a ratio of coaches to coachees is low (many coachees for a single coach), scaling coaching efforts becomes challenging. Ideally, in cases like these, a "pull" system must be used, instead of "push": coachees should pursue with coach, proactively asking to provide feedback to their improvements, instead of coach chasing them.
Withhold views and observations about pivotal organizational dysfunctions from organizational leaders to avoid personal risks and repercussions.	By being a change agent and organizational transformer, a coach is expected to speak openly about organizational dysfunctions and impediments that most of employees are not comfortable discussing. This is paramount for a coaching job. The job of coach is sometimes risky. Therefore, in majority of cases, a coaching role is consultative in nature (external to an organization), as coach must bring to light organizational dysfunctions that may put a coach in the position of scrutiny and/or criticism—something that consultants care less about than full time employees.

Are there better ways to teach?

Originally published on May 13, 2017 | Location: http://www.keystepstosuccess.com/2017/05/2017-agile-maine-day-recap/

Whether you are a high school teacher, a college professor or professional training instructor, you probably always look for ways to increase value you bring to a classroom. Some of the questions you might be asking yourself are these: "How do you enrich students' in-class experience? "How do you ensure information retention by students?" "How do you make in-class learning more applicable to real life?" This summary focuses on the following three aspects of teaching: dynamic teaching, teaching focus, feedback loop between teachers and students.

Dynamic teaching

Every instructor must have a set of Learning Objectives, based on which training content is built. Meeting these objectives deems successful training. But there are different schools of thought about educational learning:

Bloom's taxonomy classification model for educational learning (created by Dr. B. Bloom in 1956) implies that human thinking goes through six evolutionary (maturity) stages. If those stages were mapped to the Japanese martial art concept of **SHU-HA-RI**—describing the stages of learning to mastery—they would approximately group as follows: SHU (**Remembering**, **Understanding**, **Applying**) = "traditional wisdom"; HA (**Analyzing**, **Evaluating**) = "breaking with tradition"; RI (**Creating**) = "transcendence." With this thinking approach, to proceed to a next level of maturity, a person must pass through preceding levels. This type of learning is hierarchical/sequential and uni-dimensional.

An alternative and more dynamic taxonomy of learning has been proposed by L. Dee Fink, of the University of Oklahoma, in his The Power of Course Design to Increase Student Engagement and Learning. With this new thinking approach, instead of looking at learning as a hierarchical and sequential journey, we treat it as a multi-dimensional process, where each dimension is independent and can interact/overlap with other dimensions in a Venn-like style. The following are learning dimensions (categories) proposed by Fink: **Foundational knowledge, Application, Integration, Human Dimension, Caring, Learning How to Learn**.

All categories are independent of one another, and within each category students can advance to different degrees of maturity. Within each of the categories, there could be a critical minimum of learning objectives that must be met by all students—this is something that is decided by an instructor. Beyond this critical minimum, learning remains dynamic and conditional and is based on an instructor's assessment of in-class dynamics (which may vary from audience to audience).

Teaching focus

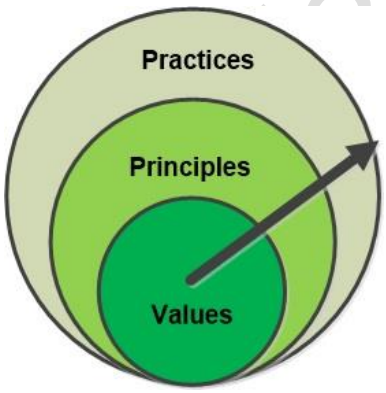
Truth be told, in comprehensive multi-session courses (e.g., college or university), where a professor has enough runway to build-up her audience for more advanced topics, there is a relatively low risk of short-cutting/by-passing basics in favor of practical learning.

On the other hand, in short, time-boxed professional training (e.g., a few hours or a few days) there is a higher chance that foundational learning could be shortened by an instructor in favor of topics that appear (only superficially) to have a more direct real-life relevance. In short training engagements, due to time constrains and a desire to jam as much information as possible in a session, we see these sacrifices primarily made because of the following:

- Instructors are pressured to deliver "maximum practical value for a buck" by their sponsors.
- Students attend against their will, with superficial goals to "rent" an instructor's immediate solutions instead of learning how to find their own.
- Certain "hot" topics that challenge current organizational values and norms are omitted to avoid inflaming discussions.

A good example of teaching focus loss would be an Agile training by an Agile consultant where a class immediately focuses on their day-to-day problems and "best" practices (e.g., metrics, tools, techniques and workflows), instead of learning Agile values first (e.g., human interactions, relationships, mindset, collaboration, compensation, etc.). [More information here about typical challenges with Agile training.]

By short-cutting to immediate practical implementations and offering ready-to-use "unwrap and install" solutions, trainers significantly reduce students' chances of retaining learning, developing autonomy and capability of creating and owning their own decisions (as opposed to "renting" from instructor).



Instead of working from outside-in (as per the diagram above), instructors should strive leading students from inside-out, by ensuring that students understand core values first, then build new principles upon values, and only then proceed to developing their own practices.

Teaching feedback loop

In 2014, in his "Don't Give Me Feedback," Tobias Mayer described how any type of direct feedback, whether positive or negative, is a judgment made by the giver on the receiver. Being a judgment call, feedback is always subjective and is anchored to a giver's personal and self-centered views and ambitions. Here is an example from a typical Agile training:

A positive feedback that is full of compliments, excitement and affirmation could mean that a student learned in class how her role will become more empowered, thanks to overarching organizational changes. This is a great reason to "celebrate" and give positive feedback to an instructor, even though the class itself was not so great! Another reason for a positive feedback could be that a student is trying to build a good relationship with an instructor for future "at-work" interaction and "special treatment" or with a hope that an instructor will provide her own positive feedback to students' superiors.

On the other hand, a negative feedback and criticism (this type of passive aggressiveness is sometimes seen in anonymous feedback forms) could mean that a student learned in class about something that will affect his personal daily work in ways that are not desirable by a student (e.g., required additional learning, loss of control or authority). So while learning itself is deep and clear, an individual's conclusions about personal consequences may lead to negative emotions and mental resistance and thus, a negative feedback.

According to Tobias, a much better way to receive feedback from a classroom would be by simple **observation**. Instead of soliciting students' feedback directly, an instructor should pay a lot of attention to in-class participation and interaction: student-to-instructor exchange, student-to-student exchange, questions and answers offered by students, students' desire to look for workable solutions that are acceptable by everyone, etc. A good way to increase objectivity of observation would be to re-shuffle students during training and re-create new working groups, seeing if in-class dynamics change, subsequently, as well.

Another big advantage of learning by observing is that it allows for an immediate adjustment of actions by an instructor and re-applying changes made back to the same group of students, without making it too obvious for students. For example, if an instructor sees one of her students being completely disengaged, she can ask a student to change to another table or request him to answer a question posed by another student.

To summarize, currently, with so much information becoming a free commodity available on the internet, unidirectional and "scripted" in-class teaching is becoming less and less effective. On the other hand, dynamic and interactive teaching, reinforced by short feedback loops between a teacher and students, will be setting high standards in future learning.

Agility in HR

NYC Salary History Ban: What Does It Mean?

Originally published on November 13, 2017 | Location: http://www.keystepstosuccess.com/2017/11/nyc-salary-history-ban-what-does-it-mean/

On **October 31, 2017**, the **Mayor of NYC Bill de Blasio** signed a new law prohibiting companies in New York City from asking, searching or verifying a job applicant's salary history during the hiring process. From now on, violation of this law, by any NYC-based employer, will be viewed as an "**unlawful discriminatory practice**." (Please, read more about the **NYC Salary History Ban.)**

Below are some potential consequences of this law as it applies to employees-candidates and *any* NYC-based employing organizations:

- If an individual has worked for a long time at the same company and, while employed, has acquired a lot of practical experience/skillset, but unfortunately was not able to secure compensation that was an objective reflection of her capabilities/expertise, she may now seek employment at another company without worrying that prior, unfairly low compensation will be a benchmark for her future offer.
- If an individual is a self-starter/entrepreneur who has acquired a lot of knowledge/experience in ways *other than* formal employment (e.g., self-paid study or research) and by doing so has significantly increased her professional maturity, she may confidently leverage these rightfully owned credentials when negotiating a salary with her next employer.
- If an individual's goal has always been to remain as a hands-on contributor (she loved what she did, and did not want to lose her practical skills) and never aspired to seek a promotional/managerial position—something that usually leads to higher compensation—she may do so more freely without worrying that she will miss out on a "compensation-bargaining-chip" at her next job interview. This also means that employees will be more experience/knowledge-seeking and less promotion-seeking since it is really an experience and not prior organizational position that define their true self-selling power. (**Note**: often, promotions are associated with loss of hands-on expertise in favor of managerial/administrative responsibilities.)
- If an individual's full compensation consists of base salary and discretionary bonus (the latter, often being too subjective since it is based on individual performance appraisals, the efficiency of which has been proven as ineffective for many decades), with a bonus representing a significant chunk of her full salary, she does not have to be concerned so much with her next employer trying to count in only her base salary as a benchmark, while making an offer. This will also, hopefully, drive companies towards paying higher base salaries and away from subjective bonuses.
- Recruiting agencies and staffing firms will have fewer opportunities to ask unethical questions ("How much were/are you making at your prior/current job?"), something that are often delegated to them by companies' HR departments, with the latter not wanting to be directly associated with unethical behaviors. Further, this may lead to more transparency and direct interaction between hiring managers and candidates.
- **Companies-employers would have to improve their vetting/interviewing/hiring approaches significantly by incorporating validation methods and more**

reliable/objective assessments of candidates to prevent under-qualified, low-skilled individuals (some of whom may have strong negotiations and "talking the talk" skills) from slipping through companies' doors and causing internal problems. This may require conducting more practical tests, real-life simulations and hands-on exercises administered directly by hiring areas and peers-coworkers. Further, this could also reduce an amount of subjective/administrative, error-prone and often unnecessary screening processes, usually handled by companies' departments that are *least* benefited from hiring high-quality candidates but at the same time most benefited from creating **and administering actual processes.**

In all the above situations, the main compensation-determining factors will be these:

- **From an employee's perspective**: her professional competency, skill/mindset, ability to produce tangible results and deliver business value.
- **From an employer's perspective:** ability to properly assess a candidate for what she is worth (not for what she was price-tagged in her past) AND clear understanding of how much an employer is willing to pay for a given job to a given candidate.

The natural question that comes to mind: Does the new law have any relevance to internal hiring situations (when employees move around a company)?

According to the **Employer Fact Sheet,** the law *does not* apply directly to internal re-employment (also, for most companies, employees' compensation is transparent to hiring managers of the same company).

However, there could be some indirect implication: NYC-based employers will probably realize that properly educated (know the Law) employees will have more confidence to ask a higher pay when they seek new employment internally. The new "compensation-bargaining chip" for employees will be their increased self-confidence that they will be able to get higher compensation elsewhere, irrespective of their current compensation (should their internal efforts not materialize). As a result, to avoid losing good people to their direct competitors, employers will probably revisit their compensation increase standards with regards to internal re-employments.

Grassroots of Modern Command & Control Behavior

Originally published on June 22, 2017 | Location: http://www.keystepstosuccess.com/2017/06/grassroots-of-modern-command-control-behavior/

Examples of Command & Control behavior can be historically traced back centuries, to the periods of dictatorship, imperialism, monarchy, feudalism and even further back to more primitive social systems. However, for the sake of this discussion, let's refer only as far back as the Industrial Revolution of the last century. Back then, workforce predominantly consisted of low-skilled laborers that performed routine, mundane physical work and managers-supervisors that were responsible for setting goals, assigning responsibilities, monitoring progress and praising-penalizing workers based on individual performance (using a carrots and sticks approach). This type of management was a classic example of what is known as Taylorian Management (Frederick Taylor), according to which there had to be clear delineation between individuals that performed work and individuals that controlled/managed work of others. This type of human relationship in work settings was also later described as Theory X Management (Douglass McGregor), and it suggested that managers needed to use totalitarian and repressive styles to ensure tight control over workers because the latter would otherwise not work hard and efficiently enough.

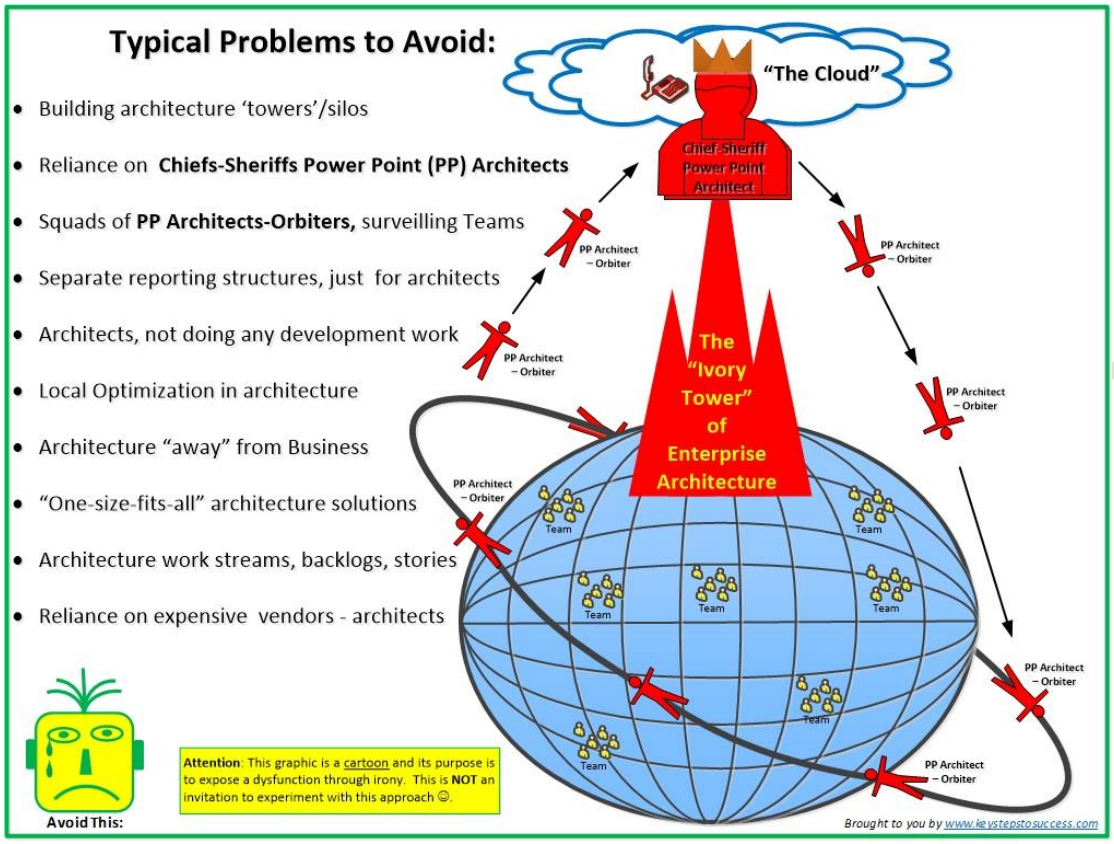
Fast forwarding to modern days…

Today we still have many examples of Command & Control behavior that shape relationships among people in modern organizations. This happens even in situations where organizations have workers that are very highly skilled and intellectually advanced. More frequently, this is seen in organizations that are at Laloux's Orange state of maturity or lower.

While in part, modern Taylorian Behaviorism can be explained by long-lasting "cultural inheritance" that hopefully will wear off over time, it would be interesting to look closer at some specific root causes of modern-day Taylorian behavior.

Although not exclusively, the examples below are more frequently observed between individuals that are related to each other by hierarchy (boss-subordinate).

Insecurity about own job. Worries about own career growth.
A manager does not feel secure about his own position. This could be caused by company reorganization (e.g., merge, acquisition, flattening) and a manager feeling that his role may be reduced or eliminated. This fear of becoming dispensable could be worsened by realization of personal incompetence and/or lack of professional knowledge. This is frequently seen in situations where managers, as they have progressed the hierarchical ladder, have given up their hands-on skills and become peoples' managers. One example of one's own job insecurity is the hands-off code "chief architect" who wants to build his own power tower of control to "own" enterprise architecture (see graphic below).



Misunderstanding roles of other people
A manager does not keep up with evolution of roles and does not understand purpose/importance of some new roles that have emerged in a workplace. As a result, a manager tries to "map" new roles to old roles and apply the same yard stick to measure and manage a subordinate. His own lack of understanding could be frustrating to a manager and, therefore, make him feel defensive in discussions of roles and responsibilities of his subordinates.

Compromised self-esteem and desire to protect own status quo
While being a part of a larger organization, a manager might be getting a significant portion of mistreatment in the form of Command & Control behavior from his own superiors. This is where the desire to protect his own status quo and not to look defeated in the eyes of his own peers and subordinates kicks in. There is a growing need for self-redemption and the urge to relieve built-up psychological stress.

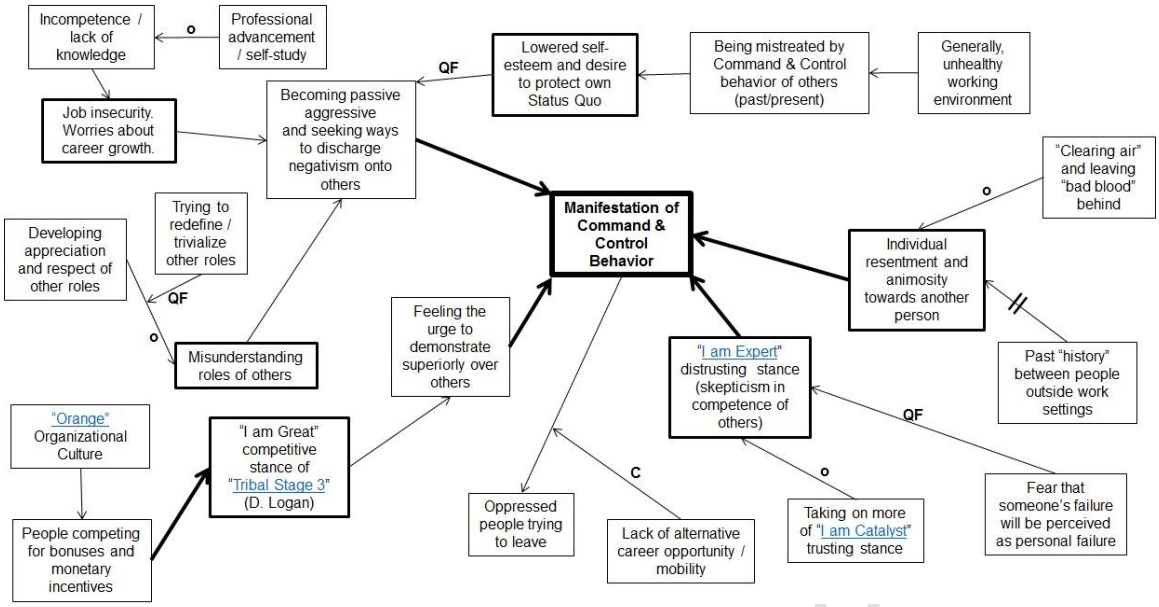
Note*:* All three examples of the root causes of Command & Control behavior above usually result in a manager becoming passive aggressive and seeking ways to discharge negativism onto others. Typically, "others" come in the form of a manager's own subordinates, with the latter becoming defenseless recipients of mistreatment.

"I am Great" competitive stance "Tribal Stage 3" (D. Logan)
With the attitude of "I am great and you are not," a manager perceives himself as someone who is smarter and greater than his subordinates. The person that acquires a managerial position—through the skillful pursuit of a new vacancy (e.g., due to reorganization, force reduction) and/or the experience to navigate organizational political terrain—may feel the need to demonstrate his superiority to others. To continue being perceived as a superstar and to stay in a spotlight of all events that can further multiply his glory, as well as to be able to claim someone else's credit (delivery, innovation, invention) as his own, that type of manager keeps his subordinates at bay to prevent their independent advancement and autonomy.

Individual resentment and animosity towards other people
While not very common and rightfully speculative, there are situations when outside-of-work relationships or individual perception outside of working environment define the relationship between a manager and subordinate. Broken friendship, unsuccessful romantic relationship, differences in personal values, norms or beliefs—all can impact professional relationships at work. A manager, who has an upper hand, may leverage his superiority to repress a subordinate in retaliation to unrelated work matters. On top of being unprofessional, this behavior could be also viewed as "non-sportsmanship-like conduct."

"I am Expert" distrusting stance (distrust in competence of others)
This could be viewed as the least "harm intended" manifestation of Command and control behavior. This is more commonly seen in situations where a manager still has sufficient hands-on expertise (e.g., technical lead) and *can-do* work. Viewing himself as a "super-doer-expert," a manager usually prefers to "shut the door" and resolve all problems on his own, instead of trusting his subordinates to collaborate and come up with shared decisions. A manager-doer prefers to make single-handed decisions while controlling actions and interactions of other people, fearing that someone's failure will be perceived as his personal failure.

Below is a System Modeling diagram-example that illustrates relationships command and control behavior with the reasons described above, as well as some additional system variables that impact system dynamics.



Detailed view & legend can be found here: http://www.keystepstosuccess.com/wp-content/uploads/2017/06/CLD_Command_Control_Behavior.pdf

The "R" Part of HR

Originally published on March 10, 2017 | Location: http://www.keystepstosuccess.com/2017/03/the-r-part-of-hr/

Frequently, HR topics find their way into Agile communities and almost always become heated discussions. Recently, and again, one such topic was raised in the global community of Coaches and Trainers.

Truth be told, HR norms and policies are a direct reflection of organizational culture (also, corollary to structure) and very much define human relationships within an organization. This is why, very often, the "R" part of HR is referred to as "Relationship." A few weeks ago, at 2017 Business Agility, held in NYC, there was a strong reminder about this by Fabiola Eyholzer.

There is a widely shared belief that the historical meaning of "R," originally defined, as "Resource," is no longer appropriate. Or has it ever been? To refer to humans as resources implies that people are inanimate objects, machines, goods or services that are simply acted upon by more intelligent resource managers. But resources do not think, do not take initiatives, do not mature and do not self-advance. And humans do. So, how can humans be just resources? "Resource" was probably an accurate description of a working human in the early part of the last century, during the Industrial Revolution in the era of *Taylorian Management.* (F. Taylor summed up his management efficiency techniques in his 1911 book *The Principles of Scientific Management*.).Back then, when most value of humans' work was in their mundane, unskilled physical factory labor, there was a strong belief that decision making (done by higher-paid skilled management) and decision implementation (done by low-paid, unskilled laborers) must be clearly separated. However, in the twenty-first century of nanotechnology, artificial intelligence, nuclear biology and galactic explorations, should humans still be considered as resources, or rather as humans?

In Agile organizations, where self-organization and self-management is one of the fundamental pillars of success, referring to humans as resources becomes even more misleading. It would be very inappropriate to consider a highly skilled, cross-functional Scrum team member—who is expected to experiment, improvise, inspect and adapt—as a resource. It would be no less misleading to call a Scrum team or a few teams working together on the same complex product as "pool of resources." A manager who says "*I got fifteen resources on this project*" is a Taylorian manager.
And back to the acronym of HR: by re-labeling "R" into Relationships makes the eaning of HR, as an abbreviation, so much stronger.

Indeed, how much more pleasant and comforting (psychologically, of course) would it be for an average worker to know that there is an organizational area (department) that strongly fosters importance of human relationships inside an organization?
Language and wording is powerful: it shapes behaviors.

For more references and publications about HR-related topics, please visit this page.

How to Cultivate T-Shaped Developers

Originally published on November 13, 2016 | Location: http://www.keystepstosuccess.com/2016/11/how-to-cultivate-t-shaped-developers/

Scrum team. It is a cross-functional group of developers that can deliver complete business functionality ("from concept to cash") to customers in a short timeframe. The easiest way to make sure that a team is crossfunctional is to compose it of developers that are initially multiskilled and possess both primary and secondary skills. Such people are usually referred to as **T-shaped**, where "the vertical bar on the **T** represents the depth of related skills and expertise in a single field, and the horizontal bar is the ability to collaborate across disciplines with experts in other areas"(Wikipedia).

There is a common misbelief that T-shaped individuals are hard to find. Here, are some of the most frequently heard concerns that we hear from hiring managers and HR:

- "Most people we come across do not have enough technical diversity."

- "It is hard to find individuals with certain skill set in a particular geographic area. Best folks with a required skill set can be found only in a particular area."
- "We are having difficult time encouraging our team members to learn secondary technologies on the job."

Below are some suggestions for how to acquire or internally cultivate and retain good, multi-skilled, T-shaped developers:

Frame job requirements clearly
Companies must ensure that their job requirements clearly state that they are looking for multiskilled workers. Jobs must be titled accurately, including the mentioning that people are expected to work on Scrum teams, wear multiple hats, contribute to *learning* of other their teammates and actively *learn* themselves. Still, very commonly, we see job descriptions that are titled in favor of conventional single-specialty roles (e.g., BA, Java back-end Developer, QA or Architect). While commonly used buzz words, like "Agile" and "Scrum" are still mentioned in job requirements, proper meaning of these words is not communicated well enough and sometimes is simply misstated. Also, the importance of having multiple skills is reduced by emphasizing titles of conventional, single-specialty roles. Therefore, it is not surprising that people that get attracted to such job requirements rely merely on their original, single skill set and don't have much appetite for extended learning and becoming T-shaped.

Assess candidates properly before hiring and ask for things that really matter
Today, in a typical hiring process, many companies still make too much emphasis on things that matter little. This is mainly due to outdated HR procurement methods and hiring policies, but also, to some extent, because companies rely on inadequately trained recruiting staff (sometimes, external). Therefore job candidates get screened for wrong things and under wrong conditions. Some pre-qualifying, "templated" questions (e.g., "Are you able to work under stress?" "Are you an outstanding performer?" "Can you be a

great team player?" "Can you describe a difficult situation and solution that you came up with?" And so forth.) have little to do with real job requirements and should be treated as common sense.

The biggest downside of using such low value qualifiers in screening is that they cost time and shift focus of both interviewers and candidates away from things that really matter. Also, such phone screening by a single person (e.g., HR or hiring manager) are more prone to bias and misinterpretation. Phone screening could still serve some purpose if it scope is reduced to something basic and much less time consuming. For example, determination of a candidate's legal status and work permission to work or scheduling an in-person interview.

Note*:* Discussing compensation ranges is not advisable by phone since some strongly qualified candidates with slightly out-of-range salary expectations may get disqualified by screening people that don't have the best understanding of a relationship between "service value and cost of service" and do know where compensation range should be tweaked for a right candidate.

It is more advisable to reduce time spent on phone screening in favor in-person interviews with inclusion of practical assessment of individuals' technical knowledge, their cross-functional capability and ability to operate in Scrum team settings. To that end, it makes more sense to involve an entire Scrum team (future team where a candidate will work for) in multiple steps of an interviewing process, including assessment of social, soft and technical skills, while using simulation techniques that mimic a team's daily dynamics.

Pay developers well

- "Good, cross-functional developers are hard to find."
- "There are no experienced Java coders in Chennai."
- "The best architects on the East Coast of the US are in Boston."

We hear these arguments a lot.

Truth be told, good developers can be found almost anywhere; companies just need to figure out better ways of attracting them. This includes, and often comes down to, paying people fairly and competitively.

No single geographical place in the world miraculously grows "the best" technologists of a one kind. Certainly, some trends exist, perhaps, where workers relocate, as they follow large companies that offer jobs. But often statistics that describe these trends are taken out of proportion, data is misinterpreted and numbers are exaggerated. The ability to find best developers of one type, in one geographic area, most likely hints to another more likely theory: companies that claim the above "phenomena" are themselves responsible for creating conditions for such disparity in the first place.

Often, in pursuit of cheaper labor, companies consolidate all their developers of more expensive skill sets (e.g., Java developer is more expensive than HTML developer) in the cheapest geographic location. What companies don't realize is that this decision creates the following challenges for Scrum:

- Firstly, this goes against key principles of team collocation that are critical in Scrum: consolidation of all developers of the same skill set in one geographic location makes it hard to co-locate cross-functional teams. Effectively, with this approach, companies create collocated component teams at the expense of co-located feature (a.k.a. Scrum) teams. But it is the latter that is best for agile product development.
- Secondly, being collocated with people of the same exact skillset makes it hard for individuals to learn new skills from one another (condition of technological "in-breeding").

Ensure that work environment and team dynamics support knowledge sharing between developers

One of the most critical requirements that ensure that individuals are willing to share knowledge with their teammates while working is the existence of a safe collaborative environment, free of internal competition. The willingness to cross-pollinate with skillsets (and become T-shaped) is much higher on teams and in organizations where people perceive each other as mutually supportive peers, not as rivals. This can be achieved by strongly supporting ideas of collective ownership and shared responsibility, by emphasizing the importance of team performance while discouraging individual heroics, knowledge withholding and silos. Organizational cultures where individual performance is overly empathized and individual performance appraisals drive bonuses and monetary insensitive, employees' willingness to share knowledge, teach each other and contribute to mutual T-shaping, while delivering work together, is significantly reduced. Many other undesirable behaviors (e.g., hostility, favoritism, etc.) are frequently seen as well.

This fundamental improvement in working conditions requires strong commitment and support of senior leadership.

Provide internal career paths for hands-on developers to ensure long-term engagement

Today, a typical career path for a successful technologist requires the sacrifice of hands-on work in favor of managing other people. Developers think that by becoming a manager and getting in charge of more junior workers they will increase their own chance to be promoted, move up a hierarchical ladder and collect more pay. This is commonly seen in companies with very complex organizational structures and Command and Control environments, but it's less seen in companies with less hierarchical, flattened structures. This anxious pursuit to become a "manager-in-control" reduces many good developers that can deliver value by performing hands-on work. People are reluctant to remain in the role of a developer for too long because it is perceived as stagnation of professional growth. Also, people feel that it is more difficult to get a decent pay increase by simply remaining in the capacity of a worker bee. Many good developers with already

strong primary skillsets are reluctant to acquire additional technical skills because they view this approach as a low return on investment, compared to pursuing a management role. To them, becoming a manger is a more certain way to grow in ranks and in compensation.

What can companies do to address an individual's reluctance of remaining in a role of a hands-on developer?

Arguably, companies must decouple the process of promotion (gaining seniority, reputation, organizational weight) and compensation increase from acquisition of "power tower" control.

Individual workers must have assurance that by keeping their hands-on technology, deepening their primary technical skills and broadening secondary skills, they will not be missing out on career advancement and ability to make a better living. People must also be assured that by becoming higher compensated, over time, while remaining in a capacity of hands-on doers, they will not become an easy target for downsize/force reduction in favor of a younger and cheaper workforce that will come from colleges and universities. Here, a strong bet is being made on the assumption that senior hands-on workers with longer industry experience will have much more technical expertise that is coupled to business domain knowledge—something that shall make their higher pay well justified by employers.

Again, this organizational change is dependent on decisions that come from senior echelons of organizational structures.

Bad Smells: Appraisals and Performance Reviews Influenced by Agile Coaches

Originally posted on: 26 June 2014 | Location: https://www.scrumalliance.org/community/articles/2014/june/bad-smells-appraisals-and-performance-reviews-infl

Agile/Scrum coaches should not position themselves in ways that will give them an authoritative role within the organization where they coach. Otherwise, organizations/people who get coached will not gain long-lasting learning. They will view coaches as "Command and Control" figures, and this will lower their chances of developing their own self-sustainable Agile practices and Kaizen culture.

Organizations may expect, at most, short-term improvements that are based on directives and commands of such coaches. In his book *The Culture Game*, Daniel Mezick very well describes the do's and don'ts of an Agile coach (Chapter 17). This philosophy neatly applies to Agile coaches who operate as consultants. How about coaches who are no longer consultants?

The situation becomes even more challenging for Agile coaches who join organizations as employees after operating for a while as consultants elsewhere. Here, coaches may get drafted into activities that would conflict with the basic rules of engaging as a coach. When such situations arise, a good, mature coach who is familiar with the do's and don'ts of his profession and who has been through various stages of coaching (teaching, coaching, advising) should try his best to maintain his coaching integrity and professionalism in his actions—resisting such drafting.

Here is an example of such a challenge: being requested to provide *performance appraisal feedback* to individuals who are being coached. (This is not to be confused with constructive one-on-one feedback that coaches are expected to give to their coachees as part of the coaching/mentoring/counseling process). Here the reference is made to a formal process that many organizations have in place for evaluating their employees in ways that may impact those employees' compensation and career development.

Drafting a coach into such position will create a serious conflict of interest for that coach and will ruin his ability to influence the natural growth and evolution of learning among the people who are coached; this is damaging to a coach-coachee relationship.

Impartiality and neutrality of a coach is highly important. Only by remaining neutral and non-authoritative will a coach be able to help the organization and its employees to self-discover, self-improve, and to become autonomous in their journey to success. Even if a coach becomes a part of an organization, he should strive to preserve some key specifications of the coaching profession.

Motivation 3.0 Is Required to Transition from Tribe Stage 3 to Tribe Stage 4

Originally posted on: January 21, 2016 | Location: https://www.scrumalliance.org/community/articles/2014/july/motivation-3-0-is-required-to-transition-from-trib

In his book *Drive*, Daniel Pink says that when it comes to motivation, there's a gap between what science knows and what business does. Our current business operating system is built around external, carrot-and-stick motivators—which don't work and often do more harm than good. We need a system upgrade. And the science shows the way. This new approach has three essential elements:

1. ***Autonomy:*** The desire to direct our own lives.
2. ***Mastery:*** The urge to make progress and get better at something that matters.
3. ***Purpose:*** The yearning to do what we do in the service of something larger than ourselves.

According to Pink, humans evolve through three distinct stages of the Human Operating System:

1. ***Motivation 1.0*** presumes that humans are biological creatures, struggling to meet basic needs for food, security, and sex. This system is mainly good for survival struggles.
2. ***Motivation 2.0*** rests on Theory X of Human Motivation, when management assumes that employees are lazy, tend to avoid work, and best respond to rewards and punishments in their environment.
3. ***Motivation 3.0*** rests on Theory Y of Human Motivation, when management begins to understand that when it comes to intellectual work, employees are ambitious and self-motivated and will exercise self-control. This system presumes that humans seek purpose maximization no less than profit maximization.

In *Tribal Leadership* by David Logan, a tribe is a group of between 20 to 150 people (150 from Robin Dunbar's research, popularized in Malcolm Gladwell's *The Tipping Point*). A tribe is a basic building block of any large human effort, including earning a living. Tribes and their leaders create each other. People who belong to a tribe tend to recognize fellow tribe members easily. A small company is a tribe. A large company is a tribe of tribes. At a large company, several cultural stages may operate at the same time. Tribal leadership focuses on language and behavior within a culture.

According to Logan, tribes go through five distinct stages in of evolution:

1. "Life Sucks." (In general.)
2. "My Life Sucks." (In the background: The lives of others are OK.)
3. "I'm great." (In the background: "You are not.")
4. "We're great." (In the background: "They are not.")

5. “Life is great.” (There is no “they.”)

And here comes the main Agile argument: *It is unrealistic to transition from Stage 3 to Stage 4 in tribal evolution unless people are driven by Motivation 3.0.*

Enterprise-wide Agile transformations, adoption of Kaizen culture, flattening organizational structures, building out cross-functional product development teams of skilled T-shaped individuals who think in terms of "We are the team, we are great," not in terms of "I am the great and what else can I do to become even greater" -- all this is the hallmark of a Stage 3-to-Stage 4 transition. This is what we want in Agile. And this is only possible if people are properly driven -- by Motivation 3.0.

Today, most companies still use Motivation 2.0 deeply seated in their cultures: monetary incentives (salary increases, bonuses), individual (versus team) performance assessments and appraisals that encourage people to think “I,” not “We.” This naturally makes people selfish and self-centered: An individual's overall happiness and success is much more tied to (is the function of) what that individual is/does then to what his/her tribe (team) is/does. Therefore, individuals are less eager to work together for a greater common good then they are to work for their own good. At the end of every day, each person still thinks about what he/she can claim as “my own” to achieve individual goals.

To move on to Tribe Stage 4, which is effectively the ideal environment for building Kaizen culture, adopting an Agile mindset, and forming and norming highly performing Scrum/feature teams, people need to be properly motivated by their organizations. Companies must crank their motivation engines to version 3.0.

Reviews, Appraisals and Incentives

Originally posted on: January 21, 2016 | Location: http://www.keystepstosuccess.com/2016/01/reviews-appraisals-and-incentives/

I wanted to start this discussion with the Wikipedia definition of Performance Appraisal: "*A performance appraisal (PA), also referred to as a performance review, performance evaluation,[1] (career) development discussion,[2] or employee appraisal[3] is a method by which the job performance of an employee is documented and evaluated. Performance appraisals are a part of career development and consist of regular reviews of employee performance within organizations."*

While all three terms are being included in the same definition ("*review*," "*evaluation*," "*appraisal*"), in practice, companies predominantly use the term "*review*" to describe PA as it implies less scrutiny and preconception towards an employee. But does this change the essence of the process if a less abrasive term is being used?

Usually, the PA process starts with setting individual career goals by an employee. Of course, this is not done in a vacuum of what an employee only chooses for herself. Individual career goals should be in-line with organizational/department career goals, usually set by management. It is expected that throughout a year an employee must *steer* herself towards set goals, while performing her day-to-day job responsibilities.

Every company that supports a PA process has a scoring system (variations exist) to rank employees against other employees, based on scores that an employee earns while performing her yearly accomplishments (goals set vs. goals achieved). Some organizations offer a midyear (quarterly, at best) checkpoint to employees, at which time an employee reviews with her manager how she is performing with respect to the goals set originally. Practically, no companies handle PA as an actively managed, iterative Agile process.

For most companies the whole PA process typically serves the following three main purposes:
1. To identify low-performing employees that are potentially a subject to downsizing.
2. To identify high-performing employees that are potentially a subject to promotions.
3. To decide how discretionary incentives (bonuses) can be distributed among employees.

While on the surface PAs still appear as an effective way ensure quality of employees and provide benefits to organizations, under the surface this process presents real challenges. These challenges become more visible at organizations that are in the process of adopting Agile culture, since in Agile environments systemic organizational dysfunctions get exposed much better.

But before we dive deeper in the discussion, let us first briefly refer to some credible research and studies that exists today:

In his book *Out of the Crisis*, originally published in 1982, Edward Deming discusses Seven Deadly Diseases of Management and refers to individual performance reviews and performance evaluation as *Disease # 3*. Deming's philosophy of transformational management is about the seriousness of barriers that management faces today, while improving effectiveness and striving for continual improvement. Deming argues that trying to evaluate and measure workers with the same yardstick causes more harm than good to individuals and companies.

Tom Coens and Mary Jenkins offer specific suggestions on how to replace performance appraisals with a more effective system that emphasizes teamwork and empowerment in their book *Abolishing Performance Appraisals: Why They Backfire and What to Do Instead.* Coen and Jenkins discuss new alternatives that produce better results for both managers and employees.

In his Forbes article, "Eliminating Performance Appraisals," Edward E. Lawler III, a distinguished professor of Business at the University of Southern California, advocates that organizations stop doing performance appraisals. Professor Lawler states that performance appraisals frequently do more damage than good, with damage levels fluctuating between wasted time (least troublesome) to reasons for alienation of employees and creating conflicts between them and their supervisors (most troublesome).

Garold Markle, an author, executive consultant and speaker, leverages his studies and experience with systems theory to illustrate his points with real-life examples of why employees and managers both have come to believe the "ubiquitous performance evaluation as industry's poorest performing, most ineffective, and least efficient personnel practice". In his book "Catalytic Coaching: The End of the Performance Review." Markle provides an innovative way to measure ineffectiveness and inefficiency of performance evaluations and then introduces his catalytic coaching to replace them. His statement is awakening: "People hate performance reviews."

In his book *Drive*, Daniel Pink offers a paradigm-shattering view on what truly motivates people in their lives. Pink draws on four decades of scientific research on human motivation to expose a mismatch "between what science knows and what business does." Pink challenges the mistaken belief of many that people doing intellectual work will demonstrate higher performance when incentivized monetarily. As to Pink's research, it becomes clear that individual performance evaluations and individual appraisals that are linked to monetary rewards are not an effective way to steer individuals to become more efficient and productive. Therefore, they should be abolished. And the list goes on….
So, now let's take a closer look at the problem, with some specific examples:

Fabricating goals to game the system
Do goals that employees officially set for themselves (in a system of record) truly reflect their genuine, personal goals?

It is not uncommon that real personal goals are risky and challenging to achieve or may take longer than initially expected. Many other goals can be also situational or

opportunistic: they may change as a situation changes or unforeseen opportunity presents itself (job market trends, other job opportunities, personal life). People want to have freedom and flexibility to adjust their goals to optimize their personal benefits; this is human nature. There is no true personal benefit for an individual to set in stone her goals into a personal development plan at year-start and then have to deal with unpleasant consequences at year-end, *if* she does not meet her goals. In general, to set her real goals, a person needs to know that it is safe to actively manage them along the way and, if needed, safely change and/or fail them, without fearing negative consequences.

But is there any safety with PA processes if job security, career advancement ability and ability to collect fair compensation are at risk? If there is no personal safety, the exercise of setting personal goals becomes nothing but a routine of faking objectives that are "*definitively achievable.*" People are forced to do it, to minimize the risk of being scrutinized by their management for not meeting their goals. Setting individual goals becomes just a formality that brings no true value to an employee.

The process of individual performance reviews becomes even less meaningful if people work in small teams where swarming and collective ownership is important and joint delivery is expected. In cases such as these, people are forced into unhealthy competition with each other over goals, trying to privatize what should be owned and claimed collectively.

Another challenge with evaluating employees' individual career goals is that in pursuit of personal goals, people frequently drop the ball and de-prioritize common goals. Again, this dysfunction becomes much more vivid in going-Agile environments, where Agile frameworks (e.g., Scrum, Kanban) de-emphasize individual ownership and reinforce the importance of collective ownership. Often, close to midyear and end-year performance reviews, collaboration and mutual support of team members worsens as silos get created and everyone begins to think about their own goals at the expense of shared goals. This translates into inefficiency and productivity drop: swarming, velocity and throughput go down; cycle time and queues grow; handovers take longer.

Example:
Jane is an employee of a large insurance company. She is being requested to enter in a company's system of record her personal goals—things that she intends to achieve throughout a year. Jane is smart and to avoid any unwarranted risk, where her personal success depends on success of others, she creates goals that are free of dependencies. Jane creates a set of personal goals that other group members do not know and don't care about. Her line manager John, also discourages Jane from sharing such information.

However, Jane does not work alone. Her day-to-day work is tightly coupled with work of other individuals in her group: Jim, Jeff, Jill, Joe and Julie.

Jane really values teamwork. She also feels that by closely working with her group members, by swarming and sharing day-to-day activities she can earn a lot more than if

she worked by herself. This is where Jane decides to put her full focus: on teamwork. She does not feel that creating an additional set of personal goals can add real value to her professional growth. But Jane needs to "feed the beast"; she needs to provide her line manager with a list of "achievable" bullets that the latter can measure. At the same time, Jane does not want to create a conflicting situation with her colleagues by diluting her focus on shared goals by shifting it to personal goals. Therefore, she fabricates her personal goals: "quick kills" and "low hanging fruits"—something that she can easily claim as her "achievement" without jeopardizing common interests of her team. Jane is forced to "game" the system to minimize harm to herself and her team.

In his book *Tribal Leadership*, David Logan describes five tribal stages of societal evolution. According to his research, corporate cultures typically oscillate between **Stage 3** ("*I am great and you are not*") and **Stage 4** ("*We are great and they are not*"), with Agile organizations trending more towards Stage 4. When individuals are motivated (a.k.a. "manipulated") to think more about individual performance than about collective performance, they mentally descend to Tribal Stage 3 and, as a result, drag along their organization to this lower stage. It is very important for organizations and their senior leaders to understand that motivation is one of the most important factors that drive evolution of corporate culture.

Note: To understand how Motivation Evolution (defined by Daniel Pink in *Drive*) relates to Tribal Evolution (defined by David Logan in *Tribal Leadership*), please refer to this tool.

Unhealthy competition, rivalry and jealousy

Let's face it, overemphasizing individual performance evaluations and allowing them affect job security, promotions and compensation of individuals does not come free to organizations. It comes at the high expense of lowered collaboration, unwillingness to share knowledge and provide peer-to-peer support, increased selfishness and self-centric behaviors. For individuals that are encouraged to work and produce collectively (e.g., Scrum or Kanban teams) uneven performance evaluations frequently result in jealousy and feelings of unfair treatment. These dysfunctions become more frequent around times when employees are due for midyear and end-year reviews. PAs have adverse effects on individuals' abilities to focus on work and, as a result, produce high quality products and think of customer satisfaction.

It is worth mentioning, ironically, when dysfunctions are uncovered, it is Agile that becomes the target for blaming.

But Agile is hardly at fault here as it only provides transparency and reflection of already existing dysfunction.

Example:
Jane works alongside Jim, Jeff, Jill, Joe and Julie. All of them are smart, self-motivated and talented technical experts that cumulatively have more than 70 years of software

development experience. Their work is intense: there are lots of deliverables and their timeframes are rigid. The group has served the same client for many years and, so far, the client is happy. The work that this team performs requires a lot of collaboration, collective thinking and brainstorming, teaching/learning from each other and, of course, collective delivery.

But then comes a midyear review period and Jill notices that Jeff is not as supportive of her as he was at the beginning of the year. Jeff becomes less responsive to Jill's requests. He does not share his knowledge as readily as he used to; he does not give advice. Tasks that used to be handled collectively by Jill and Jeff are now illogically split by Jeff as he tries to focus only on what he assigns to himself.

There is also a noticeable change in Julie's behavior. Julie becomes very eager to be the one who stands in front of a client and presents deliverables of the whole team. This responsibility used to be rotated from one person to another, with no one caring too much about being a "spokesman." But as mid-review came, Julie clearly stepped up to be the main, customer-facing presenter. Julie also tries to make it obvious to John (the group's manager) that it is her, Julie, who presents to a customer. Julie wants to be viewed as a "centerpiece" and gain most of spotlight.

Jim's contribution to the group's efforts have also subsided. Early in the year, Jim used to be a very active participant at the team's brainstorming meetings and workshops. As midyear arrived, Jim started spending a significant portion of his time working on items that are not related to the team's shared work; his focus has noticeably shifted to personal work that he does not even discuss with others.

Since the beginning of the year, it has been customary for the group to go out for drinks to a local bar every Friday. But this tradition is now barely followe, at the midyear period. There seems to be less desire for the group to socialize outside work settings. Everyone finds an excuse not to make it. The group's synergy has gone down noticeably. What used to be a well-jelled team of great collective performers has turned into a group of self-centered individual achievers that want to be acknowledged for their heroics.

"Scripted" ranking to force-fit into bell-shaped curve

Typically, when an organization ranks its employees based on individual performance, a bell-shaped curve is produced where *samples* (ranked employees) are binomially distributed around the median: the majority of samples are centered ("center mass"), representing average performing employees; left tail, representing low performing; and right tail, representing high performing (overachievers). Statistically, a bell-shaped curve is a normal distribution of any large sample. The symmetrical shape of a curve ("bell"), however, can be influenced by additional three main factors (forces):

- *Platykurtic distribution*. It lowers the number of samples around the median (average performers) and increases number of outliers (underperformers and overachievers), equally on both sides. A curve remains symmetrical.

- *Leptokurtic distribution*. It increases the number of samples around the median (average performers) and lowers number of outliers (underperformers and overachievers). A curve remains symmetrical.
  Uneven distribution of samples on left and right sides from median. Typically, this increases the number of samples on the left (underperformers) or right (overachievers) tails of a curve, while also disturbing the symmetry of a curve and evenness of sample distribution around the median (average performers). A curve loses its symmetry.

This statistical distribution is tightly coupled to actions that management takes towards its employees at year-end. However, the shape of the bell curve does not "drive" (as it might be expected) managerial year-end decisions; it is rather *driven by* them.

Managerial decisions are driven by financial conditions of an organization as well as other strategic organizational plans. When managers review their employees, they must account for such factors to make sure that a bell-shaped curve does not exceed organizational capabilities of promoting too many employees and giving out too much money. Effectively, the entire process of performance assessments becomes a retrofitting exercise that shapes a bell curve, basing it on organizational capabilities. This makes the process, practically, staged or "scripted." What further adds to the irony of this situation is that at times an employee may report into a manager that does not even have sufficient skills to perform an objective assessment of an employee's performance. For example, an architect or a software engineer that reports into a nontechnical manager (e.g., PMO) has a much lower chance to objectively discuss her work accomplishments and receive an objective feedback during the PA.

There is a need for an alternative approach that will help dealing with overly complex, overstaffed organizations that spend so much time and energy trying price-tag its employees.

Here is an idea: how about more thorough checks of background and references, more rigorous interviewing processes that involve practical (hands-on) skills assessments, try and buy periods, or some other, more objective methods before hiring an individual full time?

Instead of attracting cohorts of workers of questionable quality and then dealing with inevitable force reduction or worrying (or pretending to worry!) about employees maintaining and/or improving their quality, hiring managers should be striving to acquire and retain lower *quantities* of higher *quality* workers: self-motivated, enthusiastic professionals with a proven track record and clearly defined career goals…AND be willing to pay them higher compensations. This may require offering more competitive base salaries and abolishing manipulative discretionary incentives: removing money from the table makes intellectual workers think more about work and less about getting paid. This approach would also ensure that the quantity of employees is kept at a minimum (and also ensures lowering overhead, complexity reduction, organizational descaling), while maximizing quality. Such an alternative should render performance reviews much

less important or even obsolete as there will be no need to reduce employees at year-end or thin-slice discretionary incentives among too many candidates.

Example:
John is a line manager for the development group. John has great organizational skills. He is well spoken and can greatly articulate his thoughts. But John, has never developed software products; he is not technical. John knows that all of his team members are "good guys": knowledgeable, enthusiastic, and mutually supportive. But when the team works together, John really cannot validate quality of work that they produce. (Luckily, there is one reliable measurement of the team's success—it is customer satisfaction). The only thing that John can validate is the team's vibe and spirit. But even when John notices disagreements or a temporary misalignment among the team members, it is impossible for him to offer constructive advice or understand a root cause. What is even more challenging and frustrating for John is that due to the nature of the team's work (closely collaborative, collectively shared), he cannot objectively assess individual performance of every team member. In conversations with John, the team members rarely use the word "I"; it is typically "we."

John is in a tough position. How can he decide who the best performer on his team is and who is not? John needs to be able to 'rank' his people and based on ranking, decide who gets promoted and paid more at the end of the year. Deep at heart, John feels that everyone deserves a promotion and monetary "thanks" but he cannot satisfy everyone. John's management informs him that only one person from the team can get promoted, and the amount of incentive money allocated to his group is limited; in fact, it is less than last year.

Around midyear time, John begins evaluating how each of his team members has performed up-to-date. John does this based on "achievable" goals that were set by each employee at year-start. John's inability to truly understand the nature of peoples' technical work adds to his challenge…and frustration. He cannot objectively evaluate his employees, let alone rank them against each other.

Meanwhile, John's management expects from him a ranking model that will fit into a bigger picture of organizational capabilities to give promotions and bonuses for a given year. It means that even if John feels that all of his members are outstanding performers, he will not be able recognize this officially. At most, he will be able to recognize that they have achieved their set goals. Further, based on what John learns from his management, he has to commit an even less unpleasant act. Learning that a certain percentage of a company's workforce has to be reduced, John has to identify and set up for possible future downsizing some people from his group. It is clear to John that people that are outstanding performers are not the best candidates for downsizing (potential HR cases). Therefore, John decides to force-fit some of his team members into a bell-shaped curve, away from the right-sided tail, towards the middle (average performers) and left-sized tail (underperformers). John uses the organizational "script" to play his own team. What John does is a wasteful act that is full of subjectivity and ambiguity. The process is also

destructive to the team's cohesiveness and morale. John is at risk of losing some good people sooner than he could expect.

Generating waste

Rarely do companies consciously analyze how much time and effort is being spent on the performance evaluation process by employees, line management, senior management and HR. Unfortunately, for large enterprise-size companies, these expenditures are already budgeted. From the standpoint of lean thinking, today's typical process of PA conducted by line managers is a clear example of organizational overhead that slows corporate cultural evolution and prevents companies from maturing to Tribal Stage 4.

Example:

All members of the team, Jane, Jim, Jeff, Jill, Joe and Julie, spend a lot of time during the year writing and reviewing their personal coals. John spends a lot of time reviewing and discussing personal goals of every team member. John also spends a significant portion of his time with his line management, discussing achievements and intended ranking for each of his subordinates. Overall, the amount of time this entire group of people spends on the PA process creates a lot of unnecessary procedural overhead and over processing.

Alternative approaches to performance reviews

Are there any working solutions to this problem? Is it possible to ensure that organizational behavior towards employees (e.g., motivating and incentivizing) is more in line with what is best for organizational prosperity, business satisfaction, waste reduction and the creation of a more efficient organizational structure and Kaizen culture? Is there a way to depart from archaic norms and behaviors gradually, without causing too much stress to an organizational ecosystem, perhaps, by offering alternative, less harmful, interim solutions?

First, let's be clear: a natural knee-jerk reaction of any individual when she is told by someone why she is not "perfect" and what she needs to do to improve is defensive. Of all reasons, the biggest reason is her understanding that based on the evaluation someone will decide how much she will get paid. Although an individual may keep her feelings and emotions concealed under political correctness and diplomacy, there is harm being done.

***The end goal of any organization should be abolishing performance appraisals completely and substituting them with other, more effective methods of motivating individuals*.**

But for now, let's look at some interim alternatives that can help us depart from individual performance appraisals gradually by moving to less harmful approaches. Here are some potential, interim (short term solutions) alternatives of how co-dependency between individual PA process and incentives allocation process can be dissolved (graphics are here):

- Instead of prizing individuals an organization can prize teams—do this based on what an entire team produces, not a single individual. If individuals must work in tight collaboration and are expected to cross-pollinate with expertise and produce together, what is the point to stress individual contribution and individual performance? Let a team, internally, decide who is pulling them up and who is dragging them down. Individual underperformers will be quickly identified in such settings, and a team will try to either expel them or help them to improve. Also, please note that prizing a team (monetarily, team bonus) does not have to be coupled to "performance assessment." This could be done, simply, as a profit sharing model between business and technology: if work of technology has noticeably improved the bottom line of its business partners, the latter should be happy to prize hard and successful work of the latter.
- Take away singleton decision-making capability of defining what a team deserves in terms a "monetary prize" out of line managers' hands and "spread the wealth" across multiple parties: make it based on customers/stakeholders satisfaction, senior management satisfaction, third party feedback, etc. But again, judge teams, not individuals (important!).
- Make monetary incentives allocation more objective and formula-driven than subjective and single opinion-based. Here are a few "formulas" to achieve this:
  1. Monetary Incentives Are Equally Allocated among all employees whose work is tightly coupled and collective ownership is expected
  2. Monetary Incentives Are Allocated in Proportion to Base Salary of an employee: decide on employee's costs when she is hired (based on expertise, experience, etc.) and then fall back on formula "a."
  3. Monetary Incentives Are Allocated based on Team's Internal Voting, done confidentially (incremental, 360-degree reviews by all team members).

Note: Consider the above options as short-term, interim solutions on the way to completely abolishing conditional monetary incentives. Although, team-level incentives are less dangerous than individual incentives, they still carry harm in them: they make people think of getting paid, not about doing work. Ideally, for any kind of intellectual work, money should be removed from the table.

Conclusion

The famous quote from the book Out of Crisis, written by Edward Deming (originally published in 1982), summarizes this topic well:

"The idea of a merit rating is alluring. The sound of the words captivates the imagination: pay for what you get; get what you pay for; motivate people to do their best, for their own good. The effect is exactly the opposite of what the words promise."

References

- Deming, W. E. 1993. *The New Economics for Industry, Government & Education*. Cambridge: Massachusetts Institute of Technology Center for Advanced Engineering Study.

- David Logan; John Paul King; Halee Fischer-Wright. *Tribal leadership: leveraging natural groups to build a thriving organization*. New York : Collins, ©2008
- Tom Coens and Mary Jenkins. 2012. *Abolishing Performance Appraisals: Why They Backfire and What to Do Instead.*
- H. Pink. 2011. *Drive: The Surprising Truth About What Motivates Us.* Riverhead Books.
- Garold Markle. 2000. *Catalytic Coaching: The End of the Performance Review*. Quorum Books.
- Edward E. Lawler III. 2014. *Eliminating Performance Appraisals*, http://www.forbes.com/sites.
- Jeffrey Pfeffer, Robert I. SuttonHard, 2006. *Facts, Dangerous Half-Truths And Total Nonsense: Profiting From Evidence-Based Management.*
- Tom Coens, Mary Jenkins, Peter Block., 2002. *Abolishing Performance Appraisals: Why They Backfire and What to Do Instead.*
- Alfie Kohn, 1993, *Punished by Rewards.*
- Samuel A. Culbert, 2010, *Get Rid of the Performance Review.*
- Adobe Systems set to scrap annual appraisals, to rely on regular feedback to reward staff.
- Microsoft's Downfall: Inside the Executive E-mails and Cannibalistic Culture That Felled a Tech Giant.
- Get Rid of the Performance Review!

Agile @ Scale

HR-Related LeSS Experiments—Deciphered

Originally published on March 22, 2019 | Location: http://www.keystepstosuccess.com/2019/03/hr-related-less-experiments-deciphered/

Large Scale Scrum has a history of more than a decade. The first book about LeSS was published by C. Larman and B. Vodde (the co-creators of LeSS) in 2008. There were two more books on LeSS, subsequently written in 2010 and 2016. There is no surprise why the collection of LeSS experiments from the field is so valuable: the authors have documented many (more than 600) experiments, based on their personal experience with LeSS adoptions, as well as feedback and information collected from other organizational design consultants, coaches and early adopters of LeSS around the globe.

Today, references to **LeSS Guides and Experiments** can be found in various places on the internet and intranet of many companies that have decided to experiment with LeSS.

This writing is about a small subset of LeSS experiments that are specifically related to **HR norms, policies and practices**. They are all listed in the guide (referenced above), under the section "Organization" and it implies that they are directly related to organizational design—the first-order factor that is responsible for success of LeSS adoptions and agile transformations, at large.

Experiments with performance appraisals

"*Avoid… Performance appraisals – p. 273*" —There is a lot of research and evidence supporting that individual performance evaluations and individual appraisals that are linked to monetary rewards are not an effective way to make individuals become more efficient and productive. When a manager appraises an employee, usually only one opinion in the room matters: a manager's. Feedback that is delivered once or twice a year is not timely and therefore is hardly actionable by an employee, thus useless for the most part. Neither an individual that delivers an appraisal, nor an individual that receives it, likes the process. The process is also pretty *expensive* as it uses a lot of the company's resources: It involves lots of documentation, coordination and man-hours spent by many people, from first-line management to HR.

It is worth noting that there is an indirect relationship between the conventional budgeting process and conventional performance management process—both of which harmfully feeding off of one another. This is described in the book *Implementing Beyond Budgeting: Unlocking the Performance Potential*, by Bjarte Bogsnes. In his work, Bjarte refers to performance appraisals as a "*legal trail for a rainy day*."

"*Avoid… ScrumMasters do performance appraisals – p. 275"* —Just like performance appraisals done by Agile coaches could lead to serious dysfunctions (page 130), performance appraisals done by ScrumMasters are extremely harmful. Drafting ScrumMaster into this role will create a serious conflict of interest and will hinder ScrumMaster's ability to influence natural growth and evolution of learning among team members. Impartiality and neutrality of ScrumMaster is highly important; becoming an appraiser takes away this advantage. Only by remaining neutral and non-authoritative (performance appraisal is exhibition of authority) will ScrumMaster be able to help a team to self-discover, self-improve, and become autonomous in their journey to success.

"Try… De-emphasize incentives – p270." | "Avoid… Putting incentives on productivity measures – p. 271."—If achieving a higher productivity (output, velocity) is coupled with monetary incentives/perks or other political gains (typical of many companies that overuse scorecards, metrics, KPIs, RAGs), there will be always attempts by individuals/teams to claim successes/achievements by

"playing the system," in pursuit of rec**ognitio**n and a prize. For example, in pursuit of higher productivity, teams may start inflating estimates to claim higher velocity or deliver work that is low in priority but simple to deliver—just to create an illusion of volume. Incentivizing "higher velocity" is an invitation to moving from "low Fibonacci numbers to high Fibonacci numbers" during estimation. (Also, see Addressing Problems, Caused by AMMS)

"*Try… Team incentives instead of individual incentives – p. 272*"—The process of individual performance reviews loses its original meaning when people work on the same teams, where swarming (working together on the same task) and collective ownership is encouraged. Offering individual incentives to people would just polarize them and move them in opposite directions, towards becoming selfish, individual performers and superheroes. In cases such as these, people may be easily drafted into unhealthy competition with each other over claims of success, trying to privatize what should be owned and worked on collectively. Companies that continue incentivizing individual performance with monetary perks just continue widening the gap between "*what science knows and business does*" (quote from Daniel Pink).

"Try… Team-based targets without rewards – p. 273"—Clearly, team-level behavior is an extension of individual behavior. Just like individuals could be inclined to game the system, so could whole teams, under certain conditions. Just like individuals, whole teams could be drafted in unethical conspiracies to game numbers in pursuit of meeting targets or beating other teams (e.g., producing "higher velocity") whenever monetary rewards are at stake. It is absolutely necessary to set targets for individual teams that work on par with one another. for the same organization. It would be best to decouple team targets from team rewards. The latter could be handled through some sort of *profit sharing formula*, based on a company's financial success that is traceable back each team's work.

Experiments with Job Titles

"Avoid… Job titles – p. 276 | Try… Create only one job title. Try… Let people make their own titles – p. 277 | Encourage funny titles" – p. 277—In pursuit of job titles, individuals may also seek gaining authority and an "upper hand" over their peers and colleagues. This may lead to artificial organizational complexity and hierarchy, as well as a casting system. Individual job titles can also polarize people and drive them in opposite directions, away from shared ownership. It is for this reason that on Agile teams (e.g., Scrum) there is only one title: Developer. This approach encourages people to think of each other as on-par, as peers, and grow into T-shaped, multi-skilled, cross-functional, willing-to-swarm workers. In situations where some distinction between individual jobs is absolutely necessary, funny job titles are recommended. For example, instead of calling someone QA Tester, a person could be called "Bug Finder and Exterminator."

"Try… (if all else fails) Generic title with levels – p. 277"—If it is absolutely necessary to have title distinction (e.g., to signify different levels of seniority/expertise of individuals), try using a leveling system. For example, Developer level 1(junior), Developer level 2 (mid-level), Developer level 3 (senior). However, care should be exercised not to explicitly associate different title levels with different levels of pay.

Experiments with Jobs

"Try… Simple general job descriptions – p. 278"—Do not overcomplicate job descriptions. Precision in a description may lead to contractual perception of what a person should and should not do in a workplace. This may also limit a person's willingness to step out of his comfort zone and learn other areas of work, other skills and becoming multi-faceted. It may then further lead to "managing by objectives" that are based on detailed job descriptions and subsequently bring about problems of

performance appraisals, described above. Complex job descriptions also have a tendency attracting underqualified external candidates whose resumes are excessively long, as they are tailored to closely match complex job descriptions. (Relevantly, attracting bad Agile coaches by creating inappropriate job descriptions is a known problem.)

"Try… Job rotation – p. 279 | Try… Start people with job rotation – p. 280"—Give individuals opportunities to learn new domains, technologies, lines of business. This is will reduce the risk of a person becoming uninterested/bored with his current job. Further, by rotating from one job to another, a person may discover where he fits best and delivers the most value. By having this opportunity, a person will also have a higher chance of merging the gap between "having to do a job" and "wanting to do a job." This is especially important with newly hired people that have a limited industry experience (e.g., recent college graduates).

Experiments with Hiring

"***Try… Hire the best – p. 280 | Avoid… Hiring when you cannot find the best – p. 281***"—Do not settle for less than the "*best people your money can buy."* It is better to rely on fewer great people that you already have on staff than bring on more underqualified people to speed up work, especially at the end of a project that is already late (Brook's Law). From a system thinking perspective, if you are trying to increase velocity (output) by a Scrum team and decide to do so by adding more developers that you procured on a low budget (low pay will most likely buy you low-skilled developers), you will most likely reduce velocity by having low-skilled developers introducing more bugs into a system. Please, see why.

"***Try… Team does the hiring – p. 281***"—If you plan on hiring an individual to join a team, please make sure that a team does most of the interviewing and vetting. Through that, not only a person's skills and experience will be examined, but it will become more apparent if a person can organically jell with a team—if there is compatibility, chemistry and synergy with other team members. Panel interviews by whole teams are usually much more effective since they include practical tests, real-life simulations and hands-on exercises. It also allows some people to observe while others ask questions, and then team members should rotate roles. Try to reduce the level of influence that HR personnel and first-line management have on the process as much as legally possible. This will reduce the amount of subjective, administrative, frequently bias and error-prone screening (refer to top of page 17).

Conclusion:
As a summary, please consider the following quote that describes sushi-roll-like organizational design in Large Scale Scrum (LeSS), by C. Larman (also, explained in detail in Agile Organization, as a Sushi Roll article):

"It is vital to appreciate that organizational agility ***cannot be achieved by a development team*** *in isolation—is a system* ***challenge for organizational re-design****. Especially, when you are interested in LeSS within an R&D department of thousands, where each product group may have 200 or 700 people distributed in two or five sites around the world. If an engineering team has the technical capacity to adapt or change quickly,* ***but requirements management, legal practices, product management, HR policies, site strategies and deployment*** *processes all emphasize rigidity, conformance to original plans, conformance to the status quo, and slow practices, then how can there be real agility?" - Craig. Larman*

Original source: www.scrumalliance.org/community/spotlight/craig-larman/june-2015/less-agile-or-less-agile

In it, HR policies are listed as one of the vital elements of overall organizational agility.

Why Is LeSS Authentic? Why Should Leadership NOT Exempt Itself from Learning LeSS?

Originally published on June 18, 2019 | Location: http://www.keystepstosuccess.com/2019/06/why-is-less-authentic-why-should-leadership-not-exempt-itself-from-learning-less/

Large Scale Scrum (LeSS) is the Agile framework that has a history of implementations, trials and errors, experiments and experience reports collected and documented throughout a decade.

LeSS is Scrum performed by multiple teams (two to eight) that work on the same widely defined product for the same Product Owner.

LeSS stresses the importance of organizational descaling (a.k.a. flattening) that needs to happen before Agility can be scaled. The first LeSS book (out of three published so far) was written in 2008 and it had incorporated the ideas of its two authors, C. Larman and B. Vodde by mainly including their own experiences of initial LeSS adoptions, from years before.

Overall, the LeSS journey had begun many years before Large Scale Scrum had been officially presented to the world and recognized as a framework, and this is important to acknowledge. **But why?**

Because LeSS, unlike some other very popular and commercially successful frameworks that are very easy to "unwrap and install" was *not* invented *re-actively*, as a "quick fix/hot patch" in response to growing market trends and business needs (commercial driver). LeSS is authentic. LeSS took its time to mature and cultivate as a philosophy and way of thinking, not as a revenue-generating utility. LeSS did it at its own pace, without a rush, while incorporating learning of many coaches and companies that went through LeSS adoptions over years. LeSS has naturally "aged," in a good sense of this word.

Important Point: Whereas, deep learning of system dynamics and organizational design is equally available to everyone who attends LeSS training, not everyone can equally impactfully apply this learning when they go back to work. **But why?**

Lots of LeSS learning (through system modeling using causal loop diagrams) touches upon organizational elements such as HR norms and policies, reporting structures, career paths and promotions, location/site strategies, budgeting/finance processes, etc.—things that are considered to be "untouchable" for an average person (employee).

Of course, it does not mean that an average person is not able to start seeing things differently (they definitely do!) after studying LeSS, but it is just that he may not have enough power/influence to make necessary organizational changes that are required by LeSS. In fact, for many people, this newly gained knowledge which is no longer possible to "unlearn" (e.g., ability think systemically) is accompanied by the realization of one's own powerlessness—and this could be pretty frustrating.

Things are different for people that occupy higher organizational positions. A senior leader is able to combine the decision-making power that is given to him by his

organization and the power of newly obtained knowledge coming from LeSS training. These two powers, united, can have an amplified effect.

Notably, a senior leader who wants to apply LeSS learning to improve his organization must have something else that is very special, in addition to just having general curiosity of the subject and desire to experiment: It is called a "sense of urgency." The best examples of senior leaders that have learned LeSS and then applied learning to reality came from situations where the need to change was urgent and separated success from failure. Then, if the above is true, the formula of LeSS adoption success becomes:

(Organizational Power + Power of Knowledge) x Sense of Urgency

=

Success of LeSS adoption

Important Point: It is strongly **not advisable** for senior leaders to delegate LeSS learning to people that are below them organizationally and, therefore, not empowered to make organizational changes. Granted, individuals at *all* organizational levels will be benefited from learning LeSS. (It is a great eye opener) But senior leaders—people that are empowered to make significant organizational changes—must attend LeSS training in person and not delegate attendance to their subordinates. Leadership should not exempt itself from learning.

In fact, and ideally, senior leadership should attend LeSS training, accompanied by their respective organizational verticals, so that everyone goes through the same learning journey together. Having HR and finance people, alongside with C-level executives and staff members of lower organizational levels is a **HUGE BONUS**.

Mentor-Guided LeSS Case Study Writing Experience Report

Originally published on January 24, 2019 | Location: http://www.keystepstosuccess.com/2019/01/mentor-guided-less-case-study-writing-experience-report/

This writing is about mentor-assisted LeSS adoption case study, written by Certified LeSS Trainer-Candidate ***Gene G [MENTEE]****: Certified Enterprise & Team Coach (CEC/CTC), Certified LeSS-Friendly Scrum Trainer (LFST) / LeSS-Trainer Candidate, Certified in Agile Leadership (CAL) | Certified in Scrum @Scale (CS@S) and assisted throughout by* ***Jurgen D. S. [MENTOR]****: Certified LeSS Trainer, Licensed Management 3.0 Trainer, Innovation Games Qualified Instructor, Black Belt Collaboration Architect*

Purpose of a case study:
The purpose of writing a case study was to relive the experience of **Large-Scale Scrum (LeSS) adoption** by going back in time and memory to everything that was done by me, the agile coach, trainer and organizational design consultant at a large financial institution. This engagement was done in conjunction/partnership with my former trusted colleague **Stuart P.** (also, an experienced agile and software engineering coach). Writing this case study gave me a great opportunity to self-reflect (retrospect) and think about what I could have done differently back then, if I had to go through adoption again. The name of the organization, as well as names of people, products, projects, applications, components, etc., that were involved in the study are intentionally withheld for confidentiality and privacy protection reasons. Nevertheless, hopefully the case study, when published on less.works will serve as a guideline to others in their attempts to experiment with LeSS adoptions in their respective organizations. It is worth noting that many existing LeSS case studies on less.works had provided my former colleague and me with some great references when we worked on our artifact piece.

More About my Mentor:
My mentor, also one of not too many Certified LeSS trainers, was very knowledgeable about LeSS (as trainer, coach and practitioner) and very supportive in my case study work. He and I have met more than once in real life, at various agile- and LeSS-related public events (conferences, retreats), and this allowed for some of in-personal mentoring sessions. Visual technology took care of the rest and made our remote sessions also effective (Note: I am based in the US; he is based in Europe)

Dynamics of case study writing:
The process had been very iterative all along. My mentor and I used google docs, as a communication media, and it allowed us to work incrementally and transparently with one another: typically, I would capture my thoughts directly in the google document, iterate multiple times through them and then, once feeling comfortable enough, would share them with the mentor, asking for his feedback. The mentor would provide feedback, ask questions and suggest clarifications. My former colleague and the peer-coach, who also had full access to the case study, would attend to it at any point in time, leave his comments, provide clarifications and add his details to mine. Notably, my former colleague-coach has helped me significantly by recalling facts, decisions, ideas, events that we lived through together. (LeSS adoption took place a few years before the case study was incepted.. Specifically, my former colleague also helped me significantly in those areas of the case study that talked about technology: architecture, design, and development. In all fairness, this was “our” case study, not just “mine.”

Regularly, at least once a month, when meeting with my mentor, I would receive feedback on those parts of the case study that required further refinement and rework. Many times my mentor would ask me questions that initially seemed to be intentionally tricky or even irrelevant. But I always had to give my mentor the benefit of the doubt that he, being a deep system thinking individual just like me, tried to set me up to think deeper, broader and most systemically into the matter, helping me to discover better ways to formulate my thoughts. Specifically, many of his questions made me go backwards from many of the LeSS experiments that were leveraged during the case study to underlining LeSS principles— and making a connection.

From time to time, my mentor would also share his own experiences and give his own perspective like mine or related situations. This made our mentoring more interactive, engaging and fulfilling.

How did I decide on the scope of my case study?
One of the most important mentoring "aha moments" for me was the decision on how many LeSS experiments that were actually used during LeSS adoption did I really want to describe in detail, as a part of my case study. Here, one of LeSS adoption concepts came to rescue: **Deep and narrow is better than broad and shallow**. I consulted with my former colleague-coach on how many of our LeSS experiments and experiences do we really want to discuss and how deep. We agreed on the shorter list of experiments that represented the crust of our work and could be aligned with logical and chronological sequence of events, as we remembered them. We made our selection described experiments based on what we felt was most important during the adoption, relevant to the case study and memorable to us, as coaches. I consulted with my mentor on the final list and the overall approach and based on his recommendations, proceeded with deeper dives into the case study.

A picture is worth a thousand words.
During one of the many case study reviews with my mentor, it became obvious that long paragraphs and dry text would make many readers bored. This is when I decided to spice up the case study with graphic illustrations and other visual artifacts (e.g., causal loop diagrams, tabular data). I had to make a dedicated iteration throughout the whole case study and introduce graphics were they seemed most appropriate. Ultimately, this made the case study more readable and informative.

Overall experience.
My overall experience of writing the case study was amazing. It took me through the process of additional deep re-learning and self-discovery. It made me reassess my past decisions, now seeing them through the prism of additional experience acquired during the last three years of professional work.

Proper Scaling of Scrum and Financial Forecasting

Originally published on February 5, 2018 | Location: http://www.keystepstosuccess.com/2018/02/proper-scaling-of-scrum-and-dynamic-financial-forecasting/

The purpose of this post is to summarize two very important and independent topics and then integrate them into a joint discussion. The topics are these:

1. *Moving from rigid annual budgets to rolling forecasts (super important! in Agile/adaptive product development environments).*
2. *Quality of scaling in Agile product development, specifically Scrum*

…and ***tying effective scaling of Scrum to dynamic financial forecasting****.*

Rigid annual budgets vs. dynamic/rolling-wave forecasting

Challenges presented by rigid annual budgets have been known for a long time. For people that are new to the topic, a great way to stay on top of most recent research and publications is to follow what is going at BBRT.org (Beyond Budgeting Round Table). One of BBRT's core team members, Bjarte Bogsnes, in his book *Implementing Beyond Budgeting: Unlocking the Performance Potential* (please, refer to the book's highlights here), clearly summarizes the problems with conventional, end-of-year rigid budgets. They are as follows:

1. Budgets represent a retrospective look at past situation and conditions that may not be applicable in the future.
2. Assumptions made as a part of a budgeting process, even if somewhere accurate at the beginning, get quickly outdated.
3. Budgeting, in general, is a very time-consuming process and it adds additional financial overhead to organizations.
4. **Rigid budgets can prevent important, value adding-activities and often lead to fear of experimenting, researching and innovating (crucial for incremental development).**
5. Budget reports are frequently based on subjective metrics as they take on the form of RAG statuses, with the latter introducing additional errors and omissions. (For details, please refer to Red, Yellow, Green or RYG/RAG Reports: How They Hide the Truth, by M. Levison and The Fallacy of Red, Amber, Green Reporting, by G. Gendel.)
6. Budgets, when used as a yardstick to assess individual performance, often lead to unethical behaviors (e.g., "churning and burning cash" at year-end to get as much or more next year) or other system-gaming activities.

The list of adverse effects caused by traditional budgeting is long…

On the contrary, a **rolling-wave forecast** respects the fact that environmental conditions are almost never static and recognizes that if too much reliance is placed on prior years' financial situation, it may lead to miscalculations. Rolling-wave forecasts are based on frequent reassessment of a small handful of strong KPIs, as opposed to a large number of weak KPIs, as frequently done in conventional budgeting. The more frequently forecasts are being made, the higher chance that most relevant/reliable information will be used in assessments. One good way to decide on cadence of rolling forecasts is to align them with meaningful business-driven events (e.g., merchandise shipments, production code

deployments, etc.). It is natural to assume that for incremental/iterative product development (e.g., Scrum), when production deployments are made frequently and in small batches, rolling-wave forecasting could be a concurrent financial process. Short cycle time of market feedback could provide good guidance to future funding decisions. It is worth noting that one of the key challenges that Scrum teams face today is the "iron triangle" of conventional project management with all three of its corners (time, scope, budget) being rigidly locked. And while the most common approach in Scrum is to make scope flexible, "clipping" the budget corner brings additional advantage to teams. Above all other benefits, rolling-wave forecasts address the problem described in **#4 above**, as they provide safety to those teams that want to innovate and experiment.

But what if there's not one but many Scrum teams, each working on their own initiatives, running under different cadences (asynchronized sprints) and servicing different customers? How many independent rolling-wave forecasts can one organization or department adopt before things become too complicated? What is too much and where to draw a line?

Before we try to answer this question, let's review what is frequently seen, when organizations attempt to scale scrum.

Proper scaling vs. "copy-paste" scaling

Let's look at the following two situations: (**1**) more than one Scrum team, independently, doing *their own* Scrum and (**2**) more than one Scrum team, working synchronously on the same product for the same customer, sharing the same product backlog and domain knowledge. The former case is referred to as **"Copy-Paste" Scrum**, clearly described by Cesario Ramos. The latter case can be seen in skillful **Large Scale Scrum (LeSS)** adoptions. Here are some of the most classic characteristics of both scaling approaches:

(1) – "Copy-Paste" Scrum	**(2) – Large Scale Scrum (LeSS)**
• Product definition is weak. Applications and components that don't have strong customer alignment are treated as products. • "Doing Scrum" efforts are often a result of trying to meet goals of Agile transformation (some annual percentage goals must be met), set at enterprise level. • Tight subsystem code ownership. • Top-down, "command and	• Simplified organizational design. Reduction of silos, handovers, translation layers and bureaucracy • Scrum is implemented by coordinated, feature-centric teams, working on widely defined Product for the same PO. • Local optimization by single specialists is

control" governance with little autonomy and self-management at team level. • Importance of Scrum dynamics and its roles are viewed as secondary to existing organizational structure blueprints. • Too many single-specialty experts and very few T-shaped workers. • No meaningful HR changes to support Scrum team design.	eradicated. • Scrum is a building block of IT organizational structure. • Teams are collocated—multi-site development is used for multiple locations. • Strong reliance of technical mentoring and communities of practice. • No subsystem code ownership. • Reduction of "undone" work and "undone department." • Focus on customer values • Strong support by senior leadership and intimate involvement of HR.

Note*:* Please refer to Scaling Organizational Adaptiveness (a.k.a. "Agility") with Large Scale Scrum (LeSS) for additional graphic illustration.

Based on the above, the following also becomes apparent:

In "copy-paste" Scrum, development efforts, marketing strategies and sales (ROI) are not treated as constituents of the same unified ecosystem. In this scenario, it is almost impossible to fund teams by means of funding real, customer-centric products. Why? There are too many independent ad-hoc activities that take place and artifacts that are created. There is no uniformed understanding of work size and complexity that is shared by all teams. Estimation and forecasting made by each individual team is not understood by other teams. Team stability (and subsequently, cost-per-team member) is low, as individuals are moved around from project to project and shared across many projects. Further, with multiple teams reporting into different lines of management, there is a much higher chance of internal competition for budget. By the same token, there is a low chance that a real *paying customer* would be able to step in and influence funding decisions for any given team: too many independent and competing requests are going on at the same time.

In organizations, where "copy-paste" Scrum is seen (and is often mistakenly taken for scaled Scrum due to lack of education and expert-leadership), there is still strong preference for *fake* programs and *fake* portfolio management. Under such conditions, unrelated activities and, subsequently, data/metrics (often fudged and RAG-ed) are collected from all over the organization and "stapled" together. All this information rolls

up to senior leadership, customers and sponsors. Subsequently, what rolls down is *not* dynamic funding of well-defined customer-centric, revenue-generating products, but rather rigid budgets for large portfolios and programs that are composed of loosely coupled working initiatives, performed by unrelated Scrum teams (secondary to conventional departmental budgeting). As rigid budgets cascade down from top onto individual teams, they further solidify the "iron triangle" of conventional project management and hinder the teams' abilities to do research, experimentation and adaptive planning.

On the other hand, in Large Scale Scrum, things are different:

- When up-to-eight LeSS teams work synchronously, together (side-by-side) on the same widely-defined product (real), their shared understanding of work type and complexity (having certain Scrum events together really helps!) is significantly better. As a result, when it comes to forecasting a completion of certain work (features), eight LeSS teams will do a better job than eight loosely coupled teams that work completely independently on unrelated initiatives.
- Since all LeSS teams work for the same customer (product owner), there is a much higher chance that they will develop a shared understanding of product vision and strategy since they are getting it from an authentic source—and therefore will be able to do planning more effectively.
- Having more direct correlation between development efforts, LeSS teams (*output*, in the form of shared PSPI) and business impact (*outcome*, in the form of overall ROI) make strategic decisions about funding much more thoughtful. When real customers can directly sponsor product-centric development efforts by getting real-time feedback from a marketplace and deciding on future strategy, they (customers) become much more interested in dynamic forecasting, as it allows them to invest into what makes the most sense. Dynamic forecasting of LeSS allows companies to increase/decrease number of Scrum teams involved in product development flexibly by responding to increased/decreased market demands and/or product expansion/contraction.
-

Noteworthy that in LeSS Huge cases, when product breadth has outgrown capacity of a single product owner and requires work by more than eight teams, dynamic forecasting can still be a great approach for product (overall) owner and Area Product owners (APO): they can strategize funding of different product areas and make necessary timely adjustments to each area size/grown, as market conditions change.

Conclusion:
All of the above, as described in LeSS scenario, will decrease organizational dependency on fixed budgets, as there will be less interest in outdated financial information in favor of flexibility provided by rolling-wave forecasting that brings much closer together "the concept" (where value is built—teams) and "cash" (where, value is consumed—customers).

Applying LeSS Thinking to Basic Scrum

Originally published on April 4, 2017 | Location: http://www.keystepstosuccess.com/2017/04/applying-less-thinking-to-basic-scrum/

As per Scrum Guide, "Scrum is a framework for developing and sustaining complex products….It [Scrum] is lightweight, easy to understand and difficult to master." The guide talks about basic Scrum, or Scrum by *one* cross-functional team that works for only *one* product owner on *one* single product. The guide also mentions situations when multiple Scrum teams must work on the same product backlog and share the same Definition of Done (DoD)—an example of scrum scaling.

Large-Scale Scrum (LeSS), is a product development framework that extends Scrum with scaling rules and guidelines, without losing the original purposes of Scrum." Some of the hallmarks of LeSS are the following:

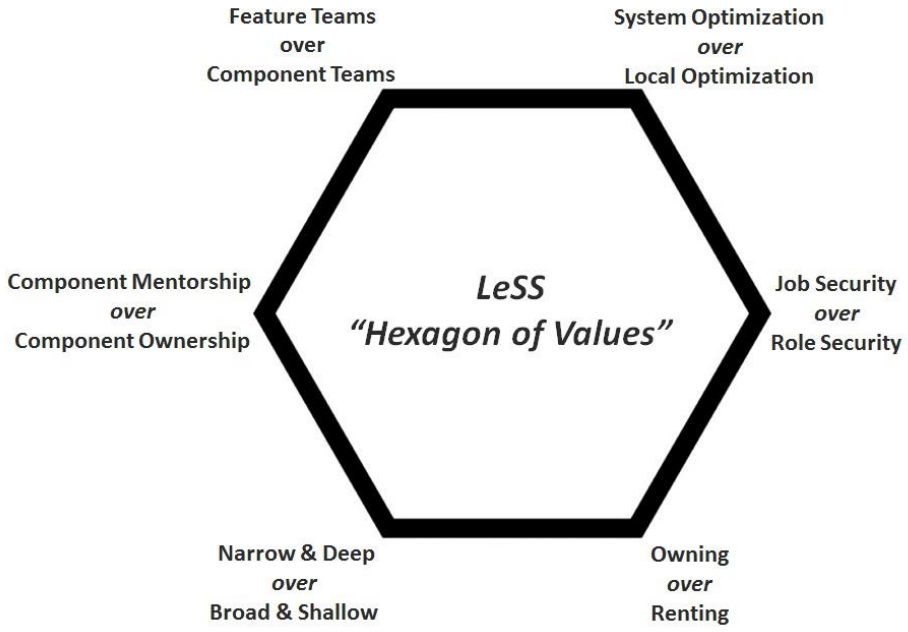
- It is the framework that requires organizational descaling (simplification), to scale basic Scrum.
- It is the framework that strongly relies on effectiveness of organizational design and system dynamics of multiple organizational layers, departments and spheres of control.
- It is not merely a group of teams doing their own Scrum while working for different product owners and supporting different products. Rather, it is a group of teams (two to eight) that works together, synchronously, on the same product and serves the same product owner.
-

It is worth stressing the last bullet point above:

Frequently, novice Agile adapters mislabel Scrum implementation by independent and unrelated teams as "LeSS." Sometimes, they are genuinely mistaken. But there are instances of intentional LeSS terminology overload. In either case, inappropriate use of LeSS terminology creates asymmetry between LeSS guidelines and rules, on one hand, and reality of teams' working dynamics, on the other.

Another prevalent misconception about LeSS (and more on this later in this post) is that LeSS postulates are ONLY applicable in large scale settings. Some less experienced Agile practitioners feel that if an organization implements Scrum in its basic form only, without attempts to scale, the facing organizational dysfunctions that are vividly exposed by LeSS could be avoided. This is a mistaken belief: LeSS principles can be used to improve basic Scrum as well. Practically, any organizational dysfunction that is exposed by LeSS also presents a challenge for basic Scrum, but perhaps in a form that is not so obvious and with a manifestation that is not so disturbing. In a way, LeSS serves the purpose of "dysfunctions amplifier/magnifier" that allows seeing more clearly at scale those things that may not be as obvious at single team level.

Below are the six big topics (LeSS *"Hexagon of Values"*) that are always brought up in LeSS discussions. Let's take a closer look and see if they still retain their relevance in basic Scrum:



Feature teams over component teams

From LeSS perspective:

From a stand-point of a paying customer, feature-centric product development is the most effective way to maximize business value. Feature teams can perform this work best, since they are able to operate across multiple application components, independently and autonomously. Component teams cannot do the same. A component team is focused only on one single application domain that is not shippable (marketable) on its own, if viewing from a standpoint of paying customer. Trying to fake LeSS by putting under the same wrapper multiple component teams creates a large amount of handover and other procedural overhead before a release. For example, if eight component teams (maximum number in LeSS) work only on respective single component items, pulling them from the same backlog, there will be a lot of component integration required before a product becomes truly shippable (PSPI). This will either, at best, consume a lot of the teams' capacity at the end of each sprint, or it will require an additional "*hardening*" sprint before going to production (at worst).

How is this applicable to basic Scrum?

It is not uncommon to see a single component team that pretend "doing scrum" by introducing Scrum roles, artifacts and events to their work model. At a glance, it may appear that a team does what other scrum teams do—it sprints, but the goal of scrum—to

deliver PSPI—is still absent. The impact of such faking on the ability to meet a product owner's goals may not be as strong and obvious because delivery expectations from a single team just are not as high as expectations from eight LeSS teams. (In other words, from a standpoint of time and resources spent and humans involved in performing work, a single component team that pretends "doing scrum" would have an easier time to justify to a product owner and stakeholders *why* their end-of-sprint deliverable is not shippable: justifying undone work by three to nine developers of basic Scrum is just easier than justifying undone work by 50 developers of LeSS). However, the core underlying problem still exists in basic Scrum: a sprint deliverable is not PSPI.

Component mentorship over component ownership

From LeSS perspective

There is a distinction between component ownership and component mentorship:

Component Owner is an individual (sometimes, a group) that has *unique* privilege to work on an application component. If any other team needs to make modifications to a given component, they must delegate this work to a component owner. Rarely, these "private code policies" are justified by external regulatory requirements or internal audit needs. More frequently, such ownership is due power retention and sphere of control ambitions of certain individuals/groups, both being secondary to internal organizational politics.

On the other hand, Component Mentor is an individual that has a lot of work experience with a particular component, but instead of becoming a bottleneck to component-specific work, he teaches others how to do the same. Effectively, a component expert becomes a teacher of "how to fish," instead of a giver of an "already caught fish." With such an approach, any feature team, over time, becomes proficient at working on any application component without being dependent on a component owner: each component becomes a shared property; no more someone's private and protected domain.

By giving up the privilege to be the only person who can touch an application component, an application mentor should not lose his organizational value: instead of being a single-contributor/performer he now becomes an educator and advisor—the role that is much more scalable in large organizational settings. This emphasizes another concept (below) that is strongly advocated by LeSS trainers and coaches: Job Security should not be confused with Role Security.

How is this applicable to basic Scrum?

Perhaps, the most important distinction in basic Scrum is that for a single-team operation, the benefit of having experienced and willing to share their knowledge with others, component mentors, is not as obvious. Also, an organization may have a hard time to

justify designating a component mentor, per component, serving a stand-alone and unrelated to others, scrum team. But even for a single team that does basic scrum, the problem remains: not being able to work directly with certain application components will continuously lead to having external dependencies on other teams/individuals, and therefore, will put at risk the ability to deliver PSPI at the end of each sprint.

System Optimization Over Local Optimization

From LeSS perspective:

There is strong contrast made between local optimization and system (global) optimization. Any single-specialty department (UI/UX, BA, QA, Dev, DB) is *locally optimized*. This means that people within each department might be very busy and work hard to complete a series of tasks that ONLY they can do. The opposite is true too: a group of single specialty workers can perform ONLY a subset of tasks that their skillset allows. While from their own (local) perspective, they work very devotedly and produce a lot of *output*, from a standpoint of a paying customer, the overall *outcome*s are still low. False perception of local efficiency by single-specialty workers does not directly translate into system efficiency: individual "local progress" does not add up to system progress. A classic example of local efficiency could be over-production of comprehensive BRDs by business analysts (BA), way ahead of development efforts: while from a standpoint of a BA, lots of great documentation is produced and signed off; from a standpoint of a paying customer, much of this work may become obsolete (projects are delayed, priorities change, budgets get cut, etc.) before implemented in code.

How is this applicable to basic Scrum?

Manifestation of local optimization, also being a direct result of single-specialty (component) work, is frequently seen on basic Scrum teams that lack T-shaped developers. For example, nontechnical business analysts and manual testers will be, most likely, focused on "writing stories" and producing manual test cases, respectively, way ahead of sprint development. For example, it is not uncommon to see BAs attempting to *clarify* user stories by adding to them a lot of conventional, contractual documentation (e.g., BRDs) that leads to reduction of communication with a product owner and stakeholders during sprinting; this introduces elements of mini-waterfall in Scrum.

Job security over role security

From LeSS perspective:

Any person should have a job and can make living. Offering job security to each employee should be one of the most important goals for any organization. However, job security should not be equated to role security. Gradual changes of needs in any role could be a natural process that follows bigger industrial changes: modernization,

computerization, robotization, etc. Efficiency in an overall system organization should not be sacrificed for preserving specific roles that no longer bring value.

One of the most common role changes discussed in LeSS is the role of project manager (usually, first-line PM). Scrum requires self-organized and self-managed teams that take full responsibility for their own work and interaction with customers. Everything is done by a team that performs work: work estimation and assignment, ownership and information sharing, communication and reporting. Historically, all of this has been handled by a project manager. Shifting all these responsibilities to a team makes the PM feel underutilized, sometimes annihilated. In environments where teams no longer require tight managerial control, managers may have to refocus on removing organizational challenges that teams face and cater to teams everything that is required to optimize team work.

It is critical that any organization that reevaluates its existing roles also takes a close look at existing job families and employee career paths. If an organization suspects that some employees might be displaced due to changes of roles and responsibilities, it must provide other alternatives for people: education, training, internal mobility to other parts of an organization, etc.

How is this applicable to basic Scrum?

The need to provide job security for potentially displaced workers in basic Scrum settings is no less important than in LeSS. Business analysts and manual testers should be given an option to move into Scrum teams and learn additional skills. In basic Scrum, unlike LeSS, where there is a much stronger need for addressing and resolving organizational impediments, it could be less obvious how a first-line manager of a single Scrum team can be fully occupied with removing impediments, as this is usually Scrum Master's responsibility. (In basic scrum, at single team level, impediments may also be not as systemic.) In such cases, line managers should be given an option to move to another part of an organization where traditional development is still practiced.

Narrow and deep over broad and shallow

From LeSS perspective:

When it comes to large-scale Agile adoptions, it is very easy to develop a false assumption that bigger is always better. Some organizations are trying to jump on a bandwagon of Agile-mania and claim early victories. (This is often caused by ill-defined motivation and bonus schemas that praise people for achieving certain fabricated goals, e.g., by "rolling out Agile" to a very large part of an organization.) Such attempts of broad coverage also lead to shallow impact: organizations can demonstrate short-term superficial (sometimes, fake) results, but there is not enough momentum to ensure continuity and long-term success.

It is always much better to focus on a narrow part of an organization by extrapolating it from a larger organizational construct and improving organizational agility by going deeper into organizational structure. In LeSS, the recommended size of an organization (at least, on the technology side) is about 50 people.

How is this applicable to basic Scrum?

This concept is probably not as applicable to basic Scrum as other concepts mentioned so far. Breadth of implementation implies wide organizational areas and involvement of many people. Basic Scrum is not as concerned with horizontal span of efforts because, by definition, a scrum team is only three to nine people.

Owning over renting

From LeSS perspective:

When it comes to making decisions, organizations and teams should look for ways to develop their own approaches instead of copying and pasting solutions of others. Relying on professional trainers is a great way to learn from the expertise of others in classroom settings. Leveraging help of professional coaches is another, longer-term way to mature through self-discovery, experimentation, inspection and adaptation.

Organizations and teams should be striving to own their own decisions and take responsibilities for successes and failures. On occasion, trainers and coaches may lead by example (e.g., mock-up a behavior of a specific Agile role) and share lessons learned from prior engagements, but this is not done to encourage customers to copy/paste approaches and styles. Ultimate decisions should be always made by those that will live with those decisions in the long-term.

How is this applicable to basic Scrum?

In basic Scrum, just like in LeSS, teams are expected to leverage what they have learned from trainers and coaches to gain autonomy and become owners of their decisions and solutions. Through frequent inspection and adaptation, teams that work in basic Scrum should be able to ultimately modify their processes, as needed. Logically, it would be also expected that a coach who assists team(s) that use basic Scrum, coaches himself out of work sooner than a LeSS coach, since organizational design challenges have much more severe impact on LeSS teams that on basic scrum teams.

Diagnosing Challenges in Scrum

Originally published on September 13, 2017 | Location: https://www.frontrowagile.com/blog/posts/10-diagnosing-the-challenges-in-scrum

When people go to the doctor's office, they often complain of superficial manifestations of a problem (i.e., “chief complaints”) that don't present any serious concerns but in reality are indicators of a much more serious systemic illness.

I've come to the conclusion that we in the Scrum field are also dealing with a similar type of scenario.

For those of us who have been coaching for a while, it's probably a common experience to have a team member or a manager come to us with what appears to them as a “key problem,” but in reality turns out to be a symptom of a much more serious underlying dysfunction.

On June 1, 2015, there was a one-day Agile conference held in New York by a well-known local Agile community: Big Apple Scrum Day. It brought together more than 250 Scrum and Agile practitioners from around the globe.

I organized the “Scrum Coaching Clinic” for the event. I personally coached about a dozen people that day, and my experience only solidified the notion that it’s crucial to trace a presented problem back to its original root cause.

From that event, here are some examples of local/topical challenges that were initially presented as “key problems” by attendees but were then traced to more systemic issues.

Example 1

Symptom (chief complaint): Lack of honesty in retrospectives. Individuals very reserved in retrospectives.

Details: Individuals are very reserved in retrospectives. Too many generalizations, not enough specifics. Real problems are not discussed or downplayed. Actionable items are not identified. Too much political correctness and fudging.

Further discovery: Direct line management attends retrospectives, usually at senior management’s command. Retrospective recaps are viewed as “official documentation” to be shared widely. This puts many people in harm's way (reprisals are not uncommon). In retrospectives, management frequently looks for individual accountability, as opposed to team accountability.

Example 2

Symptom (chief complaint): Miscommunication in executive reporting.

Details: Team-level metrics and data (velocity, throughput, forecasted dates) are not easily "translatable" to communication and reporting that is presented to senior management.

Further discovery: Metrics and data collection at the team level are not reliable in the first place (e.g., velocity is very volatile). As information travels up the chain of command, it is formatted to fit what is perceived as the “usual format” at that level. Along the way, data gets skewed and fudged. Individuals with limited understanding of work size and complexity also contribute to estimations. While Agile implementation is attempted at a team level, senior management is not educated enough to comfortably read Agile metrics and reporting.

Example 3

Symptom (chief complaint): Lack of reliability with estimation techniques.

Details: Velocities are volatile. The same work is estimated differently by the same team at different times.

Further discovery: Inability to normalize work size and translate it into “money.” Poor teaming. Lack of cross-functionality. For every developer (coder), there is at least one (non-coder) whose understanding of work size and complexity is rather limited. Team members don't get a chance to work together long enough due to resource reshuffling and can’t “normalize” their view on work.

Example 4

Symptom (chief complaint): Difficulties with talent retention.

Details: Companies heavily rely on temporary staffing instead of building their internal technical expertise.

Further discovery: Acquire unpredictable quality staff (usually lower quality workers at a lesser cost) who must soon depart a company due to HR policies. As a result, talented people are in scarcity, and so is knowledge retention. Continuous knowledge transfer is required with much of the knowledge falling through the cracks and getting lost, as people tend to protect their jobs by taking their knowledge with them as they leave.

Example 5

Symptom (chief complaint): Lack of synergy with product ownership.

Details: Odd relationships between product owners and teams. At times, direct reporting lines between POs and team members. At times, multiple “equally important” stakeholders stepping into a PO role simultaneously.

Further discovery: Very few POs have the combination of knowledge, authority and willingness to intimately engage with technology. Frequently, the PO role is assigned as a mandate and is in competition with other responsibilities of an individual. Poor definition of a customer. Difficulties finding an individual whose "paycheck" truly depends on product development success. There is no traceability between efforts and money spent on development and extra revenue generated as a result of development (i.e., no analytics on ROI).

Conclusion

While speaking to individuals that day at the clinic, I came to the conclusion that very few are actually willing to travel "upstream" from a topical problem to its deeper root cause.

As Scrum coaches, we should refrain from jumping too fast to conclusions. Sometimes, the effect of anchoring is too strong and hard to resist; the initial information presented may be loaded with hidden anticipation for a particular recommendation—a trap that we, as coaches, should avoid falling into.

By emotionally detaching from the company and people we are coaching while at the same time collecting information and not allowing ourselves to be led into false and premature conclusions, we are improving our chances to build a more objective, unbiased view that spans far beyond superficial symptoms and much deeper into organizational layers.

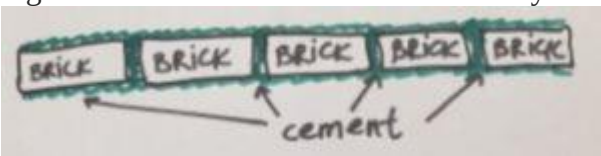
LeSS "Construction": What Is It Like?

Originally published on July 29, 2017 | Location: http://www.keystepstosuccess.com/2017/07/less-construction-what-is-it-like/

[Also, cross-posted on less.works.]
Large Scale Scrum (LeSS). It is the framework for scaling Agile development, done by multiple teams as they work on the same product and work for a single product owner. In order to be effective, LeSS requires organizational descaling that means simplification/flattening of organizational design.

What is **organizational design**? To understand it better, let's look for an analogy in the construction industry. What is required to erect a building? In our analogy, we shall stay simple: bricks (foundation block) and cement (connective material that holds bricks together).

Imagine two buildings: **building A** and **building B**.
Building A uses brick as its *main* foundation block. In fact, when looking at the building's facade, the most prevalent object caught by a naked eye is brick. Bricks are positioned next to one another with just enough cement inbetween to glue them strongly together. There is no excess of cement anywhere: the connection layer is very thin/lean.



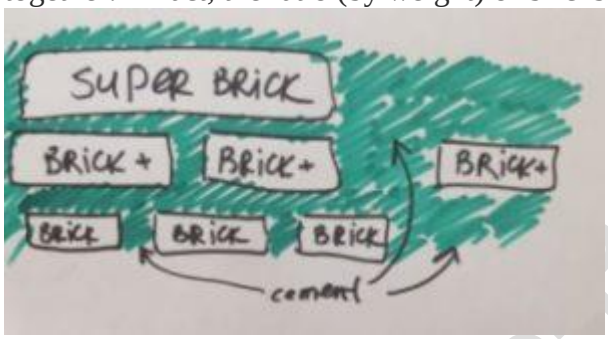
Architectural design of building A is simple and flexible: the structure is flat (one-story high), and it sits on a strong foundation, also made of brick. Because of its design, architectural adjustments are possible in various sections of the building, independently, with little additional labor. Due to such modular structure, the building can be expanded laterally just by adding more bricks to the wall. Of course, due to its flat structure, the building is also very stable and can withstand a strong wind, flood or an earthquake: practically nothing can be shaken off or washed off the building.

When waste is produced inside the building, it becomes noticeable immediately. Waste disposal is also very simple: it does not require complex chutes or automated waste packaging systems. Waste removal can be mostly done manually by building residents. Any necessary supplies (e.g., food, water, furniture, other materials) can be easily delivered to any building area without the need of advanced technology or mechanics.

Finally, building inspection and maintenance is a very easy process because of the flat structural design: foundation, walls and floor assessment—all can be performed with a naked eye; corrections can be done timely and efficiently.

This is what building A looks like:

Building B is made of a very few bricks and a lot of cement inbetween that holds bricks together. In fact, the ratio (by weight) of bricks-to-cement is very low.



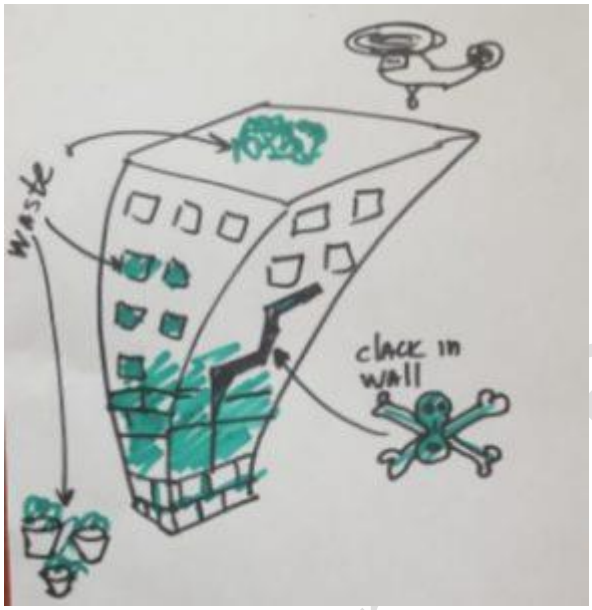
Architectural design of building B is rigid. It has many floors, with top floors made primarily of cement. The building represents a heavy and monolithic structure, and although it also sits on a brick foundation, as building A, the bricks are widely spaced with lots of cement inbetween. This means that the overall weight of building B is dangerously high (foundation can crack). The building's expansion limit, to accommodate growing occupancy demands, is low: It cannot be easily extended (scaled) horizontally with a couple of extra bricks added to the side, because the bottom brick layer would require multiple horizontal cement layers added on top—to follow the originally intended building design. If additional cement layers are added on the top of the foundational brick layer, it will further increase risks of foundation cracking.

Waste disposal is a serious issue for building B. While waste can be relatively easily removed from the bottom floor (it is also not in abundance there) and, to some extent, from top floors (by taking it to the roof and using a waste removal chopper), there is a huge amount of waste that gets accumulated at middle floors—and it sits there. It is extremely challenging to remove this mid-section waste, and what building management does from time to time is order this waste to be moved from one floor area to another (the building is very compartmentalized). Sometimes waste gets moved to floors above; sometimes below. This creates an illusion of waste removal. But waste remains.

Delivery of supplies and food to building B occupants is a real challenge, especially if elevators are out of order. This makes occupants angry and frustrated and sometimes they turn on each other, becoming competitors and rivals.

Finally, building inspection and maintenance is a nightmare for building B. Many living units are out of compliance with building codes, but violations (and violators) are hard to identify and remove because the true facts are well concealed, and numbers are gamed by building occupants.

This is what building B looks like:



Large Scale Scrum requires organizational design that is analogous to the construction represented by **building A**.

In LeSS:
Team represents the main building block (a brick). Selected team representatives (developers) and mentors-travelers ensure effective coordination/connection between teams. There are no additional roles required for coordination. Cross-team events are minimal (overall product backlog refinement, sprint review, overall retrospective).

If product definition widens and more developers are included, another team can be formed and positioned *laterally* to existing teams—just like a brick. Should product definition become too wide and the number of required developers exceeds 50 to 60 people (eight teams), another product area can be identified (new independent module, made of bricks). Now, LeSS becomes LeSS Huge. The only additional coordination that would be required in LeSS Huge is between area product owners and the overall product

owner—for strategic planning of Potentially Shippable Product Increment (PSPI) at the end of every sprint. In both, LeSS expansion from two to eight teams, and LeSS Huge expansion beyond eight teams, there is no need for additional coordination that is different from what is described above (no extra cement needed to keep bricks together). Also, in LeSS Huge, when one product area expands and another one shrinks, moving the whole team from one area to another does not require expansion or shrinkage of any additional "supportive" organizational layers.

By design, LeSS foundational structure is very lean: flat, fungible and cross-functional. There is no waste or overhead with roles, responsibilities, events or artifacts. Everything is very minimalistic. If any waste is generated in LeSS, it has practically nowhere to hide.

Because there is so much transparency in LeSS, waste is seen immediately. Any findings of waste or any other required improvements to individual teams or LeSS framework can be effectively done in Team Retrospective or Overall Retrospective, respectively. Thanks to its flat organizational structure, LeSS (and LeSS Huge) don't have to worry about waste removal from additional organizational layers—they (layers) just don't exist. There are fewer layers that sit between LeSS teams and LeSS product owners and these layers are much thinner.

What happens with LeSS organizational structure during rough times: slow down in business, increased market competition? Arguably, because LeSS is so lean and there is continuous learning, it is much less likely that LeSS people will be displaced. LeSS is also more likely to withstand other types of reorganiztions and shake-ups because LeSS has very few moving parts, loose pieces or weak links.

Organizational designers that support LeSS think like building architects that want to build strong, reliable, easily maintainable, low-waste, cost-effective and long-lasting structures!!!

Many thanks to all LeSS Trainers, Coaches and Practitioners building reliable structures **.**
Signed: _____*The Organizational Building Management*

SAFe: Market Share Increase. Rapid Growth. What Is the Recipe?

Originally published on May 7, 2016 | Location: http://www.keystepstosuccess.com/2016/05/safe-market-share-increase-rapid-growth-what-is-the-recipe/

This top paragraph is an amendment to the original blog post, created by me back on May 7th, of 2016.

What drove me to make this amendment was the recent webinar recorded by VersionOne: How to use SAFe® to Deliver Value at Enterprise Scale Q&A Discussion with Dean Leffingwell. If you fast forward to about **23 minutes, 20 seconds** into the recording, you will hear the following statement: "**We don't typically mess with your organizational structure because that is a pretty big deal.**"

This statement somewhat puzzled me. While graphic representation of SAFe framework does not shy away from supporting organizational complexity, I was still under the impression that organizational redesign is included in SAFe teaching. To me, just like to any organizational coach or system design consultant who helps companies with improving organizational agility, the ability to influence first-degree system factors and variables, such as **organizational structure**, is critical. Without this ability, any attempt to improve organizational agility and system dynamics would be short term and limited. Even such important *second-degree* system variables as organizational culture, values, norms, behaviors, policies, Agile engineering practices usually bring limited results if organizational structure remains unchanged.

But regardless of my recent new learning, I had to admit to myself that SAFe still remains a very successful and popular product among many large organizations. And at this point, I would like to refer the reader to my original post below.

Lately, there has been so much buzz in the Agile arena about scaled Agile frameworks. I just came back from the Scrum Global gathering in Orlando where I heard a lot of discussions about Agility at scale and various existing Agile frameworks that are in use. Following Orlando discussions, I have seen a wave of email exchanges and blogs on the same topic, some of which involved seasoned organizational coaches and trainers. I have noticed that there has been a lot of focus on **SAFe** (Scaled Agile Framework): opinions, comments, attempts to compare to other Agile frameworks. There are two things that strike me as odd:

1. It seems that some seasoned coaches and trainers don't explicitly state their views. When I read indirect statements or views, I continue wondering how a person really feels about the subject.
2. Among blogs and other posts that I have seen, I was not able to see any discussions that cover aspects of SAFe that are of interest to me.
3.

But before I go any further, here is my **personal disclaimer:** I am neither a SAFe practitioner nor trainer or coach. I have not attended a comprehensive SAFe course. However, I have studied/researched SAFe extensively on my own. And I do know some companies that have implemented SAFe. (I have talked with some of their employees.)

And I do know a significant number of individuals that have been trained in SAFe. And I do know a handful of respected coaches that recommend SAFe. So basically, I understand SAFe beyond just knowing how to spell it.

Now, let me put the SAFe topic to the side for a moment and shift gears to something else. (We will all come back to SAFe in a minute.)

I want to bring up the topic that has been beaten to death for everyone who understands Agility: the topic of *tooling*.

When it comes to discussions of Agile tools, most experienced Agile coaches have a long arsenal of arguments to use with their clients, prospects or less seasoned Agile enthusiasts. Here some classic examples:

- First postulate of the Agile Manifesto: "Individuals and interactions over processes and tools."
- "A fool with a tool is still a fool."
- "The best tool in Scrum is a whiteboard (or Excel, at most)."
- "Agile tool is not a right solution for your deep organizational problem."
- "Never begin your Agile education with tools. Always learn principles and concepts first."
- "Agile tool is a poor substitution for collaboration that you may never have. If you start exchanging information through a tool, you will lose the benefit of live discussion. If you just need to introduce a tool, do it later in a process when people gain sufficient amount of knowledge and experience."
- Etc, etc, etc.

We, as coaches, are never shy to express our strong views (sometimes, overly strong) about tools **NOT** being a good solution to organizational problems and **NOT** being the best method (by far) to transform organizations. And I am glad we are not shy about that. Therefore, we are called Organizational Coaches—we look at organizations holistically. For us, tooling is just a tiny fraction of a much bigger organization puzzle. And personally, I never heard organizational coaches disagree too much about the bullets above: We are all very much on the same page; we stand together on this, and we cannot be fooled.

<SIDE NOTE ON>

But I still want to be very transparent about myself with regards to tooling, so here is another **personal disclaimer**: over the last decade, I have been around and have gained a lot of experience with tools like JIRA, Version One, Rally and others. I consider this as a personal "hobby," but I know how to decouple it from daily work that must be done by an organizational coach. Over the years, I got to know some great software engineers that have built the tools mentioned above. I could probably easily pass for an in-house "Agile tool expert" (that is if I choose changing my profession one day) and find a job that says something like this: "Looking for a strong Agile tool expert to transform and move our

organization to the next level. PMP certification is a huge plus. You must also have a valid H1B Visa." Yes, sadly there are many job specs out there that sound just like this . On a brighter side, I could probably also leverage my hobby and look at any Agile tool used by a team or a group of teams that claims, "to do" Agile, and in about five minutes I could find a handful of signs of serious systemic dysfunctions (just in a tool alone!). So, there is some practical use of my hobby. In any case, I think I have earned the right to say that I know very well what tools can and CANNOT do for you. And therefore, I strongly stand with all other coaches that use the arguments I listed above.

<SIDE NOTE OFF>

Here is my conclusion: "*It seems that organizational design experts and system thinkers, change agents and Agile coaches/trainers are pretty comfortable to strongly and openly state their views in cases where they feel they are stating the obvious—for example, about tooling*." Let's hold on to this thought for a moment.

Now I would like to come back to the topic of SAFe and set the stage for my questions by stating the following:

High Market Penetration of SAFe:
First of all, let's take a look at some relevant data that has been recently published on InfoQ, with the original source being Version One, 10th Annual State of Agile Survey. While still being a relatively new framework, SAFe has acquired a significant share of marketplace—**23%**—while demonstrating the highest rate of growth: "…the largest increase from 19% in 2014 to 27% in 2015."

My understanding of safety that SAFe brings:
I have heard various opinions about what went into thinking of the acronym "SAFe." Was it an intention to make it sound phonetically as "safe," or was it just coincidental that the words Scaled Agile Framework that begin with "S", "A" and "F", respectively, made up SAFe? I don't know. And I don't want to speculate. But let me share my understanding of what some more obvious safety factors of SAFe are. But again, this is my view only:

- SAFe does not seem to be threatening to first-line management. Thanks to its first two layers (Team/Program and Value Stream) and abundance processes and roles that are present in both, everyone can find a place to work. In other words, the degree of fear about being misplaced or losing a job is relatively low. If we all recall what happens with implementing basic Scrum, where teams are expected to become self-organized and self-managed, and where the role of project manager is not explicitly discussed, we (coaches, trainers) frequently must answer the following question, usually coming from managers: "What now happens to my role?" And of course, there are ways to handle this question properly and give good options to those who ask, but this is not my point. My point is that I don't expect this question to be asked as frequently with the introduction of SAFe. Why? Because SAFe seems to be a good way to harbor

many existing management roles (role security), although, perhaps under different naming convention.

- SAFe looks "homey" to senior management. A SAFe graphic is very rich in colors, objects, lines, layers, and icons that represent roles, groups, departments, interactions. At a glance, SAFe appears as a natural fit and a comfortable habitat for many existing organization constructs. There is no explicit invitation to significantly challenge/simplify existing organizational design; no hints to change/simplify reporting lines or flatten layers (descaling). No need to have unpleasant conversations with employees(!). Senior managers that are confident that their organizations are well designed and don't need any major repairs see SAFe as a safe way to *try* Agility.

- SAFe does **NOT** explicitly compete with other Agile practices. SAFe uses them all. In fact, a cute yellow smiley-squeeze-toy that many folks picked up in Orlando from a SAFe kiosk explicitly says: "*SAFe embraces Scrum*." Indeed, at its multiple layers, SAFe diagram mentions Scrum, Kanban, XP, as well as many roles, artifacts and ceremonies and iterations that support all these practices. And this, IMO makes SAFe really safe, in a very special way. If Company X already uses, perhaps inconsistently, some Agile practices, it is relatively safe and actually convenient for SAFe consultants to come in and say something like this: "We can help you retain most (if not all) of what you have adopted so far, but it will be much better structured under the overarching umbrella of SAFe." In other words, a SAFe consultant does not have to worry too much about upsetting people of Company X by making them give up what they may have grown an attachment to—they can keep it all. Effectively, in my mind, SAFe acquisition does not have to translate into workforce/practice reduction. To me it seems to be a pretty safe offering.

My understanding of SAFe Partnerships and Strategic Goals:
Here, I am listing only the top few references that I found online. But the list could be much longer if I spent more time searching. I personally have attended a handful of webinars, where concepts of SAFe were presented, alongside with the benefits of tools (by companies that hosted webinars).

Please, finish reading the post first and then come back to the links.
With Rally:
- Rally Portfolio Manager Highlights
- Scale Agile SAFe®ly
- 10 Things You Need to Know About SAFe 4.0
- 41 Things You Need to Know about the Scaled Agile Framework® (SAFe)

With Jira:
- Announcing JIRA Portfolio: View, plan, and manage initiatives
- Scaling Agile in the enterprise with SAFe and JIRA Agile
- How do I setup JIRA and JIRA Agile for the Scaled Agile Framework?
- JIRA SAFE – THE EASIEST WAY TO MANAGE WORK ACROSS PROJECT, PROGRAM, AND PORTFOLIO

With Version One:

- Implementing Scaled Agile Framework® (SAFe®) with VersionOne
- Scaled Agile Framework® SAFe®
- Scaling Agile with VersionOne
- VersionOne Accelerates Enterprise-Scale Agile with SAFe® 3.0 Alignment

Just to be clear for those that may not be as well familiar with these tools as I am (you don't have to share my hobbies), each one of these tools now has complex Strategic Layers that sit at the top of the tools' "tactical" layers and constructs (epics/stories, backlogs, sprints, releases, team views, Agile boards, story/task boards, workflow management, etc.)—and they are meant for project, program and portfolio management. At some companies where I have consulted, each one of these layers typically had a manager (Project Manager, Program Manager, Portfolio Manager, respectively, etc.), someone who was responsible for data collection and status reporting—and that was *without or prior to* any implementation of SAFe (or any other scaled solution). Tool complexity alone was sufficient to offer a nice fit to an existing organizational structure.

<SIDE NOTE ON>

What is also probably not a surprise to anyone is that there are so many large companies out there that own tens of thousands of licenses for the above-mentioned tools. I have been to many such companies and have seen these tools being a "hallmark of organizational agility" to too many people that understood very little about true organizational agility. Please note that very frequently "best practices of use", even for Agile tools, reside within departments like Control and Governance, PMO, and Centers of Excellence, where decisions about "what is best" are made in a vacuum and then are cascaded down to organizational domains that are thousands of miles away.

<SIDE NOTE OFF>

So, here is another safety aspect of SAFe that probably belongs to the previous section of my post but I will keep it here:

SAFe is very safe to client-to-vendor relationships: it does **NOT** disrupt existing million-dollar (of course, depends on company size) contracts and license agreements between client companies and tool vendors. It should be safe, IMO, for SAFe consultant to come in and say something like this: "If you are using JIRA or Rally or Version One or any other tool that has a portfolio management layer in it, it will be very complementary to what we can do for you in terms of Agile scaling." I think that the links that I have provided above suggest exactly that.

All in all, SAFe seems to be a great complement and strategic alliance to some Agile tools companies that have gained a lot of their own market share over the years. And it does not matter that JIRA and Version One and Rally or others could be competitors to each other. They all seem to be great partners of SAFe. (I will not speculate on exclusivity of relationships, but based on the links above, there may not be any.)

Now, after I brought to light some relevant market data, shared some personal views on what I consider as "safety factors of SAFe" (some people may see them as true benefits of SAFe) and gave a perspective on some possible strategic alignments that may exist between SAFe and industry leaders in the world of Agile tooling, I would like to ask the following two questions and make one small request:

- First Question: Do you think that market penetration of SAFe and its adoption success could be attributed to the personal safety of companies' managers, as I have described above? Do you feel that role security of first-level management *is* a significant contributor to SAFe adoption rate? I stress this last point because the role of first-level manager is in super-abundance today at many companies.
- Second Question: Do you think that market penetration of SAFe and its adoption success could be attributed to (at least in part) its direct or indirect alignment with industry leaders that build Agile tools? Do you think that "SAFe + XYZ tool" produces a stronger compounded effect on organizations in terms of SAFe adoption, than SAFe applied alone?
- Third—Request: I would like to respectfully request from my colleagues, coaches and trainers when they talk about any topic (in this case, SAFe is the topic of choice, but it could be anything else) to be more open-minded and clear in their views. For example, if someone sees the same system relationships as I described above, and he is comfortable sharing his strong views about tools, would it not be reasonable to expect the same person share his explicit views on related frameworks? And the opposite should be true too: If there is no comfort to explicitly discuss frameworks, why then be so adamant to state your views on tooling?

The last request is an invitation to system thinking.

From the LeSS Toolbox: Causal Loop Diagrams to Visualize System Dynamics

Originally published on 13 July 2016 | Location: https://www.scrumalliance.org/community/articles/2016/july/from-the-less-toolbox-causal-loop-diagrams-to-visu

Introduction

When it comes to scaling, there is a common misbelief that "bigger always means better." This misbelief is also traceable to the Agile arena where companies look for ways to expand their Agile practices beyond a single organizational domain (e.g., many teams, numerous departments, multiple lines of business, etc.). Usually it is an existing (inherited) organizational complexity that becomes the main reason why companies look for complex, multi-tiered scaling solutions. And, of course, if there is a demand, there will be a supply: Many frameworks are out there that hand-hold companies to comfortably "embrace" their existing complexity and not feel too uncomfortable about their own internal dysfunctions.

However, not all scaling solutions are as "forgiving." There are some Agile frameworks that intentionally expose and boldly challenge organizational deficiencies. One such framework is Large-Scale Scrum (LeSS).

To set a stage for the rest of this discussion, I would like to summarize a few points about LeSS here.

I also would like to express my appreciation and acknowledgment to Craig Larman (one of the cofounders of LeSS) for helping me deepen and broaden my understanding of organizational design and improve my system thinking which I have been developing over years.

Overview of LeSS

LeSS is a very easy to understand. I like to speak metaphorically, so in describing LeSS, I sometimes use the analogy of the legendary assault rifle AK-47, which has the following well-known characteristics:

- It has very few moving parts and, therefore, its internal friction is low; also, there are not too many small pieces that can jam or break.
- It is simple to disassemble, inspect, and reassemble (inspection and adaptation).
- It is very reliable and adaptable under tough conditions. (It rarely fails in action.)
- If necessary, it can be modified and "expanded" at low cost and/or with low effort.

But there is something else about LeSS that makes its analogy to a weapon (probably not just to an AK) appropriate: *It assaults organizational dysfunctions.*

LeSS also has two important characteristics:

1. It is very simple in design and rests fully on the core principles of basic Scrum. (Effectively, LeSS is the same as Scrum, as it is described in the Scrum Guide, but it is performed by multiple teams.)
2. LeSS teachings rest on the pillars of:
    - Lean Thinking: "Watching the baton, not the runner," visual management, cadence, timeboxing, managers being teachers, continuous improvement.
    - Systems Thinking: Weinberg-Brooks' Law, queueing theory, indirect benefits of managing batch size and cycle time, being customer-centric, explaining differences between local and system optimization.

Thanks to these two key characteristics, LeSS is a very powerful mechanism that helps see an organization systemically/holistically while identifying and exposing (the analogy to a high-power rifle scope is suitable here) the pain points that need to be addressed.

As a framework, LeSS is lean and transparent. It does not have any "secret pockets" or "special compartments" where organizational problems can find a haven. No dysfunctions escape the sharp focus of LeSS: ineffectively applied processes or tools, ill-defined roles and responsibilities, unhealthy elements of organizational culture and other outdated norms are vividly exposed when using LeSS. Interestingly, while LeSS is a scaling framework that allows us to scale up (roll up) efforts made by multiple Scrum teams, it first requires *organizational descaling*. The phrase I often use here is this: "You can get more with LeSS." To put it another way, to build up Scrum effectively, an organization must remove whatever extra/unnecessary *muda* (waste) it has already accumulated that gets in the way of scaling Scrum. It is almost like this: LeSS prefers a thin but very strong foundational layer over a thick and convoluted but unstable foundational layer; the latter is usually characteristic of an orthodox, archaic organizational design.

Another metaphor that I use to describe LeSS is that it is an *organizational design mirror*. By adopting LeSS, an organization sees its own reflection and decides on necessary changes, depending on its strategic goals and appetite for change. Similarly, to a person who takes personal fitness training seriously and uses a mirror for "course correction," an organization may use LeSS to decide whether any further reshaping or "trimming" is required to get to the next maturity level.

LeSS is also a great guide to technical excellence. I have used LeSS teachings extensively to coach the importance of continuous integration, continuous delivery, clean code, unit testing, architecture and design, and test automation as well as some other techniques that make Agile development so great.
LeSS stresses that mature engineering practices are paramount for effective adoption of Agile across multiple organizational domains, not just IT.

CLDs

So, how can an organization take advantage of both—the simplicity of the LeSS construct, on one hand, and its deep systemic views, on the other hand—to improve its organizational agility beyond a single team? How can the principles of Lean and systems thinking (together and along with understanding "beyond-first-order" system dynamics) be leveraged to implement true Scrum without reducing, minimizing, or downplaying the importance of its core values and principles?

As an organizational and Agile coach and someone who has been using LeSS extensively in his daily coaching work, I frequently witness situations when companies must deal with this serious dilemma. Here I want to share the magic "glue" that helps me bring my thoughts together and deliver them to my clients. This "glue" is one of the most effective tools that I have discovered for myself inside the LeSS toolbox. It is called **Causal Loop Diagrams (CLD)**.

CLDs provide a great way to graphically illustrate cause-and-effect relationships between various elements of an organizational ecosystem. CLDs help me effectively uncover second- and third-order system dynamics that may not be as apparent to the naked eye as first-order dynamics. CLDs help me brainstorm complex organizational puzzles and conduct deep analysis of system challenges.

Ultimately, I have found that CLDs are highly useful in communicating ideas to my customers, particularly to senior leadership.

Here are some elements of CLDs that I use in my graphics:

- Goals —A high, overarching/strategic goal that needs to be achieved.
- Variables—System elements that have an effect or influence on other system elements (other variables).
- Causal links—Arrows that connect two related variables.
- Opposite effects—"O" annotation near an arrow suggests that the effect of one variable on another is the *opposite* of what could be expected.
- Delayed effect—"||" annotation that disrupts a causal link (arrow); it implies that there is a delayed effect of one variable by another variable.
- Extreme effects—One variable has an extreme (beyond normal) effect on another variable; it is represented by a thick arrow.
- Constraints—"C" annotation near arrow; implies that there is a constraint on a variable.
- Quick-fix reactions—"QF" annotation near an arrow; action that brings about short-term, lower-cost effect.

At this point, I would like to provide an example of using CLDs to visually illustrate second- and third-order dynamics between key system variables that I often see causing harm and unrest to organizations: *performance-driven, discretionary monetary incentives*.

I would like to follow through the process of interaction between system variables as they come to play with one another and uncover the impact they have on the overall system.

Every year, a company (hypothetical Company X) must distribute a large sum of money to many of its employees in the form of discretionary bonuses. To make the decision-making process less subjective, a company ties it to the employees' individual performances via reviews and appraisals. People who have demonstrated better performance get more money; people who have demonstrated poorer performance get less (or nothing). This requires that every employee be evaluated by her line manager, usually twice every year, at which time the employee gets some rough idea about "how much she is worth as resource." This serves as a guide to how much discretionary money the employee might expect to get as a bonus. While on its surface the process of performance evaluations and appraisals may seem to be more objective than a line manager simply deciding on his own, it is still very subjective in that the employee's opinion is disregarded when making decisions. Furthermore, the process is harmful and causes deterioration of individuals' morale and relationships, on multiple fronts. The undesirable effects and short-/long-term damage of performance evaluations and appraisals have been studied for years; lots of research and statistical data is available today. If you are not familiar with this topic or would like additional background information to deepen your understanding, you may refer to the following resources, prior to proceeding with reading:

- Linking Monetary Incentives to Appraisals, by Gene Gendel (white paper)
- *Get Rid of the Performance Review!: How Companies Can Stop Intimidating, Start Managing—and Focus on What Really Matters*, by Samuel A. Culbert
- *Abolishing Performance Appraisals: Why They Backfire and What to Do Instead*, by Tom Coens and Mary Jenkins
- *Punished by Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes*, by Alfie Kohn
- *Hard Facts, Dangerous Half-Truths and Total Nonsense: Profiting from Evidence-Based Management*, by Jeffrey Pfeffer and Robert I. Sutton
- *Out of the Crisis* (MIT Press), by W. Edwards Deming

Moving along with this discussion, I would like to highlight the following three downstream "system variables" that are directly (first-order dynamics) impacted by individual performance reviews. This type of system variables integration is mainly observed among technology groups. Once we understand first-order dynamics, we will proceed to some other downstream ("beyond first order") variables.

Employee Happiness Factor

Many research studies have proved that employees don't like to be appraised. An appraisal is equivalent to slapping a price tag on someone and is hardly an objective process, as the only opinion that really matters is that of a line manager. Yet the official story at almost any company is that an appraisal helps an employee grow and mature professionally and offers a way to improve her individual performance toward some

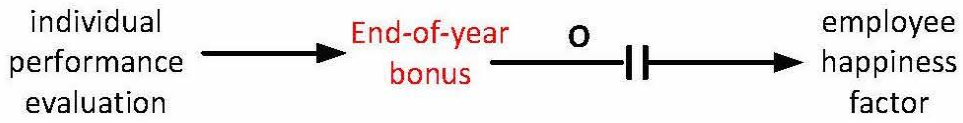
arbitrarily set target. Truth be told: Were the intent of appraisals to help employees grow and continuously improve, the process would not be implemented once or twice a year but rather more frequently in ways that would allow an employee to make a necessary course correction more iteratively. After all, why wait for six months to tell a worker that she needs to improve?

At the time of appraisal, a manager delivers to an employee her final and practically undisputable decision. An employee has practically no effective way to challenge or dispute such decisions. Frequently, even a line manager does not have control of the process (although this is rarely admitted): He or she is presented with a fixed "bag of cash" coming from management above, and this bag somehow must be distributed among lower-ranking workers. And just to be fair to line managers who are not delusional about the dysfunction they must entertain, most of them also dislike the process because it makes them resented by the employees they manage.

So, as time goes by, employees become less and less pleased with evaluations and appraisals. The impact may not be observed immediately, since it usually takes time for an employee to mentally mature to the point where she begins to comprehend the unfairness of the process.

(Of course, exceptions exist among people who have a longer experience of dealing with this process and understand its ineffectiveness and harm.)

While leveraging CLDs in my discussions with senior management, I use the following graphic representation and annotation to convey the concept:



This graphic suggests that annual appraisals have a delayed and even opposite effect on employees' happiness.

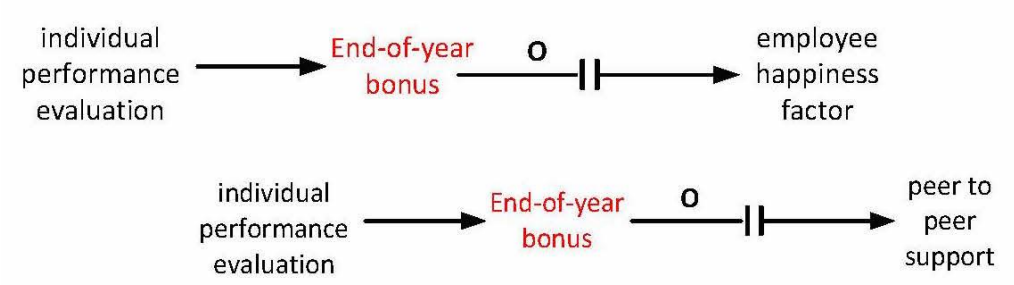
Peer-to-peer support

Peer-to-peer support, willingness to share knowledge with colleagues, collective ownership of assignments, and shared responsibility for deliverables—these are the hallmarks not only of feature teams' dynamics but also of any Agile environment. For employees to be mutually supportive, they must operate in a noncompetitive environment where they don't view each other as competitors or rivals. This is practically impossible to achieve when every employee perceives another employee (at least within the salary

tier) as a competing bonus collector. And this is exactly what is observed in environments where bonuses are distributed based on individual performance: Employees compete for the same limited pool of cash. But everyone cannot be a winner. Even in a group of the brightest individuals working together, someone within that group would have to be ranked higher and someone else lower (and note that this is frequently explained to people up front). How could we expect people to be supportive of each other if, effectively, underperformance of one employee and her inability to collect extra money increases the chances of another employee to bring home more cash? Performance appraisals and discretionary moneys drive employees apart, not together.

Again, the adverse results of appraisals may not be immediate: pain points become more obvious after bonuses are actually paid (end of year/early next year)—this is when employees start developing resentment and jealousy toward each other over paid bonuses.

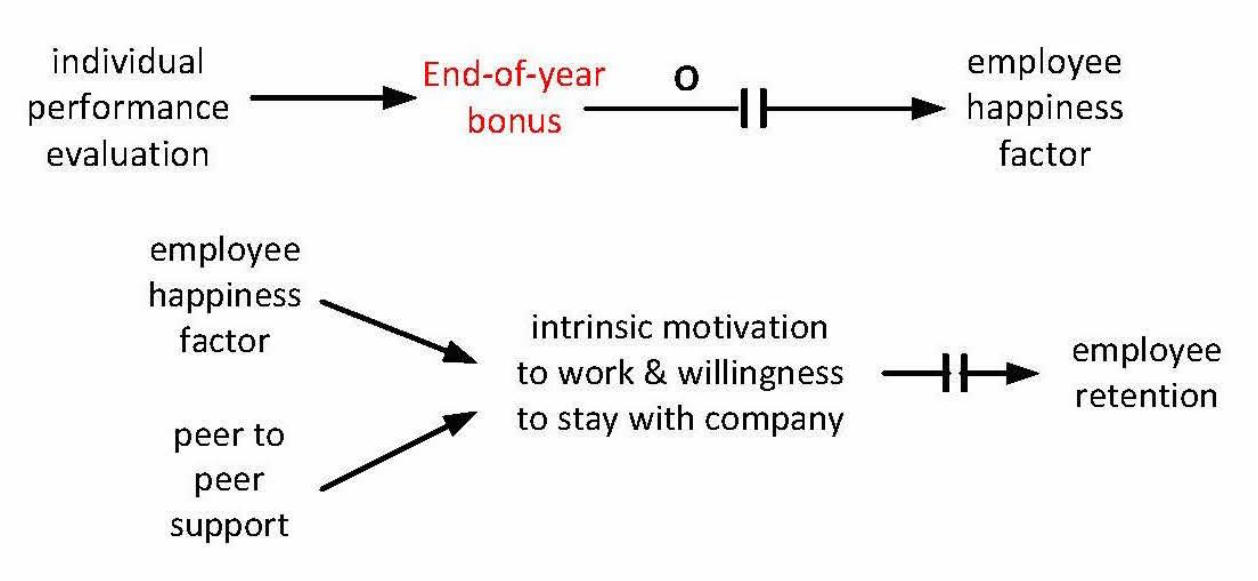
While leveraging CLDs in my discussions with senior management, I use the following graphic representation and annotation to convey the concept:



This graphic suggests that annual appraisals have a delayed and opposite effect on peer-to-peer support.

Both variables above directly define employees' ***intrinsic motivation*** to work and their willingness to stay with a company. After all, can we expect that an unhappy employee, while being in constant competition with his peers and being deprived of an opportunity to safely experiment, would want to dedicate himself to that company for a long time? Probably not. As a result, ***employee retention*** should not be expected to be high. As has often been seen, *good employees always leave first*.

While leveraging CLDs in my discussions with senior management, I use the following graphic representation and annotation to convey the concept:



This graphic suggests that both employees' happiness and their willingness to support each other are directly related to their intrinsic motivation to work and willingness to stay with a company; and, as a downstream effect, this increases employee retention. The opposite would be true as well: Lowering values of upstream (left-side) variables will lower values of downstream (right-side) variables.

"Environmental safety" and the desire to experiment

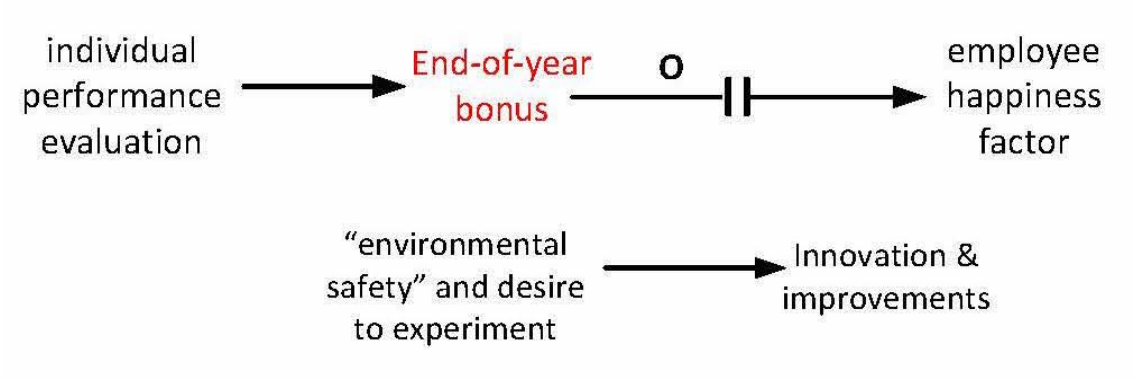
Innovation and experimentation are paramount for success in software development. This is what drives feature teams toward improvement. Scrum, for example, requires continuous inspection and adoption. It is expected that, while experimenting, feature/Scrum teams may run into roadblocks or have short-term failures, at which point they will learn and improve. But to be willing to experiment and take chances, teams need to be sure that they are safe to do so. In other words, they need to be sure that they will not be judged and scrutinized for their interim failures. Such "environmental safety" will be always jeopardized by individual performance appraisals. Why? Because individual success (high individual performance) of an employee is defined by her ability to precisely meet individual goals, set in stone early on during the year. The need to follow a "script" precisely kills any desire of an employee to experiment.

After all, why would a person want to take any chances if her failures will be perceived by line management as underperformance?

Since appraisals make working environments unsafe and kill individuals' desire to experiment, as soon as an employee is presented with her annual goals, she reacts self-protectively by starting to "work to the script" and trying to document every personal

achievement "for the record" (a.k.a. "CYA").

While leveraging CLDs in my discussions with senior management, I use the following graphic representation and annotation to convey the concept:



This graphic suggests that when employees are safe and are not afraid to experiment, innovation and experiments take place in a workplace. Inversely, lack of safety in a work place and absence of desire for experimentation reduces chances of innovation and improvement.

In the sections below, I would like to take a closer look at system dynamics that are beyond the first order of interaction by tracing some additional downstream system variables.

Team synergy and stability

In Scrum, we would like our teams to be stable and long-lived. We would like to see team members enjoy being a part of the same team and doing so as happy volunteers, not as prisoners constantly looking for opportunities to escape. In fact, the best feature teams known have been created because of voluntary self-organization, not because of a managerial mandate.

Why do we want our Scrum/feature teams to remain stable? Here are some good reasons—they give rise to the following:

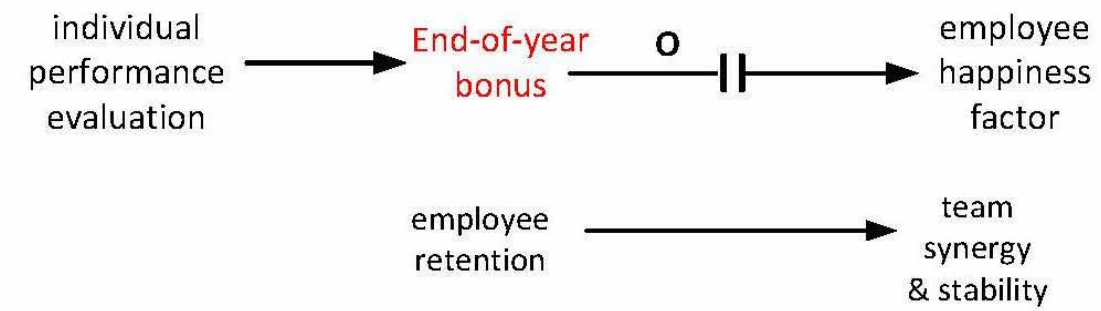
- A collaborative environment and a desire to work together.
- Shared domain expertise and cross-pollination with technical knowledge.
- Predictable team velocity and the ability to plan/forecast more accurately.

So how do performance evaluations and appraisals impact team synergy and stability? Here is how this happens, indirectly:

Via low employee retention. As employees leave a company, feature teams disintegrate. This brings together new team members that have never worked together and require time before they can "form, norm, and storm." As feature teams get disassembled and reassembled, velocities drop/become less reliable and system variability increases (estimation becomes less accurate). The effect is usually immediate.

In my personal experience, I have seen many feature teams breaking loose and falling apart shortly after companies have announced annual bonuses.

While leveraging CLDs in my discussions with senior management, I use the following graphic representation to convey the concept:

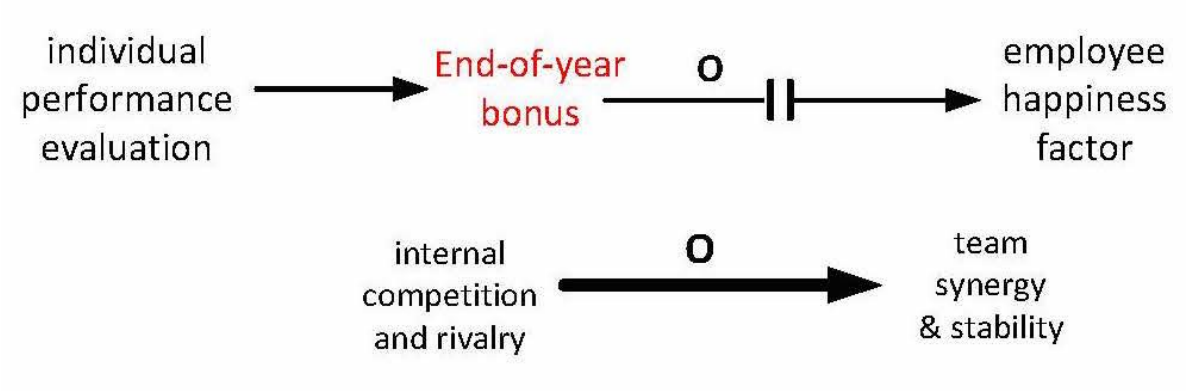


This graphic suggests that high employee retention will lead to elevated team synergy and stability. Inversely, low employee retention in a workplace lowers teams' synergy and stability.

Via high internal competition and rivalry. Once employees realize that they must compete with their own teammates for discretionary dollars, collaboration deteriorates dramatically. Individuals stop supporting each other in pursuit of common goals. Instead, everyone strives to be a superhero and solitary performer, trying to demonstrate his or her own efficiency and hyperproductivity to a manager. Everyone wants to look better than other peers and teammates. The race to demonstrate the best individual performance has a high cost: It happens at the expense of overall team performance.

Since collaboration, swarming, and shared ownership of work are critical for healthy Scrum, the obvious downstream effect of performance evaluations and appraisals becomes clearer: lowered team synergy and instability.

While leveraging CLDs in my discussions with senior management, I use the following graphic representation and annotation to convey the concept:



This graphic suggests that internal competition and rivalry will have an ***extreme*** and ***opposite*** effect on team synergy and stability.

Healthy Scrum dynamics

There are many known system variables that interact with one another and define effectiveness of *basic* Scrum. If most readers of this article are familiar with Scrum, and to keep my focus on other important downstream system variables, I am going to leave out detailed discussions of basic Scrum dynamics. It will suffice to mention that the following classic Scrum-specific variables always have to be considered: feature velocity, number of defects, dollar rate at which developers are hired (low vs. common), number of low-skill developers, cash supply, ability to guide and improve the system, etc. If the reader is interested in exploring this in depth, the "Seeing System Dynamics: Causal Loop Diagrams" section of http://less.works describes these system dynamics with the use of CLDs.

However, when leveraging CLDs in my discussions with senior management, I still use the following generalizing graphic representation and annotation to convey this commonsense, overarching concept:

This graphic suggests that team synergy and stability lead to healthy Scrum dynamics and a feedback loop that is positive (value increase on the left leads to value increase on the right). In my experience, the effect is sometimes delayed. A time lag is usually due to previously gained momentum.

So far, we have used CLDs to explore system dynamics that primarily impact technology teams. At this point, I would like to shift my focus to the business side of the house and explore the part of system dynamics that involves customers.

I would like to provide some examples of how CLDs can expose the adverse impact of individual performance appraisals and discretionary monetary incentives on product

ownership in Scrum.

Identification of a GREAT product owner

Finding a good candidate for the role of product owner is one of the most challenging tasks in Scrum. Why?

The role of product owner combines certain characteristics that are not easily found within the same individual, and it is organizations of high organizational complexity and Taylorian culture where this challenge is seen most. On one hand, the product owner is expected to have enough seniority and empowerment to make key strategic business decisions. On the other hand, the product owner is expected to get intimately involved in day-to-day, and sometimes hour-by-hour, interaction with technology groups. When these two sets of characteristics come together in the same person, we hit a jackpot: We get a great product owner—a person who is both *empowered* and *engaged*. But truth be told, it is often challenging to identify a person who possesses both traits. In most Orange organizations (the predominant color of most modern corporations, as per the Laloux Culture Model), the definition of every job includes a fixed set of responsibilities that individuals are obligated to fulfill. If we look at most job descriptions, as they are defined by HR departments of Orange companies, we will hardly ever see a job spec that has "slack time" for a person to take on the responsibilities of a product owner in addition to his primary job, let alone a job spec that is fully dedicated to the role of PO. For most organizations, the product owner role is still not well defined, and as such, it is not perceived by employees as a step toward career advancement. Today, many organizations that use Scrum have to experiment with the role of the PO by looking for the right individuals internally. Individuals who step up for the role of product owner must make a conscious decision with full acknowledgment that they will be taking on a very wide spectrum of new responsibilities. For most people, this is risky because it effectively means that attention and focus on primary activities (as per job specs) will be diluted by secondary activities—fulfilling the role of PO.

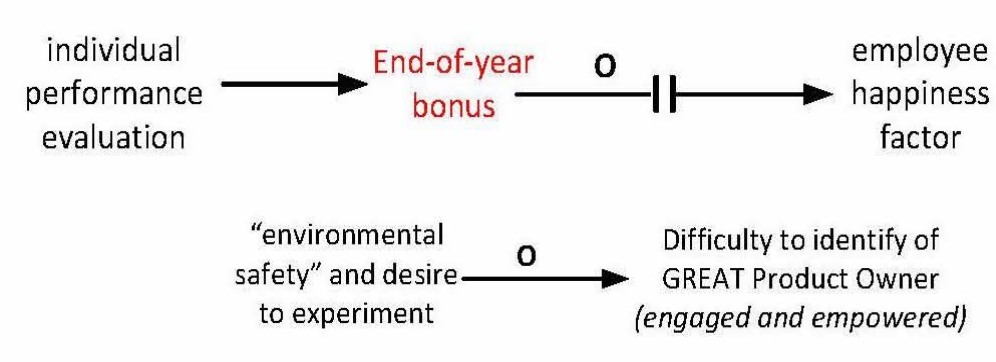
Of course, this problem could be easily mitigated with full backing and support from senior leadership and HR by redefining job specs and explicitly recognizing the criticality of the product owner role. But it hardly ever happens (mostly at product development companies, if there).

It is hard to argue that people must be recognized for the work that they do. I doubt that anyone would object to the following statement: *Nobody should be working two jobs for the same paycheck.* People must "feel safe" about stepping into a new territory, learning new activities, and developing work dynamics that they have not experienced before.

This brings us to the same concept that we discussed earlier when we looked at technology groups:

Individuals need to feel safe to be willing to experiment with a new role. It would be unreasonable to expect an employee to take on more work that would not be "counted in" when a person gets evaluated for his contribution to an organization.

So again, while leveraging CLDs in my discussions with senior management, I use the following graphic representation and annotation to convey the concept:



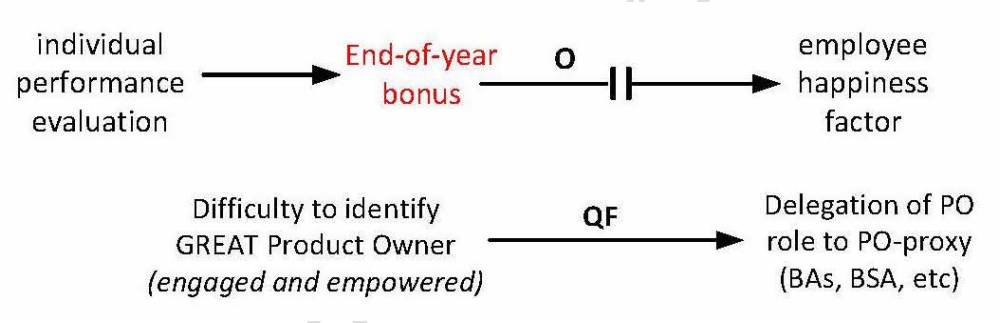
As the graphic suggests, when employees are safe and are not afraid to experiment, it will be *less* difficult to identify a good product owner. Inversely, the opposite is true as well: Lack of safety and inability to experiment makes the process of PO selection much more challenging.

At this point, it is worth mentioning one very common quick fix that organizations frequently make to compensate for shortcomings in finding a good product owner.

Empowerment usually implies that a person occupies a senior organizational position. As such, a business person's career has progressed beyond a certain point; she no longer has enough bandwidth (nor desire!) to deal directly with technology. On reaching a certain level of seniority, a person has bigger fish to fry, and collaborating with individual technology (feature) teams is no longer her priority. So, while still retaining one of the "E's" (empowered), a person is not able to demonstrate another "E" (engaged). To compensate for the missing "E," another person needs to be inserted into the system to fill in the gap between a real product owner and the technology teams. This poorly defined (even undefined) role is sometimes labeled "PO proxy"—a surrogate person tries to act as the PO but does not have the power. This role is usually occupied by someone from a lower organizational layer: a business analyst, system analyst, or another person

more accustomed to working directly with technology and for whom the activity itself is not perceived as "below pay grid." This creates a serious dysfunction in the Scrum operating model as communication between a true customer (empowered PO) and technology is now hindered: The surrogate role of PO proxy usually lacks strategic/holistic product vision and any power to make important business decisions within short time frames, as it is required by Scrum. It is worth noting that the functional expertise of a business analyst or systems analyst are both welcomed in Scrum and usually reside within teams (although single-specialty individuals are viewed as less valuable than multiskilled, a.k.a. T-shaped individuals). The reason the delegation of responsibilities described above is problematic is because it artificially creates unnecessary communication layers between end customers and technology. This type of organizational design causes a variety of additional dysfunctions (miscommunication, hindrance to information flow, confusion of priorities, etc.), and therefore is strongly discouraged.

While leveraging CLDs in my discussions with senior management, I use the following graphic representation and annotation to convey the concept:



As the graphic suggests, the difficulty in identifying a good candidate for the role of PO creates the need to look for quicker and cheaper solutions; introducing a powerless surrogate role of the PO proxy is a commonly seen but undesirable Plan B. The reason this fix is "quick" is because it usually does not take too long for a real PO to realize that he/she is not able (or willing) to handle the additional responsibilities of the role.

In my practice, the risk of losing a real product owner and getting a proxy instead did not take too long to materialize: usually a few weeks after Scrum was introduced.

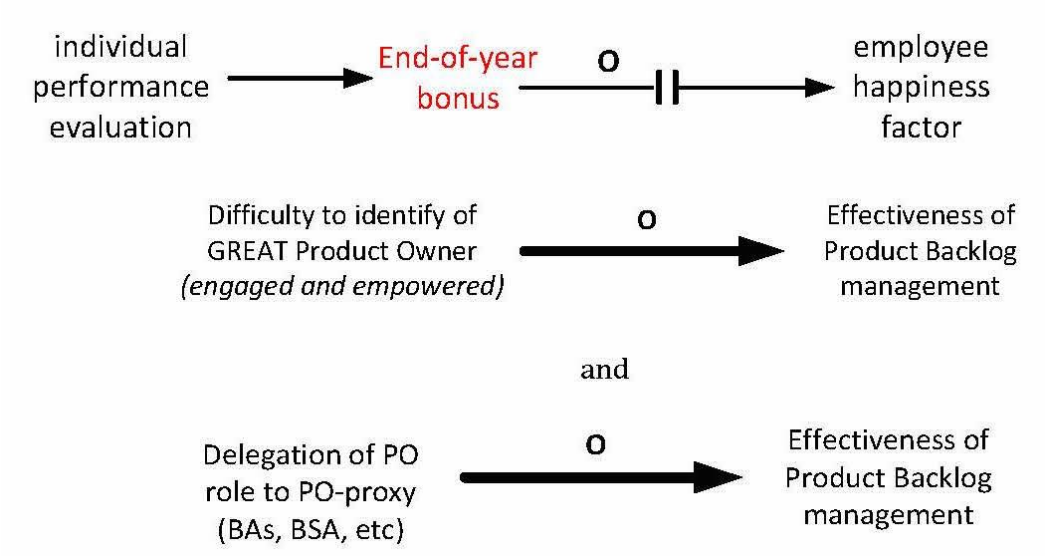
Effectiveness of product backlog management

Effective product backlog management is paramount in Agile product development. This fundamental concept has been introduced in simple Scrum, and it remains as valid in scaled Scrum.

In fact, when an organization scales its Scrum, by involving multiple technology teams, POs, and lines of business, effective product backlog management becomes even more critical: Work coordination, resource management, impediment removal, alignment of business priorities, etc., are all part of the package.

As we can guess, effective product backlog management, including work prioritization, story decomposition, etc., can be done most effectively with participation of a real product owner. And conversely, if an organization is missing the critical figure of the product owner, product backlog management will become ineffective.

While leveraging CLDs in my discussions with senior management, I use the following two graphic representations and annotations to convey these two related concepts:



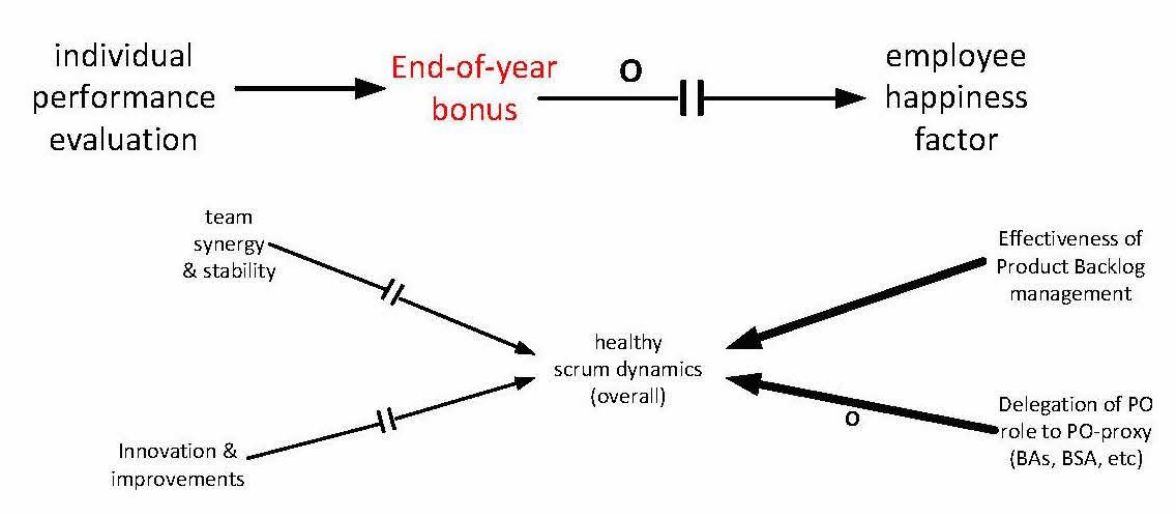
As the graphic suggests, product backlog management suffers from both: lack of true product ownership and presence of ineffective surrogate roles. In my personal experience, the effect is usually extreme.

Healthy Scrum dynamics (overall)

At this point, I usually provide senior managers with a partial summary of LCD by showing how "healthy Scrum dynamics," while sitting much further downstream from individual performance evaluations, appraisals, and bonuses, are still impacted by the latter group via second-order dynamics (through secondary variables). CLDs do a great

job of bringing many aspects of system thinking together and presenting them visually.

Below is the combined view of how four upstream system variables that we have discussed earlier relate to "healthy Scrum dynamics":



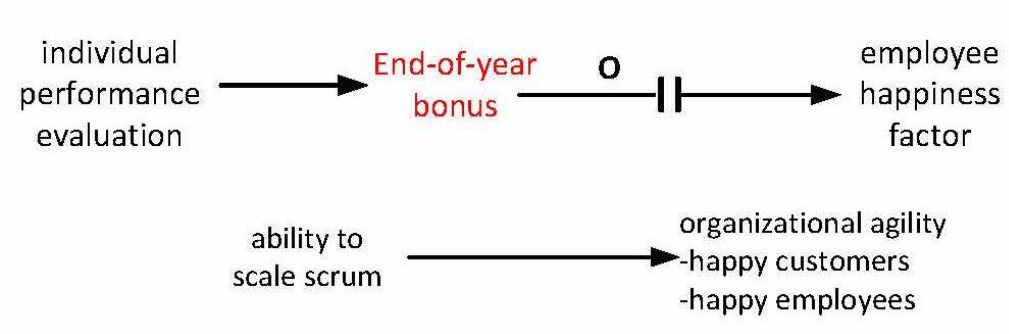
As the graphic suggests, the nature of the effect (positive vs. negative), time of onset (immediate vs. delayed), and impact (casual vs. extreme) could be unique for each variable.

Scaling Scrum and organizational agility

In this section, I want to describe how I, with the use of CLDs, bring my discussions with senior management to culmination by painting a bigger picture of organizational agility.

For most large organizations, success by a single team is not the end goal. Organizations look for "bigger" solutions. And their reasons are obvious: huge IT departments, many lines of business, many customers, multiple competing priorities, multiyear strategies, and many other elements that make organizational needs nothing less than huge. Luckily, most organizational leaders that I have met in my practice understand that the ability to effectively scale basic Agile frameworks (e.g., simple Scrum) will ultimately improve organizational agility and ensure that both customers and employees are happy.

Below is the graphic that summarizes this last, "commonsense" relationship:



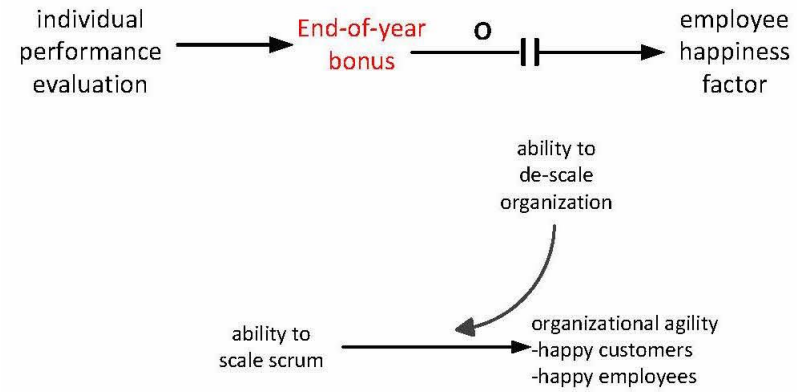
Tying it all together

What I would like to do at this point is to take one step back and describe what it takes to scale Scrum effectively:

This is where another powerful concept of LeSS comes to the rescue: In order to scale Scrum, an organization must be descaled first (please refer to "Less Agile or LeSS Agile?" by Craig Larman). In other words, to construct a model of Scrum, performed by multiple teams, an organization must remove (deconstruct) its existing organizational complexity first. As it was stated at the beginning of this article, scaling does not imply making things more complex, but unfortunately this key concept is not always well understood. Mistakenly, many people still think that to support existing organizational complexity they need to look for multitiered, complex Agile frameworks that will provide "room and purpose" for every existing organizational element: roles, processes, tools, and techniques.

The analogy that I frequently use to deliver the concept of scaling to senior management is that building a sky scraper on a wobbly, porous foundation is dangerous because it will eventually crumble. The surface must be cleaned up first, flattened and hardened, and only then will there be a chance to build something tall and strong.

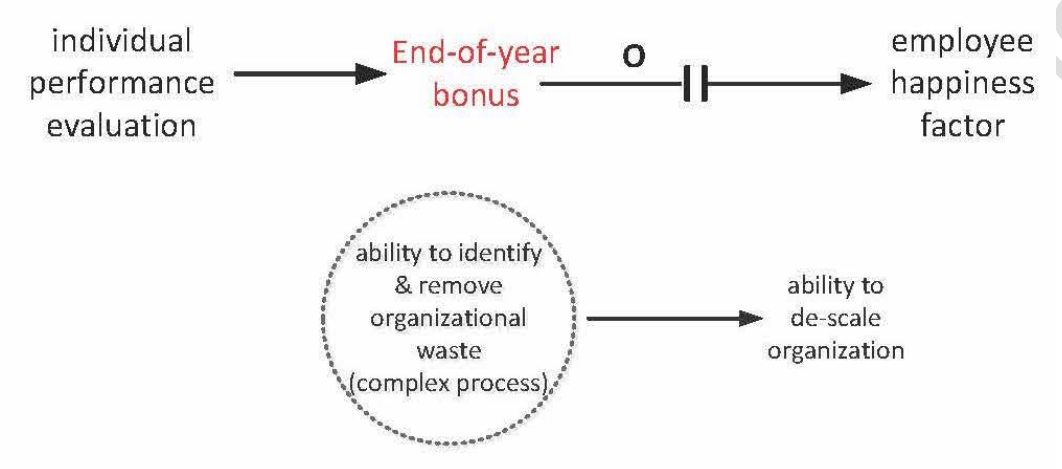
Below is the graphic that summarizes this concept:



At this point, the most common request I get from senior leaders is to elaborate on what I

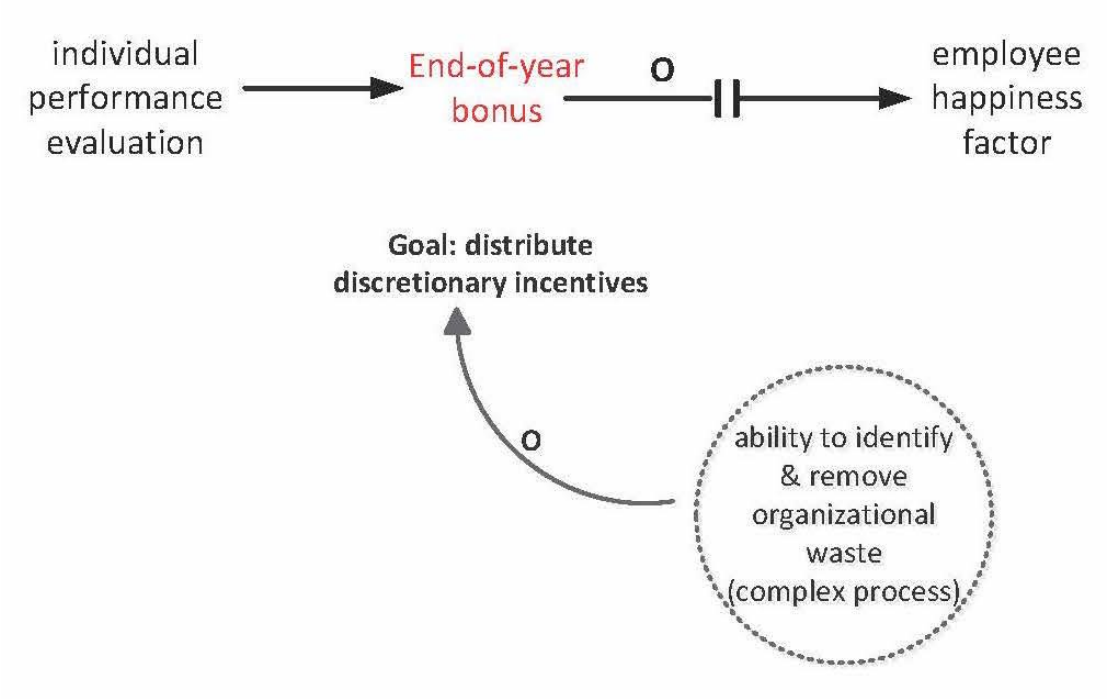
mean by *descaling*—and this is my favorite topic. This question is natural, but I usually resist answering it immediately since the topic is inherently large, complex, and, at times, inflammatory. Therefore, I request a dedicated discussion for it.

I still produce a CLD graphic illustration of the concept, as shown below, but offer a follow-up discussion to explore details:



Ultimately, when this discussion is held, I always tie it back to the present discussion and explain why ***Goal:*** **Distribute discretionary incentives** becomes so trivial with the identification and removal of system/organizational waste. This discussion is usually long, and it requires challenging many outdated organizational norms and principles that some senior leaders are not willing to give up easily.

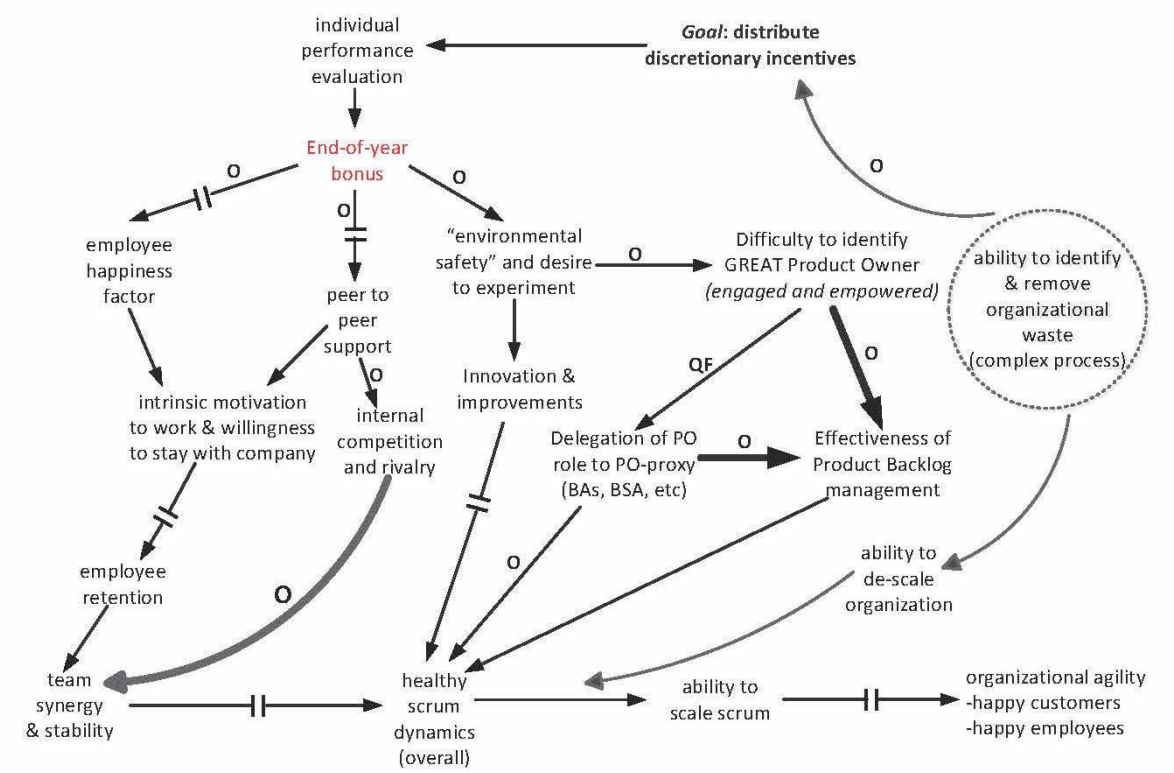
The CLD graphic illustration is a high-level generalization of the concept of the opposite (inverse) relationship between the two system variables:



As mentioned above, the variable in the dotted circle can be decomposed further into many smaller system variables that have upstream and downstream relationships with one another.

Summary

The best summaries are short. Therefore, I would like to summarize this post briefly with one comprehensive CLD diagram that brings together the variables, relationships, and annotations that have been discussed:



Although it may take hours, or sometimes days, of brainstorming to produce a CLD, when complete it becomes a great communication vehicle. A diagram like this one can be created in real time, in collaboration with others, on a whiteboard. Alternatively, it could be created ahead of time by a coach or trainer and then be used as a "cheat sheet" when appropriate. CLDs can also be shared with a wide audience ahead of time to solicit questions and provoke interesting discussions at a later point.

Coach's Experience Report: Putting LeSS Teachings to Work

Originally published on February 22, 2016 | Location: http://www.keystepstosuccess.com/2016/02/coachs-experience-report-putting-less-teachings-to-work/

The following Coach's Experience Report describes various teachings of Large Scale Scrum (LeSS) framework in the context of their practical use by the Agile coach. What is below does not represent a single case with a single organization or company. Rather, experiences with multiple organizations under different conditions are being described. By the same reasoning, not for every organization, whose experience is being drawn upon in this report, *all* LeSS teachings that are described below have been experimented.

Coach's Discovery: Scrum teams have been struggling to gain autonomy and independence due to close monitoring and constant involvement of line management. Teams' decisions made during sprint planning were continuously overruled by management. Mandatory requests coming from management were frequently in conflict with priorities coming from product owners. Teams were unable to conduct sprint retrospectives privately and safely, with management insisting on its presence in ceremonies and/or reviewing retrospectives' outcome.

LeSS Teaching by Coach: Taylorian carrots and sticks used to be effective during the American Industrial Revolution when they were applied to people who performed mundane, unskilled, manual labor. But in modern work settings of the twenty-first century (for the most part) don't work if applied to intellectual workers. Command and Control behaviors suppress individuals' willingness to explore and innovate, discover and experiment; they demotivate and demoralize workers and therefore lower productivity. Senior management needs to order fist-level line management to step back and allow teams norm and gain autonomy and independence. Close oversight and supervision will not allow teams to fully explore their potentials and achieve higher productivity.

Overall Result: Positive. Many teams have been liberated from Management Type X and have been treated with Management Type Y, instead.

Coach's Discovery: Team members were expected to work closely together, share knowledge, help each other grow complementary expertise. Teams were also asked to deliver together, "as a whole," at the end of each sprint and demonstrate shared ownership and swarming during sprints. Team members were expected to take turns in a driver's seat during showcases to equally gain visibility in the eyes of product owners and customers.

But at the same time, each team member was being stack-ranked, during an individual performance appraisal, against his/her own team members as well as against members of other, neighboring teams. Each ranked individual understood that he/she competed with other teammates for discretionary money that would have to come in the form of a bonus

at the end of year. As it came closer to midyear reviews and end-year reviews, teams' dynamics worsened and bad behaviors were observed, practically, inside every team: less collaboration, emphasis on private ownership and individual deliverables, selfishness, blame-gaming and finger-pointing. As a result, teams' velocities dropped, quality went down, and customer satisfaction was lowered.

LeSS Teaching by Coach: "The idea of a *merit rating is alluring (as per S. Deming)."* Individual performance appraisals linked to monetary incentives lead to demotivation, loss of enthusiasm and bad behaviors such as internal competition, rivalry, selfishness and organizational degradation. Having individual performance appraisals linked to distribution of discretionary monetary incentives, such as bonuses and salary increases, worsen a situation even further.

Overall Result: Mostly Negative. Line management did not accept the fact that merit rating and individual appraisals had such harmful downstream effects on teams' dynamics and cause organizational degradation. Senior management seemed to understand that the problem existed and was serious but was still too hesitant to rock the boat because many fundamental organizational norms and policies, many of which set by HR, would be challenged. In rare situations, however, management could emphasize team performance and collective results as main attributes of individual performance/results.

Coach's Discovery: Recently trained teams fell under close surveillance and scrutiny by line management. Line management viewed Agile/Scrum, as a magic wand that would miraculously resolve all their existing problems. Management started paying too much attention to metrics (e.g., velocity) and set unreasonable expectations for teams' productivity during initial sprints. When teams initially failed, management blamed Agile/Scrum for failures instead of treating it as a "mirror" that just painfully reflected existing broken processes.

LeSS Teaching by Coach: In Scrum, when a team just gets trained and is set to sail, private sprints with "fake" product owners (if a real one is identified yet) are recommended. Why? A team may want to practice/dry run Scrum dynamics (roles, artifacts, ceremonies, feedback loops) but may not necessarily want this information to be publicly disseminated across an organization to avoid premature judgments and mis-measurements of success. A team is not obligated to announce to the rest of the world that they are experimenting new ways of working UNTIL everyone who is involved is ready and comfortable.

Overall Result: Positive**.** Teams no longer viewed the last day of Scrum training as a commitment point at which time they had to announce to the rest of the organization that "they were Agile now." Teams became more comfortable transitioning into new dynamics, and they did it gradually while "playing it safe" before publicizing their intentions or results. In cases, when a real product owner was not immediately available, teams used another surrogate to play this role (e.g., Senior BA or SME).

Coach's Discovery: A team was experiencing a lot of distraction coming from stakeholders and customers. Instead of going to the product owner with requests, customers went directly to the team. Frequently, competing priorities arose: A solution that addressed one request conflicted with a solution that addressed another request. The product owner took advantage of his overly proactive clients, stepped back and did not do his job.

LeSS Teaching by Coach: When it comes to feature (Scrum) team communication, there are three main types that exist:

- Requests: from customers/users and the product owner.
- Prioritization: from product owner to the team.
- Clarification: between customers/users and the team (also can come from PO).
-

Effectively, this allows for business requests to flow from various areas/departments of an organization to the product owner but then to be *prioritized* and fed to a team/backlog by the product owner himself in a controlled fashion. While a team is shielded from customers'/users' ad hoc requests, which are sometimes competing, it still has a right to go to customers/Uusers for clarifications.

Overall Result: Positive. The Team learned how to say NO to customers and defer their requests to stakeholders. The product owner was "forced" to step up to the plate and practice one of his key responsibilities—being the voice of a customer, facing a team.

Coach's Discovery: An organization had wide geographic distribution, with technology resources present in India, Eastern Europe and South America. The long-standing goal of outsourcing was to find the cheapest resources for a single specialty. For example, front-end Java developers were all sourced from India, Flash and UI experts from Eastern Europe and architects from Argentina. This caused a lot broken communication and unnecessary coordination between feature team members: language barriers, geo- time-zone distribution, etc. Also, end-customers were from the US, and this further added to complexity. Inefficiency of highly distributed teams trying to coordinate ceremonies and optimize time overlap was painfully noticeable.

LeSS Teaching: Given today's global marketplace, geographical distribution of skilled workers is practically inevitable for most companies. However, when it comes to teaming, it is critical to avoid geographical distribution within a single team. Companies should support collocation of members of the same cross-functional team

Note: Even with the latter approach, doing this with componentized teams presents other problems). Also, bringing business (stakeholders, SMEs, product owners) closer to teams is highly desirable.

Overall Result: Partially Positive. The leadership agreed to reconsider the current geographic collocation strategy. Having a group of single specialty experts located in one place, communicating across many time zones with another group of single specialty experts became a less preferred option. The leadership started to see more value in collocating individuals based on their needs to work together on same features. At first, it became more expensive to procure certain expertise where it was not as abundant and its cost was higher (e.g., Flash developer in India), but over time, increased efficiency and higher rate of business value output by each team made changes worthwhile.

Coach's Discovery: During initial stages of Agile transformation, senior leadership came to the realization that by restructuring their organization to improve overall organizational performance, some internal "waste management" activities had to be done. Specifically, it became clear that certain processes, artifacts and roles were redundant, unnecessary and costly. As such, they had to be reduced or removed from the system altogether. This raised a concern for senior management: removing certain elements of organizational structure could become too politically inflaming. For example, an excessive amount of business analysts and project managers (PMOs) represented two thick organizational layers that were primarily focused on producing heavy documentation and less- than-reliable reporting, respectively. Reducing these two layers would effectively mean downsizing certain individuals—something that could loudly resonate across the rest of the organization.

LeSS Teaching: Organizational leadership needs to understand the difference between *local optimization* (e.g., improving performance of a single organizational layer, functional silo, reporting structure) and *system optimization* (e.g., improving performance of an entire system). Looking at an organization from a standpoint of system optimization, an organization should care to provide *job security* to its employees, not *role security.* Ultimately, the goal of any organization is to continuously strive towards improving its efficiency, not providing a haven for roles that make it (the organization) less efficient.

Further, from a system optimization perspective, it is wiser to support the idea of not having individuals that are all just specialists in a particular field or domain; an organization needs to have a good number of generalists to avoid workflow management dysfunctions and workflow bottlenecks. Presence of T-shaped individuals are highly desirable.

Overall Result: Positive but WIP. While removing organizational waste, senior leadership tried to strike a happy balance between simplifying organizational structure and removing redundant/unnecessary roles on one end while trying to provide job security and alternative career paths for some knowledgeable and highly qualified individuals.

Coach's Discovery: After more than a dozen sprints, a group of feature teams still could not show any progress in their ability to deliver potentially working software at sprint-end. At the end of each sprint, teams still produced code that needed additional testing, test automation activities, integration with code of other teams, extensive UAT, and other "udone" activities. The initially proposed Definition of Done (DoD) at the time when teams got trained did not change much, and before releasing to production, teams still required at least one "hardening" sprint.

LeSS Teaching: As a feature team matures, gradually, it should extend it definition of "done" to bring itself closer to a point where, upon finishing a sprint, it makes its deliverable production-ready.

Overall Result: Partial Success. The teams were encouraged to identify during retrospectives at least one or two elements of DoD that were either missing or needed an improvement. Based on this, the teams could gain some momentum and with every subsequent sprint improved production readiness of their code. However, there were certain organizational impediments that still prevented teams from delivering faster: certain external dependencies on organizational layers that were "outside of Agile sphere of influence"—they prevented DoD to become fully inclusive.

Coach's Discovery: Individuals that have been elected (or appointed) to the role of product owner did not have time to do the job. They were either too senior within an organization to deal with IT directly or had already too much on their plate to take on yet another full time role of product owner. This created a serious gap that made Scrum extremely ineffective and overall agility low. To fill this gap, product owners found other others within their own reporting structure to fulfill this role. They delegated most of PO responsibilities to this new, artificially created role of product owner "proxy." This brought about a lot of dysfunction and hindrance to the process as the proxy did not have the same level of empowerment as a real PO.

In some situations, existing terminology that had a completely different purpose and meaning was overloaded. For example, area product owner (LeSS term) was used to describe a role that did not fit the definition of area PO. Effectively, area PO term was used to describe the role and behavior of a PO proxy.

LeSS Teaching: A business person that represents a single area of a complex product is called an area product owner. Area PO is in close communication with other area POs responsible for other areas of the same product, as well as with the product owner (main person) that oversees an entire product. The area product owner is not to be confused with an ill-defined role of product owner proxy. The latter term is not really defined in Scrum. This term exists in places where a real PO is not able or not willing to do his job (no time, not enough interest/motivation). A PO-proxy, is the PO's surrogate that interfaces with the team(s) to mimic PO (minus authority)—it is an unnecessary organizational layer.

Overall Result: Situational success. In situations where organizational structure (on the business side) was relatively flat, the success of identifying an effective product owner and bringing him closer to teams was much higher. In situations, where business organizational structure was more complex, with multiple reporting layers, the success of identifying a product owner that would be equally knowledgeable, empowered and engaged was lower. Another consistent observation was that every time a product owner came from a middle organizational tier (e.g., operations, a.k.a. not true end-customer) chances were higher that the role of proxy would emerge.

Coach's Discovery: This was a large organization with complex structure, heavy tooling and internal processes that was looking for scaled Agile solutions to accommodate its "internet historic complexity." The organization was looking for Agile frameworks that would seamlessly fit its existing dynamics while not requiring too many changes. The organization was not really trying to improve existing dysfunctions and repair problems. Instead, the organization was trying to look for ways to improve its existing condition by overlaying Agile norms, terms and principles on the top of its current system. This included introducing more Agile roles, ceremonies, processes, organizational layers, handovers and bureaucracy on the top of what existed already. Over time, it worsened the situation even further, not improved it.

LeSS Teaching: In order to improve organizational agility and be able to implement Agile at scale (e.g., have more teams being involved in the same large-scale Scrum, as opposed to having many unrelated teams attempting to do their own scrum), organizational descaling is required first. This includes removing organizational waste, lowering bureaucracy, flattening organizational structure, removing non-value adding roles, reassigning responsibilities to key roles and discontinuing norms and behaviors that have been statistically proved as harmful.

Overall Result: Partial success. The organization understood principles of LeSS, especially Lean Thinking. The organization understood that organization descaling (removing what exists, instead of adding more to it) should come *before* any attempts to scale agility across broader organizational boundaries. However, the organization was still not fully prepared to deal with all the consequences of waste removal. There were concerns with political and legal implications of such bold actions.

To Be Continued:
TBD—more coach's discoveries and respective LeSS teachings that were used to remedy problems:

- **LeSS teaching**: The following elements and attributes lead to "the contract game": componentized organizational structure, heavy/non-negotiable documentation, bureaucracy, functional silos, lack of cross-functional experts (T-shaped people),

merit ratings/performance appraisals/bonuses or other forms of local optimization (e.g., harboring teams of BAs, PMs).

- **LeSS teaching**: Lack of proper understanding of cross-functional, customer-centric feature development leads to creating fake products or projects (e.g., server-side work, back-end coding, database, UI). This further leads to creating fake product portfolios and fake project portfolios. This further creates needs for excessive coordination that mandates unnecessary roles of fake portfolio managers and the like.
- **LeSS teaching**: By analyzing a system's feedback loops, velocity, bugs, number of developers, budget supply, it becomes clear that, for example, the increase in funding (budget) does not necessarily translate into increased velocity or improved product quality. Negative feedback loops are just as important to consider as positive feedback loops: more money may help hire more developers that will produce more bugs.

Scrum and Kanban

Non-IT Kanban Implementation at Scale

Originally publishjed on January 21, 2016 | Location: https://www.scrumalliance.org/community/articles/2014/july/non-it-kanban-implementation-at-scale

A few days ago, I went to the happy hour of a local staffing firm that specializes in placing Agile technical and leading resources. They had moved into their new office and were treating. (I've known these guys for many years and was not looking for a job.) As I was walking around the office, what caught my eye were a few huge whiteboards attached to the walls. One of the boards was in the recruiting area, another one in the account-opening area. I moved closer to see what was on them. Both boards had multiple columns and rows. I focused on the column headers of one of the boards.

Most of the columns looked like sequential steps of one long process: resume submission of a job seeker to a client company. I took a closer look at the other board and, after a few minutes of interpreting its column headers, realized that the board represented the steps in the account-opening process. The boards did not seem to be related and they seemed to be maintained by different teams—one by recruiting, another one by account opening. I looked around for a few more minutes and then asked the regional director to validate my observations and assumptions.

This is what I found out:

Without naming it explicitly, the teams have been using Kanban boards to manage their daily work flows—recruiting did their thing, accounts did theirs. I also learned that each team gets in front of its respective board every morning and discusses things in ways that are similar (not the same) to what Scrum teams do in daily stand-up meetings.

The boards were not connected. The swim lanes did not represent SLAs or levels of escalation. The lanes were not dependent. The swim lanes represented different business tracks that did not even require escalation—they were sort of independent. And so they seemed. But then we discussed it in more depth and realized that since they are consuming the same team resources, there is a relationship. There were no WIP limits on the columns. So it became apparent that the teams were not actively managing the queues. However, they were visualizing them well. Then we discussed further whether the two boards were as independent as they seemed. And this is what we discovered:

Tickets that were moving across the accounts board represented client-companies that went through various states of the account-opening process. But then, on the recruiting board, each client company represented an individual swim lane—a major work track

that was further subdivided into more granular tracks, each representing a requisition number. What moved across the recruiting board were tickets that represented names of individual candidates.

So the boards were not independent: Each ticket that came to the last queue of the accounts board automatically became a major track (could be empty initially) of the recruiting board.

Then, if a requisition number came along, it would become a mini-track.

We were dealing with a non-IT enterprise (mini)-level Kanban implementation that still needed some fine-tuning and polishing up: converging one Kanban board into another, establishing WIP limits on each queue (column) of each Kanban board, introducing buffer queues (e.g., "something"-Ready). The setup also required some additional thinking about whether there were any other intracompany processes that needed to be visualized and aligned with the known ones.

I hinted to the regional manager that by "Kanbanizing" their overall enterprise business, and perhaps adding a few burn-up charts that represented how accounts got opened or candidates got placed over time (assuming a time line of, let's say, 2014), they could really "agilize" the process of placing Agile technologists. They would demonstrate to their clients that they know the business they recruit for, and clients might just like it.

I also left excited, knowing that I have a nice non-IT Agile story to tell—something that would once again support the notion that enterprise-level applicability of certain Agile processes has a place in real life.

Tips to How Run "Big" Retrospectives

Originally published on April 6, 2019 | Location: http://www.keystepstosuccess.com/2019/04/tips-to-run-big-retrospectives/

For any team that uses Scrum framework, a retrospective is a mandatory event that takes place at the end of each sprint. It is an opportunity for a team to reflect on their most recent learning while it is still fresh in everyone's mind. There are many tips, techniques and tools for running a retrospective. They start with very basic guidelines of the Scrum Guide and expand into experiences and experiments of many teams and practitioners.

There are also recommendations on how to run a retrospective in more complex/scaled organizational settings with multiple teams sprinting together (e.g., Overall Retrospective in Large Scale Scrum), as they work on the same product or service and support the same product owner and/or customer journey.

Depending on team(s) maturity, a retrospective could be run with or without assistance of an experienced facilitator (Scrum Master, coach) that possesses *guide-level expertise in Scrum*.

(Notably, a retrospective format is not unique to Scrum. For example, Kanban teams can also retrospect, on-demand, whenever they feel there is a need.)

What about other organizational settings, outside of team dynamics? What about situations, when multiple individuals from different organizational areas need to come together and *retro-actively inspect* (a.k.a. *retrospect*) on their work within and across various organizational areas, or across multiple organizations (e.g., internal departments, between partners-companies, vendors), involving communication, collaboration, reporting, managing each other's expectations?

Below, are some practical tips on how to organize and run a "big retrospective" (e.g., after multiple sprints and/or completing key deliverable with people that are not members of development teams).

1. Most importantly, try having all required parties in the same physical location. For people that are at remote locations, use video conference rooms, and to the extent possible, cluster people together. For example, if a group is distributed between location A and location B, and there is no way to bring everyone together at either location, don't settle for letting "*everyone joining from their desks*" via video phones. At least, maximize clustering of individuals, at each respective location by using conference rooms.
2. For large groups (more than 20 people), try identifying individuals-delegates that represent views and opinions of others. This is done to reduce noise (too many communication nodes and channels) from people involved in discussions. Identifying delegates will also help with the first guideline above: collocating fewer people in the same place is more cost-effective. **Be careful when selecting delegates:** line managers, engagement managers, leads etc., are not the best delegates. Ideally, delegates should be *on-par* with people they represent.
3. Consider bringing an *external* facilitator—someone who does not represent views or interests of any specific group of people or department. A facilitator must be neutral and unbiased—a completely impartial person. Having a facilitator that understands internal organizational dynamics is great but not mandatory. An experienced facilitator will be able to adjust on-the-fly and leverage to his/her advantage, domain knowledge and subject matter expertise of other participants that are involved in a retrospective. Sometimes, one of the organizational units involved in a retrospective may have their own experienced facilitator available. Falsely, such

a person could be perceived by other retrospective participants as someone who is subjective or biased. **Such a preconceived notion may create a problem and must be addressed from the start**.

4. With many people involved and/or joining from remote locations, consider doing some preparatory work that will help running a face-to-face retrospective more efficient. This could be effectively done by a facilitator by collecting ahead of time from all future retrospective participants their preliminary feedback: wishes, concerns, recommendations. All collected information can be then reviewed and analyzed to make it more presentable and actionable at a retrospective: duplicates removed and relevant items grouped together.
5. During a retrospective, a facilitator can present all participants with collected and refined information (4 above), in the form of index cards, and leverage one of the facilitation techniques (e.g., dot-voting or priority vs. impact plotting) to decide on the order of items to be discussed. Additional blank index cards should be available on-hand in case there are last-moment ideas that emerge in a room.
6. Each discussion point should be time-boxed. However, since not all discussion points are of equal priority and complexity, time required to spend on each may not be the same. It is also important to keep a discussion focused/tailored and not let it digress to tangentially relevant (or completely irrelevant) topics. It is a good practice to spend some extra time at the beginning of a retrospective to not only prioritize discussion points but also estimate, roughly, how much time each discussion point may take. This approach of balancing discussion items' priority vs. complexity, essentially, is identical to what a team does to backlog items during a product backlog refinement session.
7. Retrospectives that involve people that don't work on the same team, let alone individuals from different organizational structures and of different levels of seniority, may create a lot of additional tension in a room. The latter, especially, may force more junior people to become very reserved and unconfident in stating their opinions in front of more senior colleagues (some of whom may also be their line managers). Allowing *privates to speak before generals (a.k.a. "military democracy")* could be one of the ways to ensure that junior people are not anchored to views of more senior people and feel comfortable and safe to speak out openly.
8. Similar to a single team retrospective, a *big retrospective* should culminate on a positive note (friendly, mutually supportive vibe) with at least a handful of the most critical items becoming immediately actionable. Since topics that are brought up at big retrospectives are usually more systemic/organizational in nature (as opposed to tactical, team-level), **each actionable should be preferably owned by a more senior person.**

Sprint Length: Who Decides? How? Why?

Originally published on May 15, 2017 | Location: http://www.keystepstosuccess.com/2017/05/sprint-length-who-decides-how-why/

What is the best sprint length? Who decides on sprint length? Are there any exceptions? What are some of the most common mistakes people make when making decisions about sprint length?

Let's start from the grassroots and answer the following basic question: "*What is the sprint main goal?*" And while looking for an answer, let's refer to the Scrum Guide, where the goal of each sprint is clearly described as "…increment (a.k.a. PSPI = potentially shippable product increment) that must be in usable condition and meeting DoD (Definition of Done)." From the Scrum Guide perspective, it is also clear that while being potentially shippable, an increment does not necessarily *have* to be shipped. Why is it so? For PSPI to be shipped, it must also represent MMP (Minimal Marketable Product) and the decision about what is marketable comes only from the product owner and it is based on several factors, including long-term strategy and economics.

Note: End-of-sprint deliverable is sometimes also referred to as **MVP** [Minimum Viable Product] and is described well by Roman Pichler here.

Sprint length and release economics
Speaking of economics, let's recall the relationship between transaction costs (shipping) and holding costs, also described in "The Principles of Product Development Flow" by D. Reinertsen. By analogy, if applied to Scrum product development, "*batch size*" would represent a volume of PSPI and "*cost*" would represent all combined efforts associated with production deployment (e.g., integration, end-user training, marketing announcements, etc.).

How does this relate to sprint length?

While each sprint is supposed to produce PSPI, it is only the product owner's decision when to release it (MMP), and it depends on finding a "sweet spot" between the two types of cost: holding and transaction, or if spoken in software development terms, costs of code deprecation/aging and costs of deployment. On the graph above, it is illustrated by the lowest point of Total Cost curve, and it is the responsibility of the team and product owner to determine what it is.

Corollary to having both strategy and economics changing over time, release frequency may change as well. It would be natural to assume that sprint duration and release frequency are related too. Indeed, the need to release more frequently may lead to sprint shortening, and vice versa.

Are there any other *external* factors that may influence sprint frequency of a single Scrum team? The most classic example would be Scrum by multiple teams that sprint together (e.g., LeSS, S@S): develop the same product, for the same product owner, and share the same backlog. While release economics principles would still apply, the situation with scaling may become more complex if multiple teams are tasked to select a shared cadence. One assumption comes to mind immediately: If multiple teams sprint together (synchronously), then their shared PSPI (overall output) will be more substantial ("voluminous") than that of a single team, and from a product owner's point of view, may sooner merge the gap between PSPI/MVP and MMP (more about factors influencing sprint length below).

In other situations, e.g., in non-scaled settings, individual Scrum teams could be dependent on other teams (Scrum, Kanban, Waterfall groups, etc.), separate organizational domains or external vendors that operate on their own cadence. In such cases, product backlog refinement and sprint planning becomes even more challenging. As a result, sprint length, as well as frequency of scheduled production releases may be impacted.

Who ultimately decides on sprint length?
Just like any other decision about Scrum team dynamics, the decision on sprint length belongs to a *team*. Nobody should be deciding *how* to structure or manage work on behalf of people that do it. Neither the product owner, nor stakeholders, nor management, nor anyone else. Teams that are new to Scrum may experiment with sprint length at the beginning while trying to optimize to conditions that are very specific to a team's dynamics. The best time to self-assess and decide if sprint length should be changed is during a retrospective when decisions about process improvements are made; it is done by majority voting. Only in rare cases, when a team has a difficulty to reach consensus, the Scrum Master, who *owns* the Scrum process and plays the role of an arbitrator in retrospectives, can step in and help a team make a choice. This should happen rarely, as frequent lack of consensus might be a sign of deeper team dysfunctions. Initially, during team formation, and before Scrum Master is elected, the Scrum/Agile coach may help a team with sprint length selection. Mike Cohn describes his personal experience in a similar situation here.

Factors to be considered while selecting sprint length?
As a rule of thumb, sprint length should not be shorter than one week and should not be longer than four weeks. If there is a strong reason to make sprints shorter than one week (e.g., could be driven by production release frequency requirements), Kanban, instead of Scrum, could be considered since it offers on-demand and almost instant release capabilities. On the other hand, extending sprint length beyond four weeks may lead to typical challenges of waterfall (sequential work, silos, hand-overs).

Statistically, anywhere between one and four weeks, a team should be able to establish a steady and healthy cadence to do product development.

Shorter sprints **do** have certain **advantages**:

- More frequent sprint reviews and retrospectives—shorten feedback loops that allow applying improvements to product and process development, respectively.
- Shorter sprints require lighter sprint planning and, subsequently, reduce the risk of going too deep into a product backlog and selecting items for a sprint that do not meet DoD (Definition of Done) and are not INVEST-able.

Shorter sprints may also bring some potential **challenges**:

- For example, the ratio of time spent on sprint preparation and process management to time spent on actual product development could be high—too much procedural overhead.
- Additional important prerequisites must be met before moving teams to shorter sprints. For example, if a sprint becomes too short (e.g., one week) and there is no full test automation and no TDD, then a team may have a difficult time keeping up with testing: after completing a few sprints, as the amount of code base increases, manual testing will fall behind. As a result, too much work may fail DoD by sprint-end.

Relationship between sprint length and Scrum maturity

It would be reckless to claim that there is a direct cause and effect relationship between sprint length and maturity. Some research indicates (as was done by Jeff Sutherland) that for as long as a sprint is under one month, there is no strong and immediate correlation between sprint length and performance. But anything beyond four weeks lowers performance and introduces elements of waterfall dysfunction to a team's dynamics.

What is, by far, more important than sprint length is sprint length *consistency*. While in early stages of sprinting, it is normal for a team to experiment with sprint length; if length "juggling" continues into later sprints or happens ad-hoc, it could be viewed as a sign of deeper problems. Some teams falsely believe that by periodically extending a sprint they will be able to get more work to done state. Thinking more systemically, this is a false assumption as cadence- and task-switching, especially done by multiple teams, can significantly lower overall output. Further, inconsistent sprint length will lead to difficulty of sprint planning, unstable velocities and unreliable long-term forecasting.

Product Development Types: What Challenges to Expect in Each Case?

Originally published on January 26, 2017 | Location: http://www.keystepstosuccess.com/2017/01/product-development-types-what-challenges-to-expect-in-each-case/

Not all types of product development are the same. The most three distinct types are the following:

- **Internal product development,** *where*
  - Users/customers are internal.
  - Product owner is internal, playing the role of a conduit between users and Technology.
  - Development/Technology/IT is internal, mainly dealing with product owner (and her team).
- **Outsourced product ("project") development,** *where*
  - Users/customers are internal.
  - Product owner/lead user is internal, playing the role of a conduit between internal users and external technology vendors.
  - Technology is external, represented by vendor-companies.
- **External Product Development,** *where*
  - Real (paying) customers are external.
  - Product owner (classic product management/business unit) is internal and facing external customer.
  - Technology/R&D/development is internal and interacts only with product owner.

While these classifications may seem to be obvious, there are some important nuances that are often being overlooked. Awareness of these nuances could be important for making product development practices less vulnerable.

Here are some potential pitfalls to look out for with each type of product development:

Note: This post does not provide any recommendations on how to avoid/reduce pitfalls. Some suggestions might be covered in future publications.

Internal product development

- Fake product owners (delegates, proxies, surrogates) that lack either authority, or vision/breath of product knowledge, or both.
- Artificial "internal contracts" between various company units, secondary to organization design. These, are often driven by internal competition, ineffective external motivation, subjective monetary incentives and bonuses.
- Technology is too remote from real customers, does not deal with them directly and must go through additional organizational layers, units, groups, roles (even to get to internal product owners).

- Excessive amount of processes, tools, metrics/KPIs and reports that are used as mechanisms for measuring business value delivered and levels of customer satisfaction. This is because of absence of real, short-loop customer feedback.

Outsourced product ("project") development

- Low mutual trust (both client and outsourcing company). Product development process is "wrapped" into artificial projects and portfolios, typically, with fixed scope, timeline and budget. Customer-vendor relationships are very contractual, with legal binders. While there are much more effective Agile contracts that exist today, they are rarely used because of lack of ability, by either side, to execute them.
- Documentation is voluminous and is at expense of collaboration and transparency. Approvals and sign offs are mandatory and are usually accompanied by rigid legal processes.
- Unfit vendors: many companies still rely on vendors with whom they have been dealing historically (via vendor management system), instead of turning to a marketplace every time there is a new need and searching for best candidates. Preferred vendors feel safe and comfortable, and with such "monopoly" in space, they have low, if any, desire to strive towards improving technical skills, adopting new technologies, etc. Also, nothing prevents such "preferred" vendors from making unsubstantiated claims about their experience with Agile product development: rarely cases studies or references from other clients are requested. Fixed contracts just make things worse.

External product development

This product development type is probably the best candidate for Agile approach: customers are real, real money exchanges hands, competition is real and often strong, fast market changes require product companies to be nimble, light-footed and fast-responding. However, there are still some challenges here.

Specifically, the communication channel called "Clarifications" (between Technology and external customer) that is crucial in Agile development (e.g., Scrum) is usually weak or does not exist at all. Due to geographic distribution, legal contracts and compliance issues, it is challenging to co-locate real customers and developers and have them directly engage with each other.

Since everything flows through product management /marketing/business internal organizational layer, technologists get their information second-hand. This also leads to producing additional overhead (processes, documentation, and approvals) by a product development company: multiple external communication channels and competing priorities (different customers) that must be collected/filtered/prioritized and then single-threaded to technology. To complete this job, the internal product owner will require help. This is how internal business units grow thick and as this happens more waste/overhead is introduced to a system.

Handling Interruptions in Scrum: Four Options (Part 1)

Originally published on September 2, 2015 | Location: https://www.projectmanagement.com/articles/302036/Handling-Interruptions-in-Scrum--4-Options--Part-1-

In an ideal world, a cross-functional Scrum team must be fully focused on Scrum. The team is also expected to hear a voice of one customer only: the product owner. But what happens when reality intervenes and you get pulled in other directions?

In this two-part article, I will do the following:

- Highlight common interrupt challenges I've encountered for organizations new to Scrum in general, and large organizations in particular.
- Outline four common interrupt scenarios and explain what you can do to handle these interrupts and maximize your productivity.

I'll use the example of my old-time friend "Joe," a senior software engineer who served as Scrum Master on one of the Scrum teams.

Common Challenges When New to Scrum

Shortly after Scrum training, Joe's team began sprinting. At the beginning, it was hard to gain momentum: the backlog was poorly defined (very few requirements were decomposed into INVEST-able stories), Scrum ceremonies were not well structured, the product owner initially struggled to prioritize a backlog, the team had to learn new estimation techniques, the Scrum Master's role was not claimed.

But these were all expected challenges, typically observed with newly formed feature teams. Soon, Joe's team started to demonstrate noticeable improvements. But as time went by, another serious challenge surfaced: Joe's team started to realize that it could not fully dedicate itself to Scrum.

Historically, the team has been responsible for all production support issues, Level 3 support requests and "one-off" items that have been submitted by end users. There was no other team that could pick up this stream of work while Joe's team focused on Scrum development. This "side" work remained the responsibility of Joe's team.

The challenge for large organizations

The above describes a very typical scenario that is observed in smaller organizations as well as large enterprise-size companies that undergo Agile transformations. It happens when newly formed Scrum teams cannot fully focus on backlog-centric product development and work for only one product owner—and according to only one set of priorities.

At large, multi-layered organizations, it is not uncommon that requests and requirements come from different angles and leak through shields and barriers that are meant to protect feature teams from distractions. Instead of going through a "filtering mechanism" of grooming and prioritization, work "leaks" directly to technology.

It is also worth noting that at large companies more so than at smaller companies, defects and other issues found in production code are not often caused by teams that are tasked with repairing them. This often means that the time required for a team to investigate a problem's root cause is significantly longer than it would be for a team that originally built the code. It also means that if a distracted team also operates as a Scrum team—by time-boxing its work in sprints—its velocity, cadence and ability to reliably forecast is hindered.

Four common options

There are four commonly known ways that teams use to handle production support or other "interrupt" items. Frequently, these items are classified by "levels" of priority/criticality, with L1 being the lowest and L3 being the highest priority. (For more information on generic definitions of support levels, please visit Joe Hertvik's blog entry.)

Here are the four commonly seen ways of interrupting Scrum sprints—and what can be done about them.

1. "Creep" and fix

When a team is requested to work on an item that was not initially committed to a sprint, it puts committed work and newly incoming work in jeopardy. It forces a team to stretch (creep) its sprint commitment beyond its capacity. This is contrary to what is suggested by historical velocity and capacity-driven sprint planning.

The impact is usually:

- Multi-tasking.
- Excessive task switching.
- Rushing to completing development work at expense of code quality.
- Lowered testing coverage and other problems.

This also indicates that the product owner does not have a good grasp of what the business priorities are and does not understand how disruptions may affect product quality, estimation reliability and, ultimately, a team's morale and customer satisfaction.

One of the ways to mitigate sprint creeping problems—and to minimize adverse effects of introducing new work in the middle of a sprint—is to do work "swapping." If new, unexpected work enters a sprint, something of the same (or close) size and complexity should be removed. Although a team's focus will still shift to new work (suboptimal), a

team's capacity will not get stretched. This may also result in sprint focus (theming) being changed in the middle of a sprint—something that might be confusing to business stakeholders and senior management. It becomes the product owner's responsibility to manage expectations of customers.

In general, this approach is an indication of dysfunction and low consideration to the principles of Scrum.

2. Stop and fix

This approach can be easily confused with the "stop and fix it" method that is sometimes seen in Scrum-only settings. It pertains to fixing bugs that were introduced by sprint work itself, not by work unexpectedly coming from outside.

In Scrum-only settings, if (while working on user stories committed to a sprint) a team introduces bugs, it would have to fix them in the same sprint to release bugless code at the end. Unfortunately, the notion of "stop and fix" is sometimes misused and taken out of context of pure Scrum. This happens when a team is requested to abruptly terminate a sprint and refocus its attention on other work that was not originally planned and committed. This is an example of "stop and fix" gone bad.

Although extreme cases of premature sprint termination due to external urgency happens (and at times are justifiable), if they happen too frequently, the benefits of using Scrum as a framework become reduced. Frequently, interrupted sprints lead to velocity volatility and inaccuracy of strategic product planning.

In cases when unexpected, unplanned work becomes a norm and predominates over planned work, it may be more suitable for teams to consider Kanban, which focuses on steady workflow and throughput instead of Scrum's iterative development and time-boxed delivery. (Read more about Comparing Kanban to Scrum.)

Handling Interruptions in Scrum: Four Options (Part 2)

Originally published on September 8, 2015 | Location: https://www.projectmanagement.com/articles/303519/Handling-Interruptions-in-Scrum--4-Options--Part-2-

There are four commonly known ways that teams use to handle production support or other "interrupt" items. Frequently, these items are classified by "levels" of priority/criticality, with L1 being the lowest and L3 being the highest priority. As we continue from Part 1 of this two-part article, we look at the remaining two ways of interrupting Scrum sprints—and share what can be done about them.

3. Triage and prioritize

This approach is most suitable when bugs/issues found in production are not extremely urgent. If findings are not life critical, they can be triaged by technologists (for size and complexity), prioritized by product owners and then be added to a backlog to be treated as all other PBIs (Product Backlog Items). This can be done during regular PBR (Product Backlog Review) sessions conducted by feature teams.

This approach is least disruptive to a team's dynamics as it does not have any impact on a team's velocity and cadence. A team does not have to dilute its focus and capacity that was allocated to sprint-based work by adding additional work mid-sprint.

This approach is frequently seen when a team is directly responsible for its production code (no L2 and L2 mid-layers). The product owner firmly controls communication with the business and is empowered to shield a team from undesirable interruptions and unwarranted requests.

What becomes important in this situation is for Scrum team members to become more mindful of their overall capacity and ways it is shared between Scrum and non-Scrum work. Teams that have developed a pattern of dealing with "interrupt" work and are able to forecast how much time in each sprint might be consumed by such side work should properly budget their time during sprint planning by allocating a portion of their capacity (various techniques are available: zero-point time-tracking stories that are placed inside a sprint, separate Kanban workflow visualization, etc.).

Note: Zero-story-points user stories are sometimes used as a "placeholder" to capture time (days, hours) spent on work that was not planned and committed. A team does not estimate such a story in story points; it is "hollow." Instead, a team allocates (roughly) a certain percentage of its capacity for doing side work (if such work arises) based on past sprint experience.

Over time, a team inspects if from sprint to sprint the same amount of time is spent on side work. If this process normalizes, a team may compare a side-work story with other planned stories of the same size/complexity and start adding many story points to its average velocity.

4. Let others fix

This approach is typically used when the ability to keep the Scrum team's focus on a new product backlog is so critical that the product owner cannot afford diluting it with anything else that is not immediately related to new product development. (Frankly, this should be always the case!)

In cases like these, there is a clear separation between new (scheduled) product development supported by dedicated Scrum teams and unscheduled ("ad-hoc") work that is supported by other groups of people—dedicated Kanban teams.

While the former type of work is funneled through the product owner and goes through PBR-ing (Product Backlog Refining) prioritization, estimation and sprint planning, the latter type of work gets triaged, primarily by technologists (business input might be minimal) and enters an independent Kanban workflow. In Kanban, ad-hoc work follows workflow management conventions and SLAs (Service Level Agreements) established by Kanban teams that are usually independent of the cadence and cycles of Scrum.

The work of Kanban teams should not impact a Scrum team's velocity and capacity as it relies on the Kanban team's own capacity, WIP (Work In Progress) limits, queue management techniques, escalation rules, and cycle time and throughput management.

Segregating side work from sprint-based work into a separate Kanban flow also creates other opportunities for further Kanban scaling by streamlining various ad-hoc requests that come from multiple sources/channels into a shared technology queue. This also helps promote the idea of non-IT Kanban implementation by extending Kanban workflow management principles further upstream from technology—something that helps an organization visualize and more effectively manage its book of work more comprehensively.

Recommendations

To summarize, if organizations really support Agile values and principles, they will try avoiding option 1 at all costs. It is a sign of bad organizational culture and dynamics, command and control behavior and the "faking" of Agile.

Option 2 should be used only in extreme cases when risks and losses of not addressing side work by far exceed risks and losses encountered by not continuing planned and scheduled development.

Options 3 and 4 are both much more benign and—depending on other variable factors—may work equally well. Choosing between these two options is often done based on organizational structure, skillset availability, frequency (density) of incoming side work, etc. The key advantage that these two options have over options 1 and 2 is that sprint-based Scrum does not suffer.

Scrum and Kanban at the Enterprise and Team Levels

Originally published on 13 June 2014 | Location: https://www.scrumalliance.org/community/articles/2014/june/scrum-kanban-enterprise-and-team-level

Scrum, as the most structured of all Agile frameworks, is a great way to ensure predictable, strategically planned, incremental product delivery. Scrum ensures good responsiveness to frequently changing market demands. Although nonprescriptive, Scrum clearly defines certain roles, responsibilities, and ceremonies. Kanban, for the most part, is silent about certain aspects that Scrum suggests explicitly (e.g., team size, velocity, story point estimation, timeboxing, Scrum ceremonies, etc.). Kanban is less structured than Scrum. Being a true pull-based system, Kanban is a great work-flow visualization tool that can be effectively used for WIP management. It is a great tool to use in production support or the gradual redesign of legacy systems; business priority-driven new product development is not the main goal.

Writing by David Anderson on "ScrumBan" does not require any introduction. Neither do the concepts that he summarized in his article some time ago; here he swiftly summarizes how Scrum and Kanban can effectively complement each other.

This post describes two practical examples of how, by combining Scrum and Kanban processes, improvements can be achieved at two levels: the individual team level and the enterprise level.

1. Individual team level: Using Kanban pull-based work scheduling intra-sprint (Scrum)
    - As per Scrum:
        1. Maintain a product backlog that is prioritized, based on the business value assigned by the PO.
        2. Conduct regular sprint planning sessions to define the scope of each upcoming sprint.
        3. Forecast/commit work for a sprint based on established velocity trends.
    - As per Kanban (once a sprint is in flight):
        1. Start pulling work from the very first ("to-do") queue, based on the difference between the WIP limit and vacancy in downstream queues. (Pulling the initial few stories should be easy as vacancy would be always equal to WIP.)
        2. Ensure that a team swarms around work (stories) in such way that in downstream queues, WIP limits are neither violated nor does the vacancy become too high.
        3. Treat an intra-sprint work flow as a "cyclic conveyor belt" that must never become too loose or too tight. Ideally, intra-sprint, throughput must be consistent at each queue. (More complex, multi-flow Kanban systems with various WIP queue limits are out of the scope of this discussion.)

To summarize, by using the principles of Scrum and Kanban together, teams achieve more precision in delivery forecasting. While Scrum's historical velocity helps in controlling WIP limits at the sprint level (sprint scope control), Kanban helps control WIP with more surgical precision at queue level. Effectively, Kanban helps optimize team swarming and therefore ensures gradual workflow from the beginning of a sprint to its end. Effectively, intra-sprint Kanban workflow management is the second line of defense to ensure the predictability of Scrum delivery.

4.

2. Enterprise level:
   - Visualize non-IT work using Kanban.
     1. Look holistically at the enterprise-wide activities that take place before a work request comes down to IT. Using queues, visualize all gates and decision points that the work request goes through before it comes to technology.
     2. Determine whether upstream queues are part of one common flow or represent distinct, converging flows (e.g., finance, legal, marketing, operations departments, etc.).
     3. Apply reasonable WIP limits to all upstream queues, paying attention to the point of converging flows. For example, if finance, legal, and marketing all flow into operations, then the WIP limit of the first operations queue must be high enough to accommodate three batches of work coming from finance, legal, and marketing.
   - Introduce a Kanban-versus-Scrum decision point before the work request comes to technology.
     1. If the work request is qualified as "new development," route it through the Scrum framework by having PO(s) review the work, enter it in a backlog, and prioritize accordingly.
     2. If a work request is qualified as "production support" (bug, improvement), have technology quickly triage it to understand its urgency and the cycle time required to complete it. If it is "quick kill" (depends on the definition of the "quick-kill cycle time" of a given organization), route it to a Kanban team or teams.
     3. If the work routed through Kanban turns out to be much more complex than initially expected and there is danger of the clogging Kanban flow, reroute it from Kanban to Scrum (channel it through the PO, backlog). This assumes that such rerouted work does not present "stop ship" urgency.

     To summarize, by using a Kanban-Scrum hybrid model at the enterprise level, a high degree of visualization and work flow control can be achieved much sooner than when work enters the technology space. This, subsequently, ensures a more predicable flow of work within the technology space itself. Once work enters the technology space, additional decision-

making checkpoints can be applied to ensure that priority and complexity/size of work determine which Agile framework is being used for its completion. Rerouting should be an option.

More to come about how to optimize the use of technology resources to support both Scrum and Kanban models.

Epic-Level Estimation

Originally publsihed on 23 December 2014 | Location: https://www.scrumalliance.org/community/articles/2014/december/epic-estimation

Imagine: You are about to form a new feature team that is composed of bright, cross-functional experts, self-motivated and self-managed. They all worked in Scrum settings before and are fully supportive of Agile principles. The organization they work for is properly structured and it nicely supports the adoption of Agile/Kaizen culture.

Imagine that the team interfaces with an experienced PO who comes from an end-user community, has a lot of client-facing product management experience, fully understands principles of Agile product development, does not tolerate waste, appreciates benefits of incremental value delivery, is fully empowered, and knows how to effectively partner up with technology.

Oh . . . and imagine this: Both the team and the PO are *properly incentivized* to work together toward the same common goals and interests. Would it not be great if every company out there was like this?!

Here are a few more "facts":

- Since every team member already has experience with Scrum, the whole team has great potential to jell quickly, establish a work cadence, and start delivering value soon. However, since the team is still relatively new, it does not have a steady velocity.
- The PO has a well-defined strategic product vision and product road map and is clear on what/how frequently he wants to release to production this year.
- The backlog contains at the top all highest-priority stories and they are well defined (meeting the Definition of Ready).
- As the PO decomposes larger chunks of work (epics) into more manageable pieces (stories), he preserves relationships between epics and stories. Why does he do that? Well, this is what helps a PO retain a strategic view for the entire product; this is also how he manages his product road map. Some epics span multiple sprints (some are large) and, while individual stories still get delivered "per sprint" (meeting the Definition of Done for each sprint), the PO decides to go to production only when some important epics are done in full. At a glance, this may seem not the most Agile approach, but from a strategic perspective it does make sense to the PO: He must consider costs and overhead associated with synchronization of delivery from multiple Scrum teams, efforts associated with communication to and training of the end-user community, etc. The PO well understands principles of economics of Lean product development and has done a great job in the past of optimizing batch size of new features and functionalities before they were rolled out to production.

Time goes by . . . and the team completes a few sprints and establishes some velocity. This means that now the team knows its "burn rate." Now if the team got a scope, it would be able to forecast when all known work could be done. It would probably produce a "cone of uncertainty" that would be equal to the gap between optimistic and pessimistic forecasting. The opposite would be true too: If a team were given a fixed release date, they could inversely forecast how much work could be done in that time (again, a "cone of uncertainty" would be present).

Further, every team member knows from his/her past work experience how to apply story pointing techniques to estimate work, and this makes story estimation during PBRs and planning a pleasant exercise for the whole team because there is no fighting, no estimate padding, no resistance, no "CYA." However, since a team does not release to production at the end of every sprint, the PO wants to be able to forecast on a time scale that is broader than a single sprint (across multiple sprints). In other words, the PO wants to know how many sprints (given a team's burn rate) it may take before all of his intended scope will be done. The PO wants to know *when* he will be able to deliver a product to end users. However, it turns out that he *cannot* do so reliably. Why?

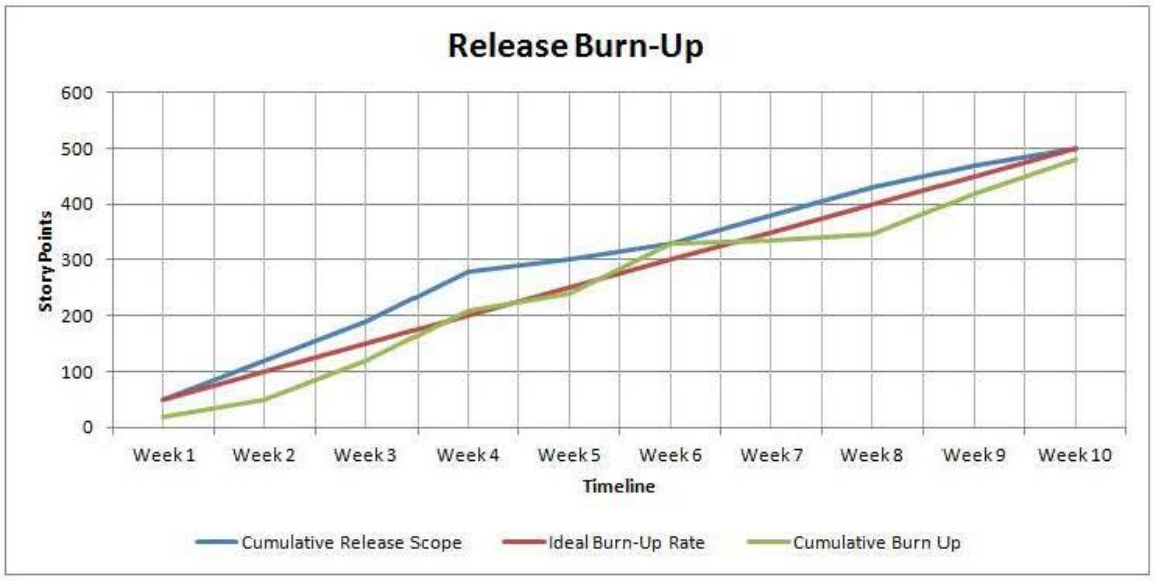
Note: In simple Scrum (individual team(s) working on a separate product), *not* releasing to production at the end of every sprint is *not* desirable. Unfortunately, at enterprise-size companies, with many post-development activities that "must happen," releasing to production at the end of every sprint is challenging. Also, in scaled Scrum (LeSS, SAFe), with many teams working on the same product, it is frequently a conscious strategic business decision to release to production only after a few sprints worth of development (by multiple teams) are complete. It is important to remember that although a team may not be required/able to release to production at the end every release, it still must produce a potentially shippable product. Waiting for "collective deployment" is not an excuse to produce partially done work.

So, back to the topic: The PO wants to know *when* he will be able to deliver a product to end users. However, it turns out that he *cannot* do so easily. Why?

Here is why:

The team estimates work in a backlog at the rate that is practically the same as a team's burn rate. In other words, release scope grows (gets estimated) gradually and just a bit ahead of sprinting. If a team burns roughly 50 story points per sprint, it estimates upcoming work in a backlog that is up to 70 story points worth of effort.

In a case like this, here is how the Release Burn-Up chart looks with respect to a scope:



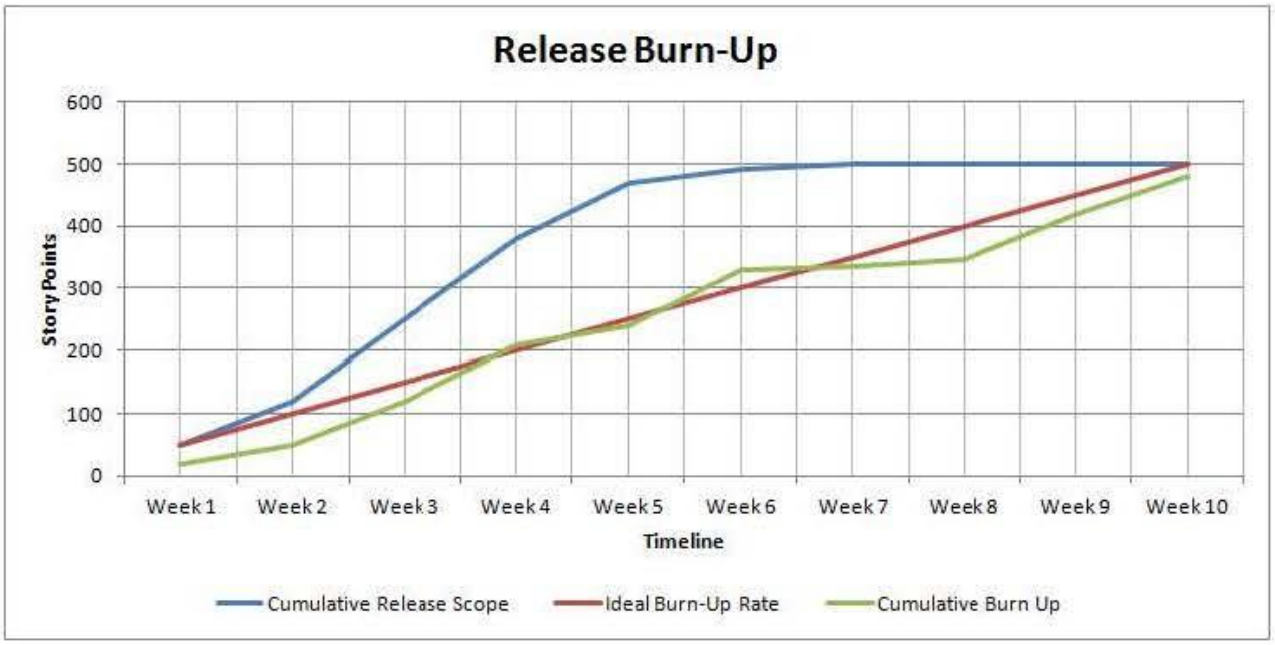
Here, the green line that represents cumulative burn-up rate of the team fluctuates up and down with respect to the ideal burn rate line (maroon). The blue line that represents cumulative release scope also grows. However, it does so just a bit above the burn-up rate—JIT (just in time). So, effectively, a release scope is a subject to continuous increase.

At this point, you may say, "But this is *exactly* how it should be done in Scrum—scope should never be fixed!" And you are right: Scope should remain fluid in Scrum. This is the whole beauty of Agile, that scope, time, and budget are never fixed at the same time. Then you may also say, "A team should not attempt to estimate all stories in a backlog in depth, because what is at the bottom of a backlog is not groomed well enough, and any attempt at estimating vaguely defined and not well-groomed stories would increase the margin of error and unwanted variability." And you are right *again*: Estimating an entire backlog based on poorly defined stories is unreliable and wasteful. If the PO asks the team to provide estimates that are based on a low level of understanding, he will eventually end up communicating unreliable dates to his end users and stakeholders. This will probably lead to finger-pointing, blame-gaming, and a deteriorated reputation on the side of both business and technology.

So what can be done to determine a multi-sprint release scope while *not* introducing too much variability and uncertainty by micro-analyzing what is still vaguely defined? What can a team and its PO do to get Cumulative Release Scope to reach a plateau *sooner* and therefore forecast a reliable release end date *sooner*?

Well, there might be a way.

The image below illustrates a situation where, around Week 5, the PO is able to bring reliable forecasted dates to his business partners:



Remember that the team works for an experienced PO who is not only good at effectively decomposing epics into stories but also at preserving relationships between overarching epics and underlying stories; release-level forecasting is important to him! The PO is a great product strategist with a holy vision!

On the other hand, the team is composed of experienced Scrummers who, after sprinting together for a while, have developed a good shared understanding of each other's complexity-estimation style. Therefore, their estimation scale is reliable too. In other words, for this team, a 13-story-points user story most of the time proves to be bigger than an 8-story-points user story, which is bigger than a 5-story-points user story, and so on. In other words, the team's estimation scale is reliable.
"Great," you may say, "but what is next?"

This is what can come next:

Since the team has been sprinting for a while, some of its epics eventually get fully done (all underlining stories are accepted). This enables the team to do some very basic "reverse mathematics":

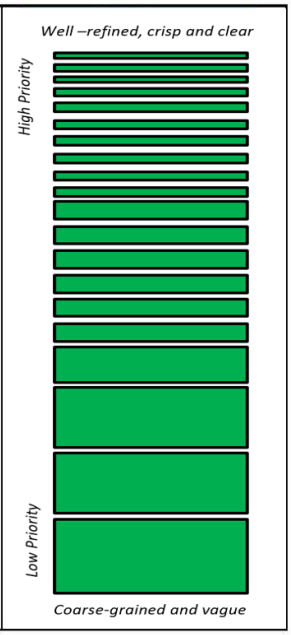
- If Epic A is composed of ten stories (of various complexity) totaling, let's say, 50 story points, then it would be fair to say that Epic A is 50 story points.
- By using the same logic, if Epic B is composed of 15 stories totaling 80 story points, then Epic B = 80 story points.
- The same goes for completed Epic C, Epic D, and so on.

Based on the above, what the team is now able to introduce is an epic-level estimation scale. This scale can now be applied to estimate the remaining backlog (in depth) at the epic level. Nothing should prevent a team from comparing one epic against another to establish relative sizing.

Usually, it is advisable to estimate epics.

The beauty of having a handful of epics fully done is that they can then be used as a baseline from which to relatively estimate all remaining epics. This coarse estimation prevents a PO from rushing into epic-to-story decomposition beyond what is needed at the time, and prevents the team from rushing into "guess-timation" of poorly defined stories.

Instead of pretending that the team understands enough about individual, poorly defined stories, it "swags" against larger, more visible targets – epics. By "pooling together" poorly defined work, the team reduces noise in estimation and therefore reduces variability, while subsequently increasing estimation accuracy.



Many teams prefer not to estimate epics on par with smaller stories, specifically to emphasize that the former are much larger than the latter and the two are not comparable. Teams frequently use a Small (S), Medium (M), Large (L), Huge (H) scale to estimate epics—this is faster and cheaper. But for teams that have a good track record of decomposing epics into stories and completing individual stories, and by doing so completing epics, another opportunity presents itself.

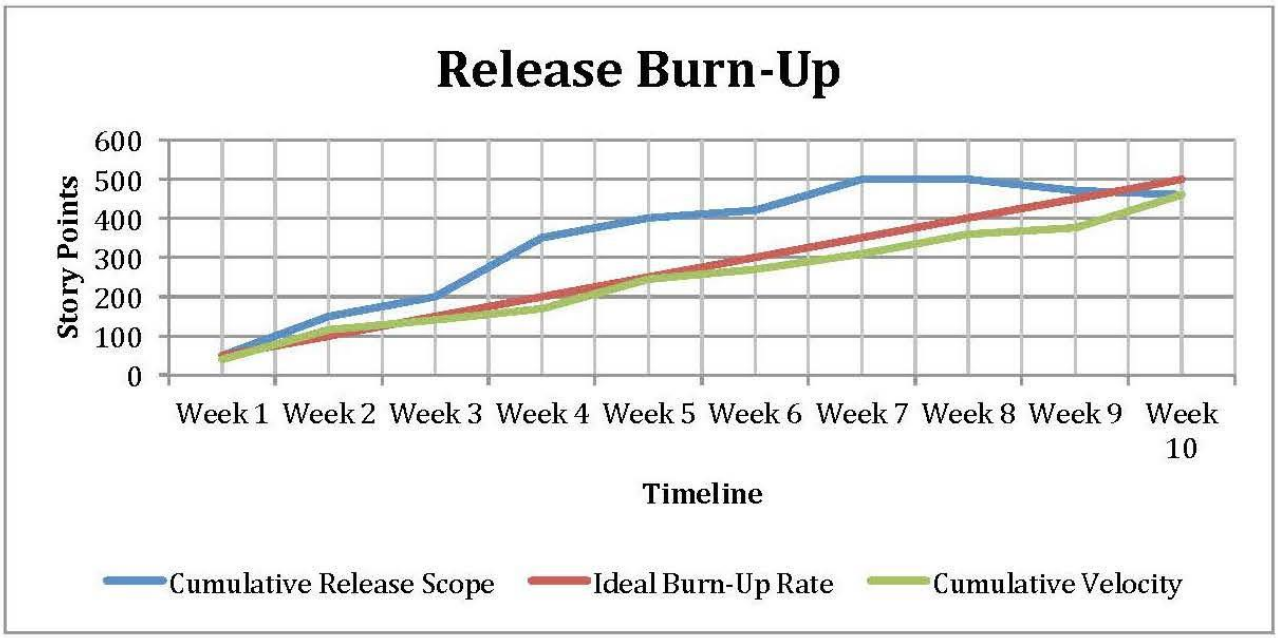
Leveraging the "reverse mathematics" described above, a team can now not only compare new epics by using the S-M-L-H scale but also compare new epics to completed epics and "normalize" the S-M-L-H scale to story points scale, without actually "guess-timating" new epics in story points.

Again, in order for this to work, a team must have a good track record of breaking down epics into stories, estimating and completing individual stories and maintaining initial relationships between epics and stories.

But there is something to watch out for:

When a team starts working on an epic (that is "normalized" to story points) by chipping off, estimating, and developing individual user stories, it would also have to ensure that the overall scope does not get inflated because of double booking. For example, if Epic X were initially "normalized" to be 80 story points but then the team chipped away and individually estimated one five-story-pointer and two eight-story-pointers, then it would make sense to "back out" 21 story points from the 80, to keep the overall scope of epic in balance. What could happen by the time that an entire epic is complete is that a cumulative estimation of individual user stories ends up being less than, more than, or

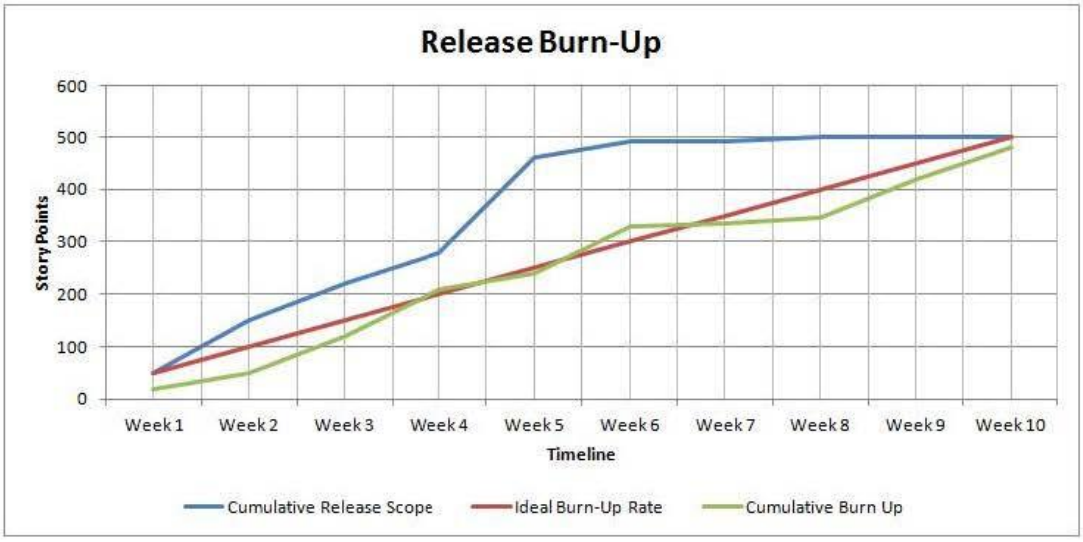
adds up to 80 story points (miracle!). As an example, if the final sum of all chipped-off and individually estimated stories turns out to be less than the initial 80, the overall scope would have to be adjusted downward, as it is shown at around weeks eight to nine below:



At this point, it is worth nothing that scope decrease (de-scoping) can be done as the team approaches a planned release date for other reasons as well. Among others, a good "Agile reason" why the PO may want to "trim the tail" of scope and deploy sooner is purely a business decision: There is already enough business value in delivered code to justify going to production.

Here is an example of a team that after a period of sprinting has decided to use epic-level estimation and normalization techniques to forecast a multi-sprint release, based on the existing track record of story pointing and well-maintained relationships between overarching epics and underlining stories.

In the diagram below, a noticeable spike in Cumulative Release Scope around Week 4 is due to epic-level normalization to story points and "plugging" them into a release scope:



This pretty much sums up the approach. Obviously, to implement it, a team and PO should have some history together: estimating, delivering, having a number decomposed and delivered in full epics, etc. Also, there should be no resource attrition on a team. Having a reliable, continuously engaged PO, whose epic/story writing style remains consistent over time, is another important requirement.

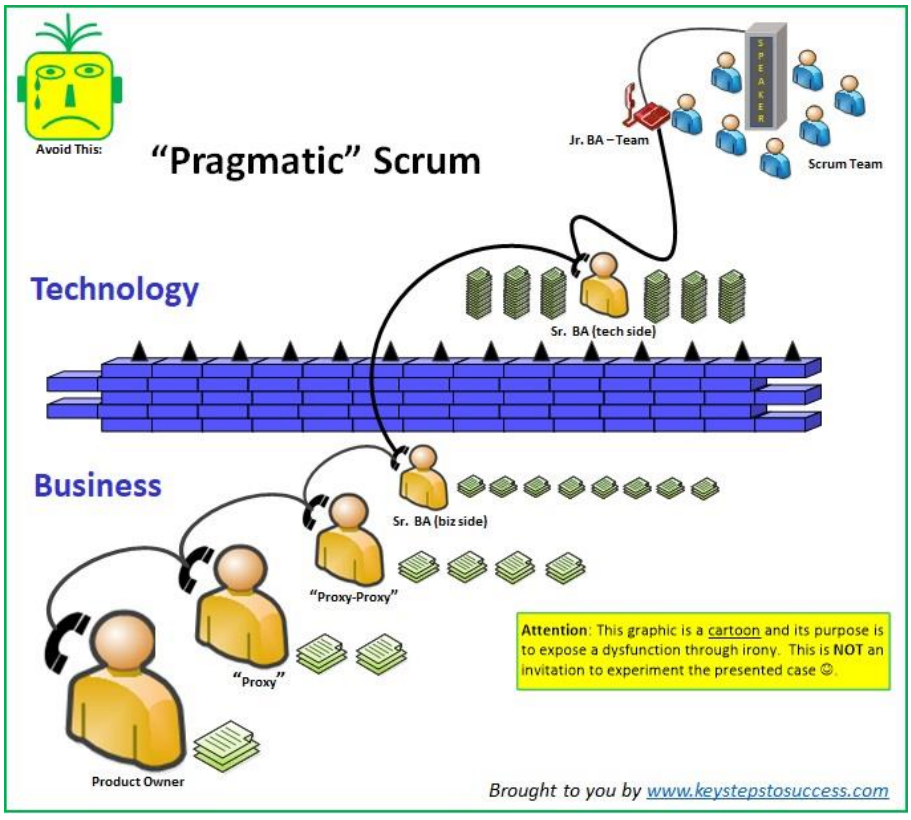
Agile Anti-Patterns

Not All Scrum Anti-Patterns Are Easily Identifiable

Originally published on June 14, 2019 | Location: http://www.keystepstosuccess.com/2019/06/not-all-scrum-anti-patterns-are-easily-identifiable/

"**This is not true Scrum!!!**"—we often hear from people, pointing at omissions and pitfalls of fake Scrum implementations. The list of classically seen Scrum anti-pattern could become pretty exhaustive. Here are some examples of "Dark Scrum" (as Ron Jeffries puts it):

- Multiple people playing the role of product owner.
- No Scrum Master present or line manager performing the role.
- Scrum events are dismissed, in favor of status calls.
- PMO or first-line managers driving team dynamics, assign and monitor team work.
- Teams are understaffed and miss required skillset to complete work.
- Too many translators-BAs that sit between real customers and development teams (see below).



But not all Scrum anti-patterns are equally obvious. Let's take a look at three instances of team-sprinting, where (1) Scrum anti-pattern is obvious, (2) Scrum anti-pattern is not obvious, (3) Scrum is done well.

Just before we proceed, let's recap (paraphrase) what the Scrum Guide says: In Scrum, in every Sprint, a team delivers Potentially Shippable Product Increment (PSPI). ***This is fundamental for Scrum***. In order for this to happen, each team must possess all necessary attributes (skills, knowledge, domain expertise) required to get work fully DONE (potentially shippable). This is what makes Scrum—real Scrum. Many teams that lack the required Scrum attributes still attempt to sprint, however, effectiveness of such "sprint-like activities" is significantly reduced. ***Not all anti-patterns of Scrum are equally obvious***.

Instance 1: Scrum anti-pattern is obvious

- Separate, phase-specific backlogs or…
- Single backlog with phase-specific items.
- Local optimization by single-skill specialists (e.g., PM, BA, QA, Architect, Developer).
- Hand-overs, toll gates, "internal contracts."
- Long periods of down-time by specialists, when it is *not "their phase" to work.*
- "Water-scrum"/" Scrum-fall."
- Very weak Definition of Ready and Done.
- **PSPI—takes many sprints to produce.**

Instance 2: Scrum anti-pattern is *not* obvious

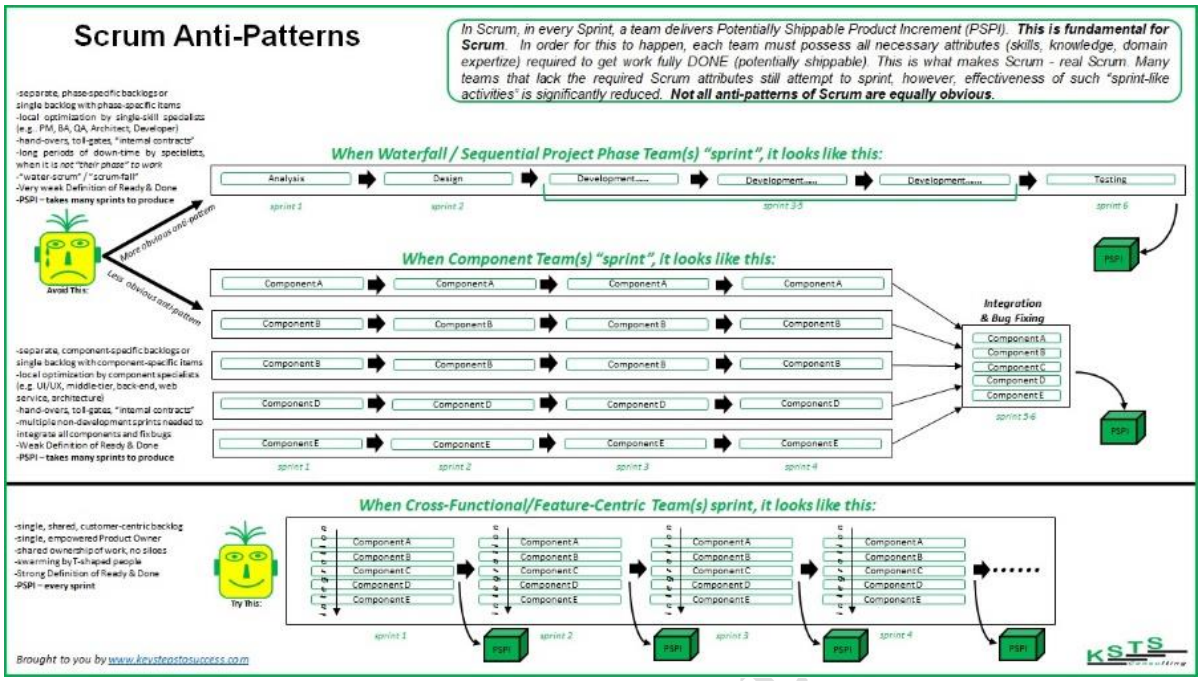
- Separate, component-specific backlogs or…
- Single backlog with component-specific items.
- Local optimization by component specialists (e.g., UI/UX, middle-tier, back-end, web service, architecture).
- Hand-overs, toll gates, "internal contracts."
- Multiple non-development sprints needed to integrate all components and fix bugs.
- Weak Definition of Ready and Done.
- **PSPI – takes many sprints to produce.**
-

Instance 3: Scrum is done well

- Single, shared, customer-centric backlog.
- Single, empowered product owner.
- Shared ownership of work, no siloes.
- Swarming by T-shaped people.
- Strong Definition of Ready and Done.
- **PSPI—every sprint.**

For organizations and teams that are new to Agile and Scrum, and they did not make a required investment in proper training/education, have not made an effort to improve organizational design for better agility and do not have experienced Agile coaches supporting them in their journey, option 2 above, may seem as an "OK way to sprint." **But please, beware, that you are not getting a real benefit of true Scrum, when you have so much "Undone" work at the end of each sprint cycle.**

Below is a graphic illustration of the three types of sprinting described.



Survival List to Vendor Selection on Agile Projects

Originally published on January 29, 2019 | Location: http://www.keystepstosuccess.com/2019/01/survival-list-to-vendor-selection-on-agile-projects/

How to Choose Vendor:

- **Vendor Management System (VMS)**—The easiest thing could be to refer to and pick from VMS, as long as a vendor card-rate is in a ballpark of what you wish to pay. But please, don't do that. Do not let costs become the most important determining factor in your selection. There is a chance that a vendor you are about to choose ended up in VMS based on old selection criterion, other than the ones that you might be looking for, for Agile work. Do not automatically assume that old relationships will seamlessly work under new conditions, operating in new ways.
- **Case studies/use cases/references**—These, could be great ways to understand if a vendor is really capable of doing work they were tasked to do. As a client be always skeptical about heavy, well-formatted power point decks with lots of fine-print if they are used by a vendor during initial presentations. Ask for practical demonstrations, working solutions and engage a vendor in extensive Q&A—and please make sure that real hands-on doers present/answer, NOT engagement/sales managers that are specifically trained to make a great first impression. Whenever possible, ask for references from other clients of the same vendor who can provide feedback about similar work that was performed for them.
- **Interviewing vendor workers**—Make sure that you interview every person from the vendor-side who will be involved in performing work. Be on the lookout for workers that were just hired by a vendor or swapped (for other workers) last minute, just before work commenced, or were being asked to split their time with other projects/clients.

How to structure your ongoing relationship with vendor?

- **SOW**—Regardless of SOW type (e.g., Design/Detail, T&M, Performance Based) try avoiding "fixed everything" (time, scope, budget) agreements. When all three corner of the "management triangle" get locked, work becomes very non-adaptive/non-Agile/rigid. It will increase risk aversion and decrease interest to experiment and innovate. Try building in some contingencies and flexibility into one of the three above variables. Don't fix-plan work by using waterfall tooling (project plans, Gantt charts, etc.).
- **Location of vendor people**—Ideally, bring vendor workers on-site (client) and fully integrate them (physical space, daily interaction) with your internal workers. Once engaging vendor people, don't treat them as second-class citizens. Engage them in team-bonding and other social activities to minimize polarization and other adverse behaviors that are not supportive of Agile teams. If geographic distribution is inevitable, at least try to engage with a vendor in the same time zone.
- **Communication with vendor people**—Communicate directly with doers, not with their line/engagement managers or alike proxies/conduits. Make sure that intra-team (e.g., Scrum, Kanban) relationships between vendor people and client people prevail over reporting relationships on a vendor side.
- **Investing in vendor learning**—Invest in education and training of vendor people if you think this will strengthen your relationship and there will be a notable ROI while they work for you (client). Be also wise about who you invest into and to what extent. Make sure you don't (over)invest in what a vendor was expected to know in the first place.
- **Multiple vendor involvement**—Be on the lookout for any signs of potential rivalry or competition between multiple, concurrently engaged, vendors—this will jeopardize a healthy working environment. Avoid assigning activities to different vendors in ways that increase hand-overs and lead to additional contractual relationships and sequential work (e.g., vendor A: design/development; vendor B: architecture; vendor C: testing).

How to track progress of your relationship with vendor?

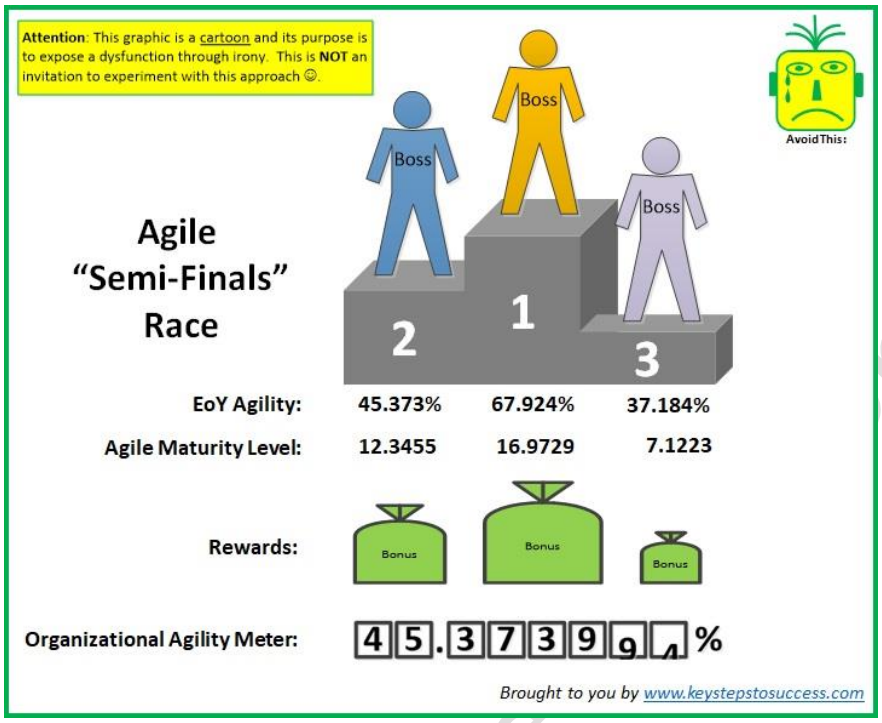
- **Progress tracking and communication media**—Select a single "source of truth" to capture, track and visualize work by Agile teams. If a physical board is not sufficient, leverage an electronic tool but make sure that there are no multiple versions of information (metrics, reporting, statuses, RAGs, etc.). Try basing all communications to senior leadership and stakeholders on raw metrics and empirical data that come directly from teams without passing through multiple layers of massaging and refinement. Avoid having a vendor using one set of work management tools and you (client) having another set.

How to position product ownership in fully outsourced (vendor) solutions?

Client-Vendor interaction—Make sure that product ownership represents you (client), clearly and unambiguously. A product owner should be positioned organizationally in a way that he/she faces externally, and communicates with/sets priorities to a doers/team members (vendor) directly (not through engagement managers, BAs or other translation layers), as well as internally by closely interacting with SMEs, stakeholders and other internal customers, with the latter providing clarifications but not setting priorities.

Addressing Problems Causes by AMMs

Originally published on October 15, 2017 | Location: http://www.keystepstosuccess.com/2017/10/addressing-problems-caused-by-amms/



Source: http://www.keystepstosuccess.com/wp-content/uploads/2018/07/cartoon_internal_agile_competition-2.jpg

Nowadays, for too many organizations, **Agile Maturity Metrics (AMM)** have become a trusted way to measure improvements of Agility at personal (individual), team and organizational level.
However, it is not always apparent to everyone that AMMs are different from Agile checklists (e.g., classic example of Scrum checklist by H. Kniberg), and this can often lead to problems and dysfunctions: Checklists are just a set of attributes that are usually viewed on-par with one another; they are not bucketed into states of maturity. (Other logical grouping could be applied though.)

On the contrary, AMMs place attributes in buckets that represent different states of maturity, with one state following another, sequentially.

With very rare exceptions (favorably designed organizational ecosystems), there are three potential challenges that companies face when relying on *bucketed* AMMs:
1. System Gaming: If achieving a higher degree of Agile maturity is coupled with monetary incentives/perks or other political gains (for many companies that are driven by scorecards and metrics, this is the case), there will always be attempts by individuals/teams to claim successes/achievements by "playing the system" in pursuit of recognition and a prize.

2. Attribute-to-maturity level relationship is conditional, at most: Placing Agile attributes in maturity buckets implies that attributes in higher-maturity buckets have more weight than attributes in lower-maturity buckets. However, this is not always a fair assumption: weight/importance that every organization/team places on any given attribute, while defining its own maturity, is unique to that organization/team. *For example, for one team, "being fully co-located and cross-functional"* could be much more important than *"having product owner collocated with a team."* For another team, it could be the other way around.

3. Correlation between attributes is not linear, at system-level: Regardless of buckets they are placed in, many Agile attributes are interrelated systemically and impact one another in ways that is not apparent to the naked eye. For example, placing *"Scrum Master is effective in resolving impediments"attribute* in a maturity bucket that comes before the maturity bucket with *"Organization provides strong support, recognition and career path to Scrum Master role"* attribute, dismisses the real cause-and-effect relationship between these two variables, misleads and sets false expectations.

To avoid the issues described above, it would be more advisable to treat every identified Agile attribute as a system variable that is on-par with other system variables, while assuming that it has upstream and downstream relationship. In many situations, instead of spending a lot time and resources on trying to improve a downstream variable (e.g., trying to understand why it is so difficult to prioritize a backlog), it is more practical to fix an upstream variable that has much deeper systemic roots (e.g., finding an empowered and engaged product owner who has as the right to set priorities).

Below, is the list of Agile attributes (a.k.a. system variables) that are logically grouped (checklist) but **are not** pre-assigned to levels of maturity (all flat). Some examples of

suggested system-level correlation between different attributes are provided (cells are pre-populated).

Note:
1 - Download the matrix to your local PC and study all the Attributes (Column F). Ensure they may sense to you. If necessary, add/remove/update the attributes.
2 - Assign Yes/No value to each attribute (Column G)
3 - Decide if any attributes in Column F have upstream dependencies on other attributes in Column F. List upstream Attribute # (Column E) in Dependency on Other Attributes? (Column H). Some cells a
4 - Decide if improving an attribute in best to be done by directly acting towards an attribute OR acting upon its upstream dependency.

Attribute Grouping			Attribute #	Attribute	Attribute Value (Yes/No)	Dependency on Other Attributes?
			38	Is entire Team in attendance?	N/A	
			39	Is the event's format appropriate and helps Team proceeding through Sprint?	N/A	
	Scrum Artifacts	Product Backlog (PBL)	40	Does the artifact exist?	N/A	
			41	Is it highly visible?	N/A	
			42	Is it prioritized?	N/A	
			43	Is it estimated?	N/A	
			44	Is it continuously refined?	N/A	
			45	Is it treated as a single source of prioritized Team work?	N/A	8,9,10
		Sprint Backlog	46	Does the artifact exist (based on DoR)?	N/A	58
			47	Is it clearly defined at the beginning of each sprint?	N/A	
			48	Does it reflect priorities of PBL?	N/A	
			49	Is it supportive of sprint vision?	N/A	
			50	Is sprint scope protected?	N/A	
		Product Increment (PSPI)	51	Does it get delivered at the end of every Sprint?	N/A	
			52	Does it meet Acceptance Criteria?	N/A	
			53	Does it meet DoD?	N/A	59
			54	Does it get delivered to end-customer (production)?	N/A	
	Scrum Team Dynamics		55	Is Sprint of appropriate duration (1-4 weeks)?	N/A	
			56	Is Sprint duration chosen by and optimized for Team and its work?	N/A	
			57	Does Sprint duration remain consistent?	N/A	47
			58	Does Definition of Ready (DoR) exist?	N/A	
			59	Does Definition of Done (DoD) exist?	N/A	
			60	Is there shared ownership (commitments, deliverables) by Team?	N/A	93-96, 99,100
			61	Is Team size appropriate (3-9)?	N/A	13-16
			62	Is Team co-located?	N/A	
			63	Is Team's structure flat?	N/A	
			64	Does Team has all skill set required to work on PBL?	N/A	15
			65	Does Team have hard external dependencies?	N/A	64
Agile Transparency & Information Radiation	Strategic Planning		66	Does Product Owner have Product Vision?	N/A	9,10,11,12
			67	Does Team have clear understanding of Strategic Goals?	N/A	9,10,11,12
			68	Is Scope fixed?	N/A	9,10,11,12
			69	Is Timeline fixed?	N/A	9,10,11,12
			70	Is Budget fixed?	N/A	9,10,11,12
			71	Does Team rely on historical Velocity to forecast PBL completion?	N/A	

Please, follow the link below to download the matrix to your desktop. Amend the list of attributes if you feel that your situation calls for modification, and then use "Dependency on Other Attributes?" column to better visualize system-level correlation between the attributes that are of interest to you and other related attributes (some examples provided):
http://www.keystepstosuccess.com/wp-content/uploads/2017/10/KSTS_AMM.xlsx

Bad Choice of Verbs Associated with "Agile," by EFL People

Originally published on June 12, 2017 | Location: http://www.keystepstosuccess.com/2017/06/bad-choice-of-verbs-associated-with-agile-by-efl-people/

These days, almost everyone knows that organizations cannot "*do*" Agile; they can "*be*" Agile. And today, this contrast is used not just by Agile coaches and scrum masters. Everyone likes building this fancy figure of speech in their daily lexicon: managers, analysts, developers. Great! Below is the quote from Wikipedia defining the word "**agility**," using the most natural reference: a human body:

"Agility or nimbleness is the ability to change the body's position efficiently, and requires the integration of isolated movement skills using a combination of balance, coordination, speed, reflexes, strength, and endurance. Agility is the ability to change the direction of the body in an efficient and effective manner."

Source: https://en.wikipedia.org/wiki/Agility

From reading the definition, it appears that body agility is equivalent to a body's fitness/health. And if so, it would be fair to assume that when we talk about organizational agility, we also talk about organizations being fit and healthy (organizational fitness/health). Just like a body *cannot* "do fit or do healthy," organizations *cannot* "do fit or do healthy."

But while the wrongfulness of "doing Agile" is mostly admitted today, there are many examples of using other *sophisticated* synonyms of "doing" that hint to the fact that people are still NOT clear about what agility is.

As the title of this post suggests, and this is where the biggest irony comes from, the most advanced EFL (EFL = English First Language) people have been making the most noticeable language omissions while attaching "sophisticated corporate terms-verbs" (other than "do") to the word "agile."

Below, is the list of verbs that are not advisable to be used in conjunction with the word "agile":

- *"Implement Agile"*
- *"Adopt Agile"*
- *"Use Agile"*
- *"Introduce Agile"*
- *"Accept Agile"*
- *"Follow Agile"*
- *"Move TO Agile"*
- *"Transition TO Agile"*
- *"Transform TO Agile"*
- *"Install Agile"*
- *"Administer Agile"*
- *"Leverage Agile"*

- *"Upgrade to Agile"*
- *"Practice Agile"*
- *"Establish Agile"*
- *"Experiment Agile"*
- *"Standardize Agile"*
- *"Execute Agile"*

What is advisable instead is just to **BE** agile.

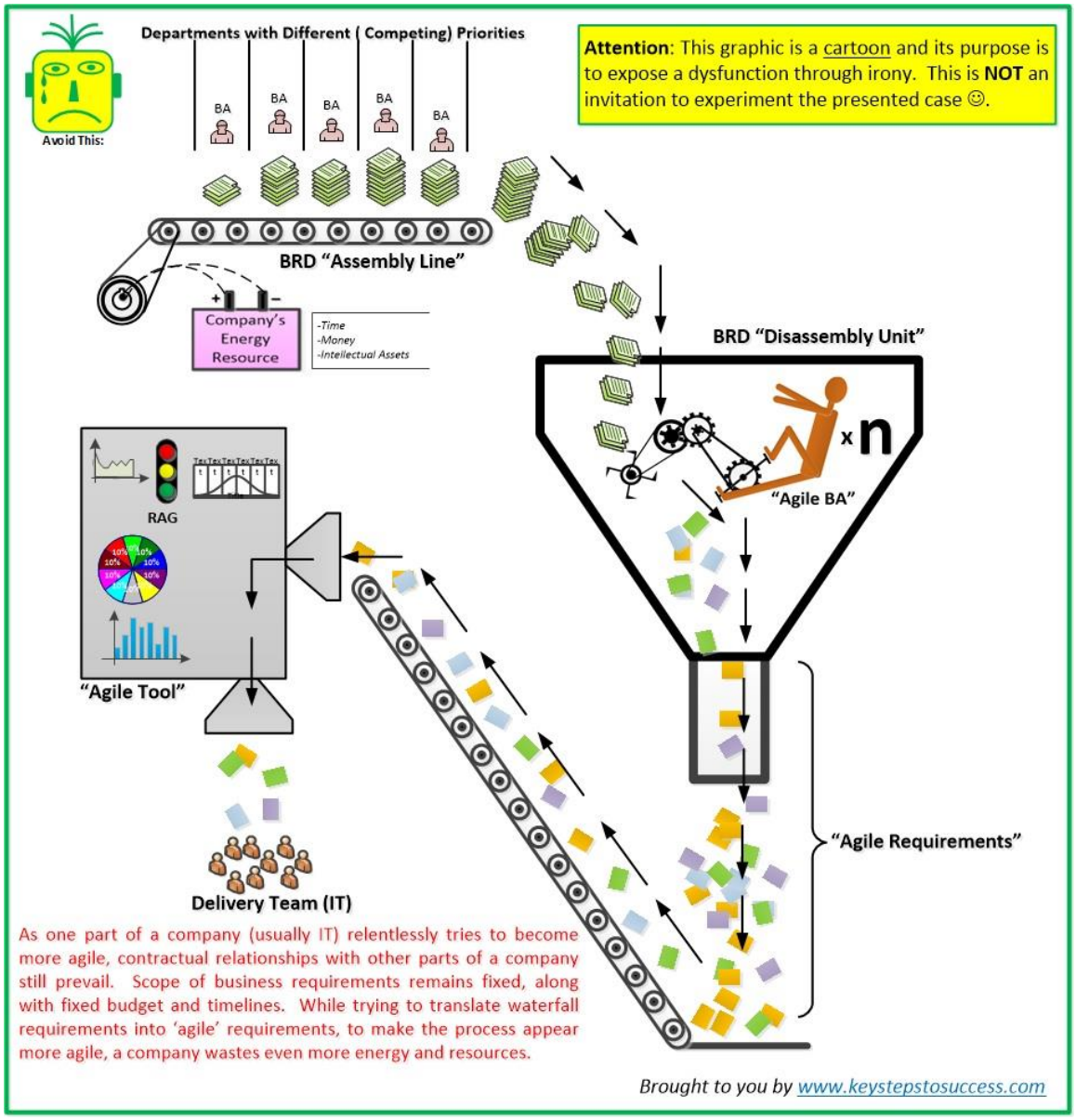
Waterfall Requirements in Agile Product Development

Originally published on Aug 11, 2016 | Location: https://www.infoq.com/articles/waterfall-requirements-agile

Key takeaways

- By nature, BRDs imply conclusiveness and are "contractual" in nature, and therefore should not be used in iterative development.
- BRDs are frequently produced by middle men and weakly represent views and desires of the end-customer.
- Creating BRDs, done by a separate organizational layer, frequently leads to local optimization.
- In Agile development, BRDs may lead to deterioration of relationships between the product owner and the feature team.
- In Agile development, the use of BRDs leads to having false expectations and reduces involvement of the end-customer.

Everyone is familiar with conventional Business Requirements Documents (BRDs).



Source: http://www.keystepstosuccess.com/wp-content/uploads/2018/07/cartoon_brd_assembly_line.jpg

Usually, BRDs are written by Business Analysts—individuals representing a function specialty organizational tier—a layer that usually resides between end-customers and software engineers. BRDs explicitly talk about "what" a customer wants from a system, including functional and non-functional requirements. BRDs are comprehensive documents: heavy in content and with many supporting details. BRDs are based on a far-reaching future that spans multiple product development phases and releases. By design, a single BRD may represent an implementation effort that takes anywhere from a few to many months.

Historically, BRDs have been used in sequential/phase-based product development where thorough upfront analysis and documentation would be followed by comprehensive system design and architecture, then followed by componentized (not feature-centric) coding/programming, further followed by QA, and finally by UAT. In this process, commonly known as waterfall, BRDs, once created, are passed down a chain from one organizational layer to another for sign offs and approvals.

As such, BRDs represent a formal contract between different organizational units, within the same organization structure, primarily between customers and technologists. Such contracts (or contractual agreements) typically contain hard assumptions about "WHAT" must be delivered, "WHEN" it must be delivered and "HOW MUCH" a delivery will cost.

Note: This post is not about what the role of the business analyst becomes in Agile software development or what type of contribution the BA can make to Scrum teamwork. A separate dialogue would be required to elaborate on this topic.

General shortcoming with BRDs

BRDs imply conclusiveness

Creating a comprehensive spec for software development implies that all best solutions are known upfront. This also assumes that throughout a development cycle the customers should not change their minds about what they want and should not come up with better, brighter ideas. Any changes usually trigger what is known as a "change request process" that is a standard protocol to deal with scope creep. During this process, business analysts or other individuals in similar capacity produce voluminous documentation, usually in a vacuum, finalizing requirements without direct input from technology.

And rather frequently, the best ideas and solutions come much later in the process when development phase is well underway. It is also not uncommon that customers change their minds about initially stated requirements after development begins. In cases like these, to justify BRD scope creep, a tedious and overly bureaucratic process of change control is implemented—something that requires additional time and effort. By design, BRDs are meant to resist changes; anything that requires an update after BRD is finalized and signed off carries a negative connotation. Lastly, having BRDs produced without initial participation of technology creates a lot of wishful thinking and unrealistic expectations from customers that sometimes look for complex and expensive solutions.

BRDs are produced by "middlemen"

Typically, BRDs are produced by business analysts (BAs)—a group of people that represents a distinct functional layer, controlled by its own line management. Since BAs are not true customers, they may not have the best understanding of real business needs. Neither are BAs software developers, and therefore they may not know all nuances and specifics of technical options, dependencies or constraints. Effectively, BAs are "middlemen" or translators that reside between business and technology, sometimes being imbedded on either side of the organization yet still representing a separate functional layer. And while frequently BAs are imbedded on either side of the organization (customers or IT), they still represent a separate functional layer. Such additional translation layer frequently leads to miscommunication, misunderstanding, lengthening of communication feedback loops and increasing work cycle time.

Contractual behaviors caused by BRDs

By design, BRDs represent a contract between Business and Technology. This unofficial internal contract between different parts of the same organization splits it the oorganization) into "us" and "them." Each organizational layer is focused on its own commitments, frequently losing a view of a larger picture and collective/shared strategic goals. Frequently driven by monetary incentives and other discretionary perks (e.g., next year hiring budget, performance driven bonus), every organizational layer attempts to do their best in delivering "their own piece" and then passing it on to another organizational layer that then becomes responsible for signing off on a receivable and producing their own deliverable that must, in turn, become a continuity of upstream receivable.

Producing BRDs leads to local optimization

Because each organizational layer is only focused on its own deliverable, it (thenlayer) optimizes their internal process and dynamics only for its own, local success. This is called "local optimization." In the case of BRD writing, it is frequently seen that a heavy document is produced way ahead of development beginning. This is how an organizational layer that is responsible for producing documentation (e.g., business analysts), claims completion and their "local" success: "BRD is finalized and signed off." For this to happen, an organizational layer that is solely in pursuit of completing its own local goals expands disproportionally by hiring more and more people to meet strict deadlines for BRD (in this case) creation. After documentation is signed off and consumed by another downstream organizational layer (e.g., architects, designers, developers, testers), a layer responsible for heavy documentation (e.g., BAs), becomes underutilized (idle) again. While being significantly expanded due to prior excessive hiring and to look busy and justify its existence, this layer begins focusing on work (additional documentation, analysis) that has low business priority: a layer "locally

optimizes" itself to justify its size and activities. If it does not, it would have to decrease itself by off boarding its own people. This phenomena can be qualified as organizational waste (Lean term: "muda").

Are there exceptions to general rules?

But just to be fair, there might be some exceptions where more comprehensive documentation done upfront would be justifiable. Such exceptions are rather rare, but for the sake of argument, here are some examples:

- Organization has signed an external agreement with a third party (vendor), where statement of work (SOW) is clearly defined and legal contracts define a relationship. In cases like these, where work will be competed in sequential (waterfall) fashion, having a more detailed documentation might be necessary to avoid liability on both sides. Here, producing a lightweight requirement document could be acceptable. However, it is still not advisable to lock everything upfront: scope, budget and timeline. Most likely, scope and budget will become fixed (as oppose to Agile development, where scope usually remains flexible), but if so, then timeline should remain flexible.
- Organization/group/department takes on a limited-scope initiative that is expected to be complete within very short time frames. In cases like these, risks of going off track and encountering classic challenges of waterfall lifecycle are significantly lower.
- Organization wants to decommission a legacy system and substitute it with another system (newer technology), and while preserving/replicating most of existing functionalities, there is no new development expected (prioritization is not required and time frames are not rigid). Unrealistic, but possible. Here documentation is used to capture findings made during reverse engineering of a legacy system.

Observed anomalies with the use of BRDs in Agile software development

Cases are known when teams and their product owners have attempted to retain BRDs for Agile development (Scrum or Kanban). Two main cases are worth describing where this mistake becomes costly:

A team attempts sprinting and iteratively developing based on its "understanding of an entire BRD."

Given the fact that BRD is an all-inclusive, monolithic document that usually covers work of many weeks or months, decomposing it into smaller, independent requirements becomes very challenging. As a result, since requirements are not independent, a team cannot safely estimate and commit to a small, independent scope of work to be done in a short time frame (sprint). Needless to say, that product owner is reluctant to prioritize parts of BRD that cannot be decoupled from one another. Subsequently, a team ends up committing and executing on randomly selected parts of BRD that describe

functionalities with upstream and downstream dependencies but cannot be classified as minimal marketable features (MMF) or potentially shippable product increments (PSPI) that a team can deliver independently as vertical, cross-componentized slices. As a result, product owner and customers do not get to see a working product at the end of each sprint. This further creates challenges for a team with delivering independent, vertically sliced, cross-componentized features (a.k.a., minimal marketable features, or MMF) at the end of each sprint: Product owner and customers do not get to see a working product at the end of each sprint. Finally, negotiations and trade-offs between a team and product owner halt when delivering against BRD, as BRD represents all-or-none contractual agreement: a team ends up "faking" incremental development. What is frequently seen is that teams spend many initial sprints working on individual system components (e.g., database-only, or mid-tier only, or UI-only), with later sprints dedicated to massive integration and bug fixing. This translates into a team's inability to deliver anything complete ("Done-Done") to product owner and customers in any given sprint. Lack of cross-component, feature-centric work inside a sprint also discourages knowledge sharing and collaboration among team members.

Product owner shields a team from BRDs while continuing to use BRDs with end customers

Here, product owner, along with a team, attempts to embrace principles of incremental development. As such, a backlog is maintained to support communication between PO and a team. This includes continuous backlog grooming, work prioritization and sprint commitments. End-of-sprint showcases attempt to deliver small, independent, cross-component-cutting, potentially releasable chunks of functionalities. This is all great, however…

However, on the other side of the communication chain, between product owner and customers, communication still takes place in the form of BRDs. What does this really mean? It means that product owner continues to live in the waterfall world with his customers by accepting requirements from them in the form of heavy static, conclusive documentation and giving them (customers) hard, all-or-none, non-negotiable commitments. At the same time, product owner still attempts to prioritize and negotiate work with delivery teams. Product owner spends a lot of time and effort trying to decompose heavy BRDs into small independent chunks of work (user stories) to feed a backlog. This is usually done in a silo without a team's involvement (to "spare" their time) which creates many wrong assumptions and false expectations. It also makes the product owner to work a double-shift.

While on the surface this willingness to shield a team and "absorb heat" may seem to be as a noble and courageous act by PO, at a deeper level this approach is wasteful and even

dangerous. Effectively, product owner makes commitments and gives promises on behalf of delivery teams. PO creates an illusion for technology that customers appreciate transparency, are ready to accept work incrementally and, if necessary, willing to re-negotiate scope and/or "trim a tail" of less critical work. PO also creates an illusion for customers that technology will deliver an entire requirement (as per BRD) by a certain date, and on budget. There is a set of false expectations on both sides because of miscommunication created by product owner. This problem may not be too obvious at the beginning of development cycle but does become very apparent as a release date approaches. This dysfunction is usually accompanied by inaccurate reporting: Agile metrics (such as velocity and release forecasting that are used in communication between a team and product owner) get completely out of sync with RAGs that are used in communication between product owner and customers. Further, this translates in diminishing credibility of technology in the eyes of customers and deteriorated relationship between technology and product owner, as well as product owner and customers. At the end of the day, nobody wins.

A much more effective and safe method to bring customers and technology closer together, while keeping product owner engaged in his critical role, is shared education. It is unreasonable to expect that by educating product owner(s) and team(s) only on Agile principles, norms and frameworks, an organization can solve the problem of miscommunication and distrust (not uncommon!) between business and technology. The rest of an organization, beyond product owner, cannot remain on the sidelines; they also must be educated on core values and principles of Agile product development.

While the key role of product owner remains to be a voice of the customer, product owner should not put himself in an uncomfortable position of making competing commitments and giving unrealistic promises that will cause harm to other parties involved.

The Fallacy of Red, Amber, Green Reporting

Originally published on January 21, 2016 | Location: https://www.projectmanagement.com/articles/315648/The-Fallacy-of-Red--Amber--Green-Reporting

Red	Amber (Yellow)	Green
• Serious over-budgeting • Serious delays against critical milestones • Unacceptable lack of expertize or shortage of resources • Very long delays, unresolvable dependencies and bottlenecks • Low product quality that makes a product unacceptable • Very unhappy stakeholders	• Significant over-budgeting • Significant delays against critical milestones • Lack of expertize or resources • Noticeable bottle necks; Significantly longer delays and cycle time • Problems with product quality that impacts reputation • Unsatisfied stakeholders	• On budget • On time • Expertise and resources are available • No bottlenecks or handover delays • High product quality • Happy stakeholders

Rarely do we see projects where the categories above are meaningfully parameterized by using numbers. But even in situations when they are, numerical values representing each category are not comparable, so the problem of accuracy remains.

Lack of accuracy and reliability

One of the key issues with Red, Amber, Green (RAG) project status reporting is that they represent a *point-in-time* situation. Frequently collected from different functional groups—such as business analysis, design, architecture, development and testing--statuses get collated and someone (usually the first-level project manager) assigns them a color grading.

Between the time initial data is collected and the time RAG reporting is produced, there is usually lag time of at least a couple of days, sometimes weeks. This means that by the first time RAG reporting is presented at a project status meeting, it is already outdated. By the time status reports get produced across multiple projects and combined for senior-level status reporting, more time lapses and this makes reporting even more stale.

Another issue with RAG reporting is that usually, even at the lowest point of collation (first-level project management), there is a degree of variability that is being introduced into the system. First-level project managers do not always have enough depth and breadth of subject matter expertise to allow them to objectively collate reports from multiple functional areas into one all-inclusive RAG report.

Very rarely do we see a number-to-color scheme that allows for subjective conversion. Although there are many technology teams that use more objective engineering indicators to report on project development health (e.g., SONAR) to monitor code quality (test coverage, duplicate lines of code, broken rulesets, number/priority of bugs, etc.) when this information is combined with less objective indicators (e.g., BRD or Business Requirement Document, level of completion, number of change requests created, fast approaching deadlines), the overall result is less than objective. We have an effect of adding "apples to oranges" when non-comparable datasets are added up and are further combined and then translated into colors.

What presents even more ambiguity are the threshold points, where green becomes amber and amber becomes red. Since there is no reliable numerical scale that is based on scientific evidence and historical data, decisions about project color changing are purely arbitrary, subjective and guessed.

Frequently, we hear questions coming from senior management and cascaded down to teams: "Can we identify path to green?" or "What would it take us to go back to green again?" Effectively, senior management is asking what it would take to put a project back on track. And the same challenge presents itself again, as with escalation: while potential improvement steps can be identified, it is practically impossible to quantify what it would take, for example, to downgrade the project from amber back to green. Decisions are still very subjective.

Blame passing and "contract gaming"

What is the underlying cause of, and problem with, conventional RAG status reporting? And what can it lead to? RAGs are caused by *internal* contractual plans (not to be confused with legal contracts that define relationships with external vendors or third parties) where different functional areas of the same organization hold themselves contractually accountable for "all or none" commitments.

In situations like these, a classic project management triangle is mistakenly fixed at all three angles (time, scope and cost). In their lightweight Large-Scale Scrum (LeSS) framework, Craig Larman and Bas Vodde discuss problems associated with "The Contract Game" (which starts about the thirty-fourth minute on the YouTube video) by revealing that convoluted schema of bonus distributions and promotions leads to system gaming and subjective RAG status reporting.

Effectively, fearful workers and pressured first-level managers hesitate to announce to more senior managers and executive leadership that the project is no longer green. Why? Because individual compensations and promotions are at stake. Naturally, as status reports "roll up" they typically get "fudged" and become less inflaming. By the time top executives get summary reports across multiple projects, programs and portfolios, information is not only outdated but also much rosier than when it came from a source.

It is worth noting that constant fear of reprisals and other adverse consequences caused by yellows and reds also lead to bad, unethical behaviors such as finger pointing and blame shifting. Different functional areas that are involved in internal "contract" negotiation and handovers tend to demand "more, more, more" from their respective "obligors" and provide "less, less, less" to their respective "obligees."

As everyone who is involved in a convoluted RAG reporting schema has something to lose in case a project shifts away from green, information gets sanitized and subjectively "improved" at every escalation level. It is not uncommon to see a few amber and a few green statuses meet at a higher level and the outcome being unjustifiably green.

At every level, attempts are being made to keep status reporting within "green boundaries" or as close to green as possible. This goes on until some short time before the last possible deadline when a customer no longer wishes to wait and demands a product. This is when we observe an embarrassingly abrupt color change: green, green, green, green, red (sometimes even skipping amber). It is at this point when all parties start making monumental efforts to return a project back on track (back to green). Usually this is done without the customer, but it is not uncommon that an internal customer is a part of the conspiracy. Why? Because their bonuses and promotions are also at stake.

It is here where a project management iron triangle starts to experience unbearable tension at all three corners. But since all three corners are locked, it is usually Q (quality) that gives way. Larman refers to this phenomena as "Secret Toolbox": spaghetti code, lack of testing, numerous bugs, etc.

Conclusion

The ways RAG statuses are being used today by most project managers are unreliable and misleading. This claim is based on the following:

- Project managers don't truly control project progress, and their project progress indicators do not have an objective numerical scale. Reporting is subjective and immaterial.
- Project managers are under constant pressure and scrutiny of more senior managers that are positioned even further away from action. This pressure is passed on to individuals and teams that perform work and invent reporting data containing flaws.

Today, RAG status reporting is analogous to "Chinese telephone," which is also accompanied by an element of fear of reprisal from above and unhealthy motivation to present things better than they are as they roll from the bottom to the top.

Other Options

Below are some simple alternatives for how to handle issues associated with RAGs:

1. Complete abandonment of RAGs: Abandon RAGs altogether. Lack of reliability, bad behaviors and the systemic harm that they cause in most of cases is apparent. Why continue causing a self-inflicted wound?

Even when produced without fudging and data-alteration, RAG reports are still subjective and provide a false sense of comfort. Why not put a stop to what has outlived itself? Instead of reporting on "point in time" colors, teams and managers at all levels should educate themselves and start using more incremental/gradual status reporting tools.

Such tools are frequently used by product development teams that use Agile frameworks. Instead of communicating point-in-time outdated statuses, Agile teams/projects are able to seamlessly produce reports that are based on reliable, empirical and up-to-date data. Such reports include (but are not limited to): sprint or release burn-up or burn-down charts, cumulative flow diagrams (CFD), epic burn-up or burn-down charts and other Agile metrics.

What is left is to collate and meaningfully roll up relevant data across multiple streams of work. This can be seamlessly done by using various scaling techniques and a variety of dynamic Agile tools. This approach also removes the need for subjective interpretation and escalation of data equally and fairly across all organizational levels. It also minimizes manual labor and therefore reduces systemic waste.

2. Tight coupling of color coding with numerical data: But what if at certain high organizational levels, senior management does not have time to review raw metrics and interpret data from multiple sources? What if, indeed, at certain levels of organization structure, reporting of "we are green" versus "we are amber" is still preferred?

If RAG reporting is inevitable, organizations need to introduce an intuitive and objective numerical scale that represents each color, emphasizing its upper and lower boundaries. This should not be done subjectively by hands-off people, but rather by individuals that intimately understand the true impact of a measurable parameter.

Agile Leadership

What Should Agile Management Care About?

Originally published on September 18, 2017 | Location: http://www.keystepstosuccess.com/2017/09/what-should-agile-leadership-care-about/

Agile frameworks (e.g., Scrum, Kanban, XP), individuals' roles and responsibilities, processes and tools, metrics and reporting, burn-up charts, estimation techniques, backlog prioritization, Agile engineering practices, Agile maturity models, etc., are all important attributes of a typical Agile transformation. However, NONE of them are first-degree-of-importance system variables that are responsible for transformation success. Most of them are good superficial lagging indicators of Agility but they are all corollary (secondary and tertiary) to another much more important system variable.

What is the most important system variable that defines a company's agility? It is **Organizational Design**—the most deeply rooted element of organizational ecosystem that defines most of the system dynamics.

When organizational leadership decides to take an organization through an Agile transformation journey (it could take years, sometimes), it (leadership) needs to acknowledge that real, sustainable Agile changes are only possible if deep, systemic organizational improvements are being made. For that, leadership needs to be prepared to provide to its organization much more than just *support in spirit*, accompanied organizational messages of encouragement and statements of vision. Leadership must be prepared to intimately engage with the rest of an organization by doing a lot of real "gemba" (genchi genbutsu [現地現物]) and change/challenge things that for decades, and sometimes for centuries, have been treated as *de-facto*.

What does it really mean for leadership to engage at System Level? First, it is important to identify what a system is: what are a system's outer boundaries? For example, one of the most commonly seen mistakes that companies make when they decide on "scope of Agile transformation" is limiting its efforts to a stand-alone organizational vertical, e.g., Technology—and just focusing there. Although this could bring a lot of *local* success(to IT), it may also create unforeseen and undesirable friction between the part of an organization that has decided to change (IT) and the part of an organization that decided to remain "as is" (e.g., Operations, Marketing). For example, if Scrum teams successfully adopt CI/CD, TDD or other effective engineering practices that enable them to deliver PSPI at the end of every sprint, but business is not able to keep up with consumption of deliverables (too many approvals, sign offs, red tape), then the whole purpose of delivering early and often gets defeated. Then, instead of delivering to customers soon, in exchange for timely feedback, teams end up delivering in large batches and too far apart on a time scale.

A successful Agile leader must treat an organization that is expected to transform as a sushi roll. Just like seaweed alone does not provide a full spectrum of flavors and does not represent a complete, healthy meal, one single department (e.g., IT) is not sufficient enough to participate in Agile transformation efforts. Other organizational layers need to be included as well when identifying a slice for an Agile transformation experiment. A slice does not have be too thick. In fact, if an organizational slice is too thick, it might be too big to "swallow and digest." But still, even when sliced thinly, an organization must include enough layers, to be considered as a "complete meal."

Note: A great example of treating an organization as a sushi role, while making it more Agile, is Large Scale Scrum (LeSS) adoption.

So, what are some key focus areas that every Agile leader must keep in mind while setting an organization on an Agile transformation course?

- Location strategies. Geographic locations.
- HR policies (e.g., career growth opportunities, compensation, promotions).
- Budgeting and Finance.
- Intra-departmental internal boundaries and spheres of influence.
- Organizational leadership style.
- And some other areas that historically have been considered as *untouchable*.

All the above listed areas are defined by Organizational Design and can be better understood through self-assessment, done by organizational leaders at all levels.

Be an Educated Consumer

Originally published on January 21, 2016 | Location: https://www.scrumalliance.org/community/articles/2014/july/be-an-educated-consumer

You just bought a house and decided to renovate. You brought in a contractor to estimate the work that you want done. What is your biggest fear? Here are a few possibilities:

- The work will not be done.
- The work will not be done on time (winter is coming and you need wall insulation before it gets cold).
- The work will be done on time but it will cost you much more than you can afford and more than this job really costs. The contractor is charging you more than he should; he is taking advantage of your ignorance.
- The quality of the work will be poor but, unfortunately, you will not know this until all of it is finished and you move in (after you pay in full).

Why is this happening? Because you lack expertise as a consumer. Because you don't know the market you are in. Because you don't know how to look for the warning signs of an agreement gone bad. You are not equipped with knowledge. You are not an educated consumer.

You must be an educated consumer. From the moment a contractor walks in your house, he must know that he is dealing with someone who knows what he wants, when he wants it, and for how much. A contractor must know that you will be able to tell an economically sound, timely job of high quality. You are also going to be very much benefited if you work out a deal with the contractor to have the work done incrementally, as well as paying for it incrementally.

Agreed, many contractors will not work this way, but there are some who will. Look for those. All-or-none, one-big-bang deliveries are not good anywhere, including in construction.

When it comes to software development, being an educated product owner is very important. Not only should you be able to validate the work quality that gets delivered to you, but you should also be able to understand and speak to other things that happen before the work gets done. Learn the language that teams use to communicate with each other and you when they discuss their progress on sprints and releases. Learn the jargon, speak the jargon. Learn Agile collaboration tools, learn Agile reporting, and learn the meaning of various Agile metrics, tools, and techniques.

This will help you, as a product owner, intelligently discuss with your teams the sprint velocity and release scope, monitor gradual delivery of work (or the lack of such), gauge the accuracy of forecasting, and understand the importance of strategic planning.

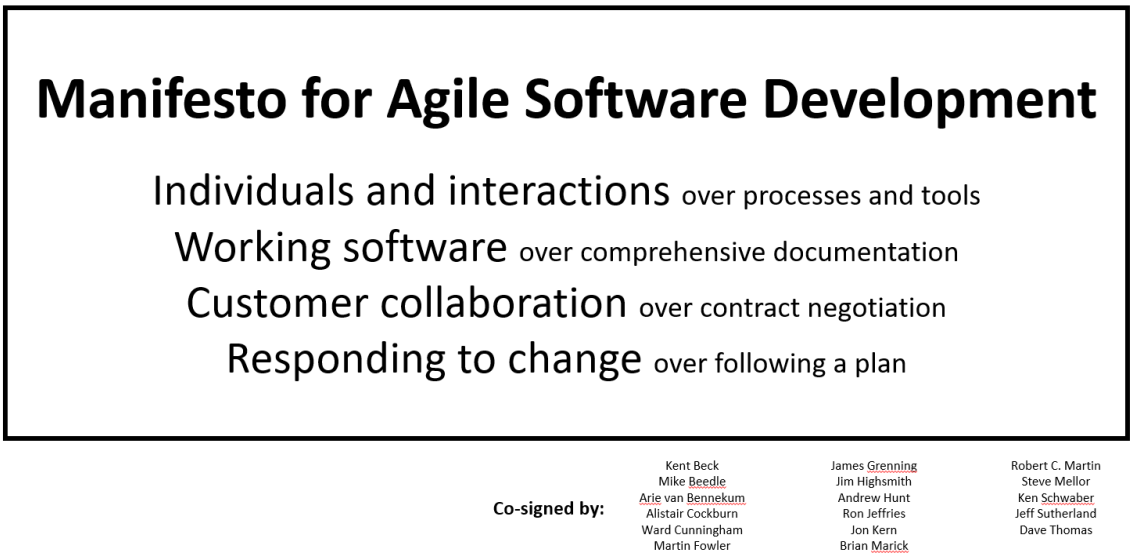
Gene Gendel, CEC-CTC, LSFT, CAL, CLP, CS@S | www.keystepstosuccess.com

Collaboration is always much more effective when both parties take each other's knowledge and expertise seriously and with respect, which is possible when each party understands that another party cannot be fooled by unrealistic promises and unfair deals. If you are consuming a product or service, you must be an educated consumer—this will both make your life easier and protect your relationships with service and product providers.

Extending Agile Manifesto

Originally published on August 9, 2016 | Location: http://www.keystepstosuccess.com/2016/08/extending-agile-manifesto/

In February of 2001, in Snowbird, Utah, a group of 17 bright software developers got together to discuss software development methods that were lightweight and easy to implement. The term that they decided to use to describe these methods was "**agile**" and the four main postulates that they all agreed to became known as the "**Agile Manifesto**."



Source: www.agilemanifesto.org

But just like anything in Agile, continuous improvement and inspection and adoption prevail. Therefore, while the four core values of the original Agile Manifesto still represent a foundation, the Manifesto can be extended to other supportive values.

Here are some examples:

- **Being Agile** over "doing" Agile.
- **Servant leadership** over command and control.
- **Feature teams** over component teams.
- **Component mentors** over component owners.
- **T-shaped people** over single specialty workers.
- **Monitoring work in progress (WIP)** over managing workers.
- **Shorter development cycles** over longer development cycles.
- **Automated testing** over manual testing.
- **Team performance** over individual performance.
- **Higher base salaries** over subjective discretionary bonuses.
- **Continuous improvement** over "best practices."
- **Real Agile coaches** over fake Agile "experts."

- **Funding business cycles** over budgeting calendar years.
- **Dynamic forecasting** over "Dec 31 end-of-world" forecasting.
- **Team collocation** over geo-distribution and location "strategies."
- **S.M.A.R.T. + E.R. (Ethical, Reasonable) goals** over S.M.A.R.T. goals.
- **Scorecard translation** over scorecard cascading (top to bottom).
- **Relying on what counts** over relying on what is easily countable.

Unspoken Agile Topics

Originally published on 18 July 2014| Location: https://www.scrumalliance.org/community/articles/2014/july/unspoken-agile-topics

> *"Observe always that everything is the result of change, and get used to thinking that there is nothing Nature loves so well as to change existing forms and make new ones like them."*
> *– Marcus Aurelius*

Introduction

This paper, originally written in February 2013, brings to light some of the least-discussed topics and consequences of "broadband agilization" that currently take place in the industry. The materials of this paper are subdivided into two general sections:

- The first section describes certain impacts that Agile has on individuals and their personal career advancements.
- The second section describes organizational-level Agile impacts that pertain more to client companies that undergo Agile transformation, as well as service-providing vendor companies that deliver Agile-transforming expertise to their respective clients.

The reader will most likely focus on the section that best represents his primary interests and concerns. However, it is recommended that both sections are read in full—in unison they create a better holistic perspective of the industry changes brought about by Agile-mania. The reader will be taken out of his comfort zone and forced to think more uninhibitedly and realistically about those aspects of Agile that may not be as obvious and are not explicitly covered in other literature.

Organizational impact of Agile

Peer pressure

> *"Change does not necessarily assure progress, but progress implacably requires change. Education is essential to change, for education creates both new wants and the ability to satisfy them."*
> *– Henry Steele Commager*

Problem statement

According to Wikipedia (http://en.wikipedia.org/wiki/Peer_pressure), *peer pressure* is defined as "influence that a peer group . . . exerts that encourages others to change their attitudes, values, or behaviors to conform the group norms."

Today companies often decide to introduce Agile practices without thoroughly thinking through why they are doing it, without doing enough due diligence and research to reasonably attest that, indeed, their efforts will bring benefits in the long run. Instead, these companies do so because their executive management has decided that "the time has come," since there are so many other peers out there that do the same.

For such companies, Agile adoption has almost taken the form of a fashion statement, a way to prove to themselves that they are up to date with others. Such companies care more about keeping up with the mainstream than making a well thought-through and carefully planned step forward of bringing better values, practices, behaviors, and cultural patterns inside their walls.

There is whole array of commonsense Agile transformation readiness prerequisites that these companies either ignorantly overlook or intentionally ignore. These companies go after what may seem to be a very good cause. However, it is nothing more but a chase of the "status quo."

In the majority of cases, such poorly substantiated motives for Agile adoption bring about failure, and since there is only one chance to make a first impression, any future attempts to reintroduce Agile at a later point, even when conditions might become more favorable, usually meet resistance and very little support. Trust is lost as nobody wants to repeat the same mistakes twice.

Discussion

"Going Agile" should never be a final goal. Agile is just a way to get to something much more measurable and tangible. For example, achieving near-term and long-term economic benefits, ensuring cost-effectiveness and higher return on investments (ROI), adjusting corporate cultures and working environments in ways that make companies more desirable places to work.

Many firms proudly announce that they have been undergoing Agile transformation without accepting the fact that in order to truly appreciate the benefits of Agile tools and techniques as the mechanism for more effective product development, firms also must adopt agility in their structure and culture. The former is just not possible without the latter. Specifically, if an organization does not have in its plans to fundamentally adjust its conventional hierarchical structure, flatten its convoluted reporting lines, remove redundant roles that do not really contribute much value to the overall process, and trim wasteful activities, then there will be no cultural shift. While using the Toyota Production System (TPS) as an example in their studies of Lean, Tom and Mary Poppendieck have clearly identified seven types of waste in Agile product development.[3] The third item on their list of wasteful activities is "extra processing." If we think about any process in terms of how many people are involved in it, then it begs the question:

Should removal of wasteful processing also remove wasteful processor(s)?

Just as an assembly line automation makes a lot of manual labor unnecessary, eliminating unnecessary steps in a process makes the performers responsible for those steps also unnecessary. This supports the claim that to successfully transition to Agile, companies should be willing and ready to reorganize their cohorts in ways that require trimming off what is no longer needed.

In his writings, Craig Larman[5] alludes to organizational waste management by referencing not just wasteful and counterproductive processes and individual behavioral patterns but also certain organizational layers and individuals that cause retardation of the Agile process.

While being a big proponent of flattening organizations in general, he explicitly refers to certain mid-level management that should be removed due to their lack of value.

There are some other pitfalls that are more frequently seen with larger, enterprise-level organizations:

- Inability to perform cost/benefit analysis to determine whether the adoption of an Agile framework is monetarily beneficial for a company. Since larger companies typically do not release to production as frequently as smaller ones, it is not always easy to map end-client satisfaction and demand and subsequent revenue increase to changes of product development approaches. This delayed cause-and-effect is due to longer time to market and prevents executive management from seeing results of Agile transformation soon enough to decide whether it is worth continuing the experiment. Executives do not use a "stop the ship" approach to objectively analyze what has been accomplished to decide whether it makes sense to continue. If things do go south, by the time executives realize this, the damage is too high to be dealt

with silently, behind the scenes. When the political current gets too strong, as it does in majority of cases, fighting it and acknowledging that the latest and greatest add-on to a company's strategy is not working is not something that the top echelon of executives can do easily.

- Inability to consider existing business relationships with third parties whose operational models, processes, and functions adversely impact Agile adoption. Specifically, dependencies on third-party product development vendors who do not practice Agile, do not deliver incrementally, and—what is even more alarming—fundamentally are not equipped for transparent two-way communication (preferring everything in writing, sealed and signed) is very costly. This can negatively impact a company's ability to produce its own deliverables. Such relationships and dependencies must be thoroughly reevaluated and, if necessary, terminated.
- Inability to develop standardized techniques or mechanisms to measure levels of Agile maturity across multiple organizational verticals. Although developing universal best practices for multiple teams, even within the same organization, is *not* expected (also contradicting the idea of decentralized control and frequent inspection and adaptation—something that is required by Scrum), the ability to tie low-level (local, single-team) Agile metrics to global performance indicators is possible and even desirable, as ultimately every company measures its profit and loss by using universal units (currency).

Instead of attempting to do "big bang," top-down Agile transformations of multiple teams, projects, and programs at the same time, it is much more advisable for companies to start small, to consider a pilot Agile project to gain small, quick wins, and then gradually proceed with Agile adoption by sharing knowledge and lessons learned laterally: from team to team, from project to project. It is important, however, that executive support and buy-in come all the way from the top-executive level—what is inevitably required for a lasting cultural shift.

Executive-level coaching is required for this to be a success.

Numbers do lie

> *"Nothing in education is so astonishing as the amount of ignorance it accumulates in the form of inert facts."*
>
> *– Henry Brooks Adams*

Problem statement

In most of reference literature, the word "metric" is defined as the system of standard or measurement. In the conventional world, the notion of metrics analysis is frequently

associated with establishing success/failure criteria, progress indicators, and benchmarks. Although the ability to properly analyze and communicate Agile metrics data still serves its meaningful purpose for Agile teams as it gauges them in their journey toward maturity, for large-scale Agile transformations at enterprise level (no so much small- and mid-size organizations), the improper use of Agile numbers is common and dangerous. The ability to incorporate Agile data into a broader picture and integrate it with other enterprise-level measurement tools, techniques, and analytical facts is frequently lacking.

Discussion

If Agile transformation is driven from the top of an organization but appropriate training is not timely provided to executives, then their expectations are not properly set, and this leads to misjudgment and poor decisions.

The problems vary and include constant pressure on workers and deterioration of morale, mistakenly (in a rush) selected corrective actions, and/or making things "look pretty" by falsely communicating unachieved progress further up the chain of command and to the rest of organization in order to preserve personal credibility and reputation. *One of the most common misinterpretations is of metrics obtained from Agile collaboration tools. The words of wisdom in IT are "A fool with a tool is still a fool."*

This wisdom has proven itself many times. One of the most frequently misinterpreted and misused metrical units is velocity—specifically, what it measures and what it depends on.

Here are some most common misjudgments *regarding velocity:*

- Velocity reflects how much time it takes for a team to complete work. No consideration is given to the fact that for complexity estimation to be accurately translated to time, the same team must point stories and stories must come from the same PO who has the same writing style and is able to write "sizably." This is wrong.
- Without prior normalization of story point estimation across multiple teams (deriving a conversion factor to normalize one point of one team against one point of another team), it is OK to compare velocity of one team to that of another team. This is wrong.
- Over time, velocity should continuously increase and if it does not, it indicates that a team has stopped improving. This is wrong.
- A team's velocity can be increased by increasing the number of team members; this is an effective way to increase velocity. This is wrong.
- Resource rotation (on/off a team) to ensure cross-training and knowledge transfer outweighs the importance of keeping a team together and does not have impact on velocity. This is wrong.
- Re-estimating work mid-sprint is acceptable if it can provide a correction to inaccurate estimations at the beginning of a sprint. This is wrong.

- Attributing a certain percentage of a team's committed velocity to an individual team member, based on the contribution of that individual to any story, is an objective way to measure individual productivity/performance and a reliable way to gauge overall team velocity. This is wrong.

This is how some of these false interpretations come about:

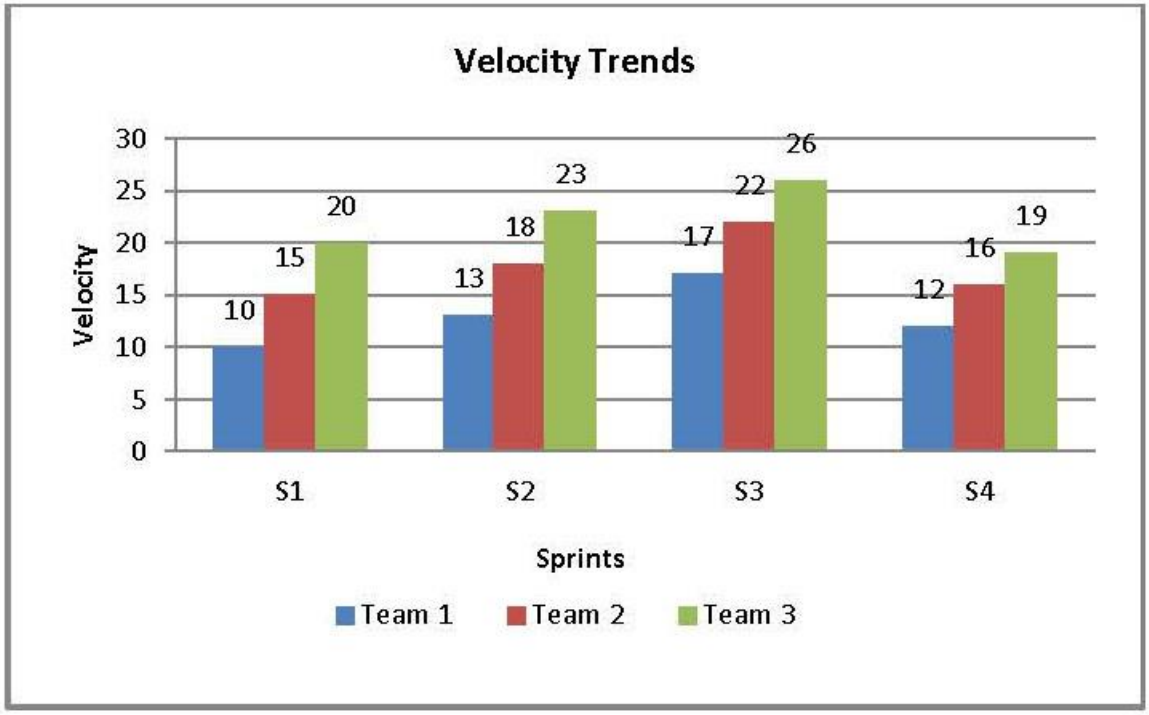


Figure 1: Comparing un-normalized velocity

Figure 1 above illustrates an example of three separate teams being compared against each other in terms of their velocity. Such a comparison does not hold value for the following reasons:

- Team size and, subsequently, its capacity most likely varies. This means that the man-hours of one team are not comparable to the man-hours of another team.
- The estimation scale of each team is different. To compare a story point of one team with that of another, a conversion factor must be derived to understand what each team's story point really means in terms of ideal hours.
- Finally, Agile maturity of teams might be different. Although a team's maturity can be related to a team's productivity/output, comparing teams that are novice to Agile with those that have been in operation for a while and have developed some cadence is not objective.

Here are some graphic illustrations of how such improper judgments originate:

Figure 2 below illustrates an example of a false expectation that, over time, the velocity of each team will indefinitely increase. Team 2 (brown) appears to have the velocity trend that aims at infinity. Team 1 (blue), on the other hand, has reached a plateau and is no longer increasing. It is false to assume that the velocity trend of Team 2 is better than the velocity of Team 1. Yet, frequently, this is exactly the conclusion management derives when they are presented with multi-team velocity trend charts.

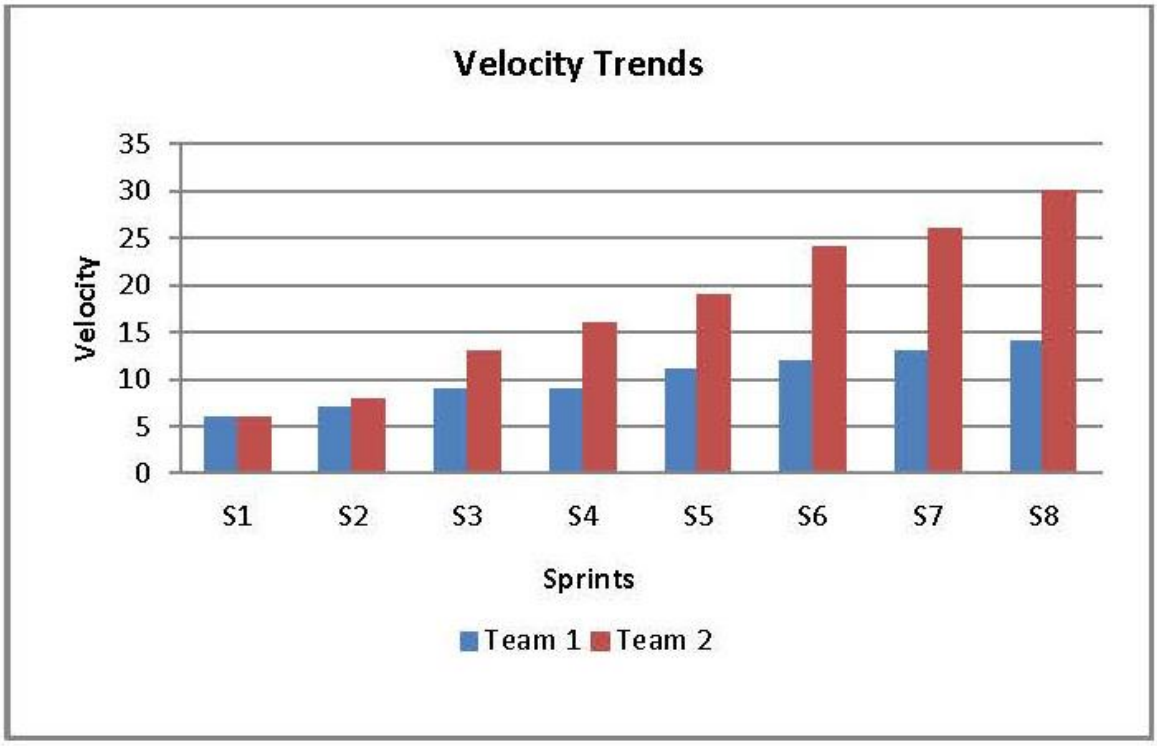


Figure 2: Expecting continued velocity increase

Such ever-growing to supersonic velocity of Team 2 is not realistic. Increasing velocity to a certain reasonable level and then sustaining it would be much more reasonable. Having a steady velocity would also make forecasting and forward planning more reliable.

Figures 3a and 3b below illustrate how a team's sprint velocity can be mistakenly attributed to individual contribution.

	Stories			
	Story A SP	Story B SP	Story C SP	Total
	3	5	8	16
John (% of time on Tasks)	0.45	0.25	0.3	
Jeff (% of time on Tasks)	0.25	0.5	0.25	
Jim (% of time on Tasks)	0.3	0.25	0.45	

Figure 3a: Measuring "individual velocity"

Note: If added diagonally and horizontally, percentages add up to "1" (100%).

Figure 3a shows how three individual team members spend varying amounts of time (in percent) on each individual story (later, accepted). Such work distribution could be very reasonable as everyone may need to contribute differently to each story based on his type of functional expertise and type of work required for each story.

Figure 3b, on the other hand, shows a very unreasonable calculation and, subsequently, a conclusion that is frequently made by conventionally-thinking management, especially if an environment strongly supports the idea of individual performance. According to this figure, each team member is assigned a certain amount (even fractional) of story points, based on his percentage of effort contribution to each user story. The numbers are derived by multiplying individual percentage contribution and number of story points each accepted user story is worth.

For example, John, who performed 0.45 (45%) of work (in hours) against Story A (3 story points), established his own "velocity" of 1.35 story points against Story A.

	Points Per Story by Person			Total "Individual Velocity"
	Story A	Story B	Story C	
	Points =3	Points =5	Points =8	
John (% of time on Tasks)	1.35	1.25	2.4	5
Jeff (% of time on Tasks)	0.75	2.5	2	5.25
Jim (% of time on Tasks)	0.9	1.25	3.6	5.75
"Individual Velocity" per Story	3	5	8	16

Figure 3b (Measuring "individual velocity")

Such an approach is wrong, and here's why:

To achieve higher productivity and cohesiveness, Scrum team members are expected to *swarm*—work collectively on the same story or sometimes even a single technical task to achieve common success. Linking any single piece of a team's work, successful or unsuccessful, to individual contribution and performance is inappropriate and subjective:

it prevents teaming and collaboration as well as makeing team members worry about individual achievements more than about overall team success.

Another reason why the above described calculation is a fallacy is that although each team member cumulatively contributes the same amount of time to sprint work, based on an individual's functional skillset, each person may spend time (capacity) differently against differently pointed stories, which means that the multiplication factor in deriving individual velocity is not constant from the beginning.

Another misuse of the velocity concept is splitting up partially done stories at the end of a sprint to produce "partial" or "conditional" acceptance by the PO. By doing so, higher team velocity gets fabricated.

While breaking the concept that each story must be an independent and deployable piece of functionality with intrinsic business value, such numbers-cooking and story-point chasing will prevent a team from establishing a reliable velocity and doing accurate sprint forecasting and strategic planning.

There are also some other inaccurate interpretations of Agile metrics that, for the most part, have to do with leaders' lack of understanding of the economic principles behind product development. Among others mistakes, the two that are very commonly observed are around capacity utilization and work in progress (WIP).

For example, forcing teams to maximize their capacity utilization by increasing individual and team workload to close to 100 percent, as it is frequently seen in "high-performance" organizations, especially when using offshore resources, is a fallacy.

Such an approach forces teams into working without any slack time and, therefore, deprives teams of any chance to improve processes. By the same token, pushing teams into making aggressive commitments during planning and starting a sprint (e.g., in Scrum) with an initially unrealistic amount of work causes end-of-sprint failures. All of this results in a team's diluted focus, excessive multitasking, making irrational decisions, and ultimately producing code of very low quality.

Expecting high work in progress (WIP) by having executives constantly question teams about the amount of work in flight is yet another fallacy.

By applying such orthodox beliefs that everyone should be preoccupied with their own work at all times and promoting the idea that a high amount of work *in progress* is a sign of high effectiveness and productivity is bogus. This contradicts principles of one piece of work flow, queue size management, and capacity-utilization principles. It also conflicts with the concepts of collaboration and swarming that are strongly supported by cross-functional feature teams.

Some convincing studies about capacity utilization and its effects on productivity were done by Donald Reinertsen.[7, 8] They are worth mentioning here because they can be

tightly coupled with one of the Agile product development tools: Kanban.

Based on Reinertsen's Principle of Part-Time Resources (using part-time resources for high-variability tasks), maximizing the load of key resources with high-priority tasks is dangerous if more high-priority work is expected. Individuals who are highly loaded with high-priority work have low surge capacity, which is extra bandwidth that they can use against newly arrived high-priority work.

What this means for Kanban teams that work on production support issues with different levels of severity is that the lack of surge capacity prevents a team from being able to switch to high-severity issues when they arise.

For example, imagine a Kanban team that has work of L1, L2, and L3 levels of severity moving through the same queue:

If a worker is always fully preoccupied with L3 work, his surge capacity is very limited (literally, to his lunch hour), and this prevents him from picking up any additional L3 work, should such work enter a queue. This is particularly dangerous, especially if other workers that are fully preoccupied with L1 work (though having much higher surge capacity) have insufficient skillsets to handle incoming L3 work. It makes much more sense to optimize individual workload in ways that everyone, especially highly skilled specialists, have enough surge capacity (slack time away from L3 work) to be able to react to suddenly incoming high-priority work.

Challenges with Agile leadership

> *"Divorced from ethics, leadership is reduced to management and politics to mere technique."*
> *– James MacGregor Burns*

Problem statement

Today, the most widely recognized Agile leadership role (above an individual team level) is an *Agile coach*. Typically, Agile coaching is delivered by an external consultant who either engages with a company (client) directly and independently *or* represents a specialized Agile coaching/training firm that deploys him on site. Recently, companies began introducing internal Agile coaching practices by way of attracting external consulting talents and then converting them into full-time employees, and/or by retraining existing company employees to transform them into Agile coaches (native coaches).

In the former case, the following two questions usually come up:

- Does engaging with a reputable Agile coaching consulting firm guarantee a top-notch Agile coach-consultant who will be deployed on site?

- Once engaged, what are the odds that a consulting firm, intentionally or unintentionally, positions itself in such a way that its financial benefits from the engagement become more of its focus than value delivered to a client?

In the latter case, when internal coaches are used to help an organization with Agile transformation, the concerns are somewhere different:

- Internal coaches that have not been exposed to the outside world (other industries, other corporate cultures) tend to have views that are narrowed to only what they have seen at their own companies (structure, culture) and, therefore, their ability to comparatively analyze organizational problems is significantly hindered.
- Internal coaches, native or "naturalized" (defined further below), being full-time employees of an organization, are subject to evaluation, scrutiny, and performance measurements that conflict with what Agile culture needs. Such strict limitations, obviously, make it difficult even for the best coaches to provide (uncensored) reflection of a company.

Discussion

When a company decides to procure an external coach and convert him into an employee (get him naturalized), a coach becomes subjected to the same type of evaluation and scrutiny as a native coach, or for that matter, as any other employee.

What does it mean for a role that historically is meant to be held by an independent, neutral third party?

An internal coach, unlike an external coach, cannot as freely reflect upon a company's ability to help itself by acknowledging its own problems and finding its own ways to resolve them. Now, a coach is *part of* a company and the expectations of him are different. What an internal coach says about his own employer and the conclusions and recommendations a coach gives to his own employer will inevitably influence how an employer treats the coach. A coach's job is discovering/exposing organizational pain points, asking powerful, and at times uncomfortable, questions, as well as giving bold reorganizational recommendations—a coach's ability to do so becomes hostage to his or her fear of becoming the subject of criticism and reprisal.

As coaches become (or remain) part of an organization, they are naturally forced to move away from being servant leaders to becoming more of personal achievers and politically correct commanders and controllers.

Back to the *external* coaching consulting model. One of the clearest representations of the Agile coaching dilemma has been described by Dan Mezick in his book *The Culture Game*. The author vividly paints how important it is to define entry and exit criteria for every coaching engagement, as well as its duration, *before* it begins to avoid excessive

monetary transactions and long-lasting codependency between a client company and an external coach—an indication of unethical coaching. Today, unfortunately, most companies still rely on external Agile coaching expertise blindly without questioning its effectiveness. Large-scale coaching engagements are frequently secured based on personal relationships, where SOWs get signed and monetary transactions take place, *not* where transformation takes place and value gets added.

With external Agile coaching, when Agile transformations get done in one "big bang," and there is suddenly a high surge in Agile coaching demand by a client company, the Agile transformation company runs a fire drill and tries to immediately produce additional coaching staff by turning to the marketplace and procuring independent coaches from the street and then reselling them to the client as "their own." This certainly does not guarantee to a client the good quality of a coaching resource, and it most certainly does not render such resource at a true net cost. (The mark-up for these types of engagements is usually high).

Another scenario is when a coaching role gets occupied by an individual with great coaching skills but very little practical, hands-on experience within Agile. Typically, such situations arise when a person comes from a completely different area of coaching (personal coaching, spiritual coaching, career coaching, life coaching, etc.) and effectively uses his "soft"/people facilitation skills to compensate for very superficial subject matter expertise.

Since Agile is primarily about product development, ideally the person who steps into the Agile coaching role should have some technical or at least semi-technical background. This would help him or her better understand and appreciate product development issues and therefore provide better advice and earn a higher reputation.

Going by the same logic, when a company decides to build its own Agile practice, it is very important that selected individuals do a good share of observing and shadowing more experienced Agile coaches before taking their own initiative. Co-coaching with more experienced peers helps a novice coach not only capitalize on his understanding and practical know-how of Agile mechanics but also adopt proper behavioral patterns of being a servant leader and enabler, not a commander and controller—something that is often seen when a coaching role gets filled by an internal person who was previously an authoritative figure.

Agile impact on individuals

Struggle for personal adaptation

> *"Today a thousand doors of enterprise are open to you, inviting you to useful work. To live at this time is an inestimable privilege, and a sacred*

obligation devolves upon you to make right use of your opportunities. Today is the day in which to attempt and achieve something worthwhile."
– Grenville Kleiser

Problem statement

Agile affects professional careers and personal lives. So, let's pause here for a moment and make an important distinction: Agile was initially introduced to develop *products*. Its purpose was *not* to manage individual projects, as it is incorrectly perceived by those who are familiar only with traditional software development. Nor was it a "plug-in" into any other method or framework that companies use. The purpose of Agile was (and remains) to get a product of the best quality to a client in the shortest time frame, at the lowest cost possible.

For many individuals, Agile is a great way to explore themselves, to reveal and further develop their individual potential. Agile favors innovative thinking, helps merge the gap between creative art and the science of technology, and assists in gaining the freedom of making choices and developing the Kaizen culture.

Since Agile originated primarily as the mechanism for product development, individuals with skills that are required for product development can adapt to Agile relatively quickly.

On the contrary, for those individuals whose skills are not directly related to product development processes, Agile adoption presents a significant challenge. Such individuals cannot effectively contribute to day-to-day activities needed for Lean product development; they don't easily fit into the flatter organizational structure required by Agile, and they cannot adapt to the absence of the command and control environment that prevailed under previous working conditions.

Discussion

Agile is meant for true doers. If we recall Donald Reintersen's[7,8] discussions about product development, the closer an individual is located to the automated production line, the more value he brings to the process. In terms of software development, this might be translated as follows: The closer an individual is to a code base, the more value he brings to product development—and vice versa.

Developers

Let's look at a developer. In an Agile environment, a developer is given many more

opportunities than in a non-Agile environment to explore his intellectual capacity while thinking outside the box and coming up with innovative solutions. No longer does a developer obediently execute against frozen business requirements that were most likely put in silo by a business analyst with minimal or no input from technology and with minimal or no exposure to real end users. In the latter case, by the time a developer starts coding, requirements are most likely stale and out of date. This ultimately leads to change requests and, most likely, to product changes when they are the most expensive to make.

In Agile, a developer has direct communication with a "buyer" (end-client or an empowered proxy, the product owner). A developer now has an opportunity to look at each business requirement (usually a user story) individually and offer a unique, at times innovative, technical solution with a very short feedback loop (response) from a client. In case of a positive feedback loop, a developer is encouraged and happy to claim a small credit for his win and a job well done.

In case of a negative feedback loop, a developer has little damage control to do, as the amount of rework required is usually minimal.

What developers do find challenging, however, as they transition from a more conventional environment to Agile, is that they are not always able to work "on par" with other developers. In Agile (let's take the Scrum framework for example), developers on the same team might be at different levels of seniority and, even more undesirable, unevenly positioned with respect to each other in the same organizational hierarchical structure. This is a problem because the lack of equality among team members prevents them from having effective collaboration.

But all in all, for a skilled developer who is willing to cross-train further and become a true T-shaped person (specialist in one field plus a generalist in one or more other fields), Agile is a land of great opportunities for personal and professional growth.

QA

The situation is even more straightforward with QA. Let's make a note here about one very important precondition for scalable Agile: automated testing coverage.

If manual testing still predominates over automation, development will stall and plateau as soon as manual QA is not able to keep up with development. Ideally, test automation should begin with the first line of code or, even better, it should come before coding begins (e.g., TDD/ATDD).

Success in Agile is not possible without test automation.

In Agile, QA involvement begins much sooner than in Waterfall, where QA gets to see a product only when development is practically done and when the discovery and repair of

bugs is the most expensive. In Agile, QA's role is *elevated* significantly, becoming a much more reputable role.

We hear at times this discouraging notion: "A QA person is not good enough to become a true developer." This is no longer valid in Agile because the QA person is now rightfully considered a member of the development team and not just someone who manually executes only by following a handwritten test case.

There is one big assumption, however: that a QA person is willing and able to become and/or remain technical. Any automation test tool requires some coding skills, and since true Agile cannot exist without test automation, the QA person should be comfortable with coding. There may be no need to become a full match to a senior programmer, but QA does need to come closer to a developer in terms of technical savviness. This might present a challenge to those QA people who have done manual testing only. In Agile, manual testing is only temporarily valuable during initial sprints when code base is limited and there are not too many features to test. Over time, as more code gets produced and more functionalities become available, manual end-to-end testing cannot keep up with development.

Therefore, we need a machine.

Like with developers, the ability to work "on par" with other team members may present a challenge. Here, the adjustment for QA people is more psychological and cultural than functional—they must be at the same level with other team members, regardless of their current position in the organizational tree.

Overall, if we assume that cultural adjustment does not present a serious challenge for a technically predisposed QA person, Agile also presents an array of opportunities in terms of gaining more hands-on knowledge, professional respect, and team recognition.

BA

The role of the business analyst is clearly articulated on the www.iiba.org website. According to the International Institute of Business Analysis, the role of the BA includes various types of analysis: systems, requirements, data, process, business intelligence.

Since the discussion of this paper focuses on Agile, and given the fact that the main intention of Agile is to improve product development practices, our focus here is on BAs who participate in product development and serve the purpose of a primary conduit between business and technology.

In Agile product development the intent is to bring closer the business community (end-users) and technology (feature teams). If BAs want to stay close to the product development process and survive organizational flattening, they must position themselves in one of the following ways:

Assume the role of product owner

- Assume the role of product owner-proxy (if such a layer is justified)
- Embed with a feature team as team BA

For a BA to be able to assume a product ownership role, his level of authority and executive decision-making power must be significantly increased. Today, in conventional settings, even very senior BAs cannot make final business decisions on their own, as they must seek approval of more senior staff, including business stakeholders, sponsors, etc. Even in instances when BAs can formulate decisions based on the inputs of many, the cycle time from the moment the BA raises a question to the moment he is able to derive a conclusive decision and communicate it to IT is way too long to support an Agile development pace that is based on short cycle times.

Looking at the situation realistically, it is highly unlikely that the BA will get empowered to a level that he could be rightfully considered the PO. For this to happen, most likely, the BA would have to jump a few hierarchical levels, something that does not happen frequently. Such BA empowerment is even less expected at large, enterprise-size companies than at small/mid-size companies, as reorganizational decisions take place much more quickly and a flatter structure is more natural in the latter. Therefore, although having a BA become a PO is possible, at most enterprise-level companies it is unlikely.

What is much more likely to happen is to have a BA assume the role of PO-proxy—a role that is sometimes introduced to help the PO with his responsibilities. Introducing the PO-proxy role is much more common at large, enterprise-size companies than at smaller ones where a company's primary line of business is software development. You may ask why. The answer is simple: A company that primarily generates its revenue from building and selling software to external clients cannot afford to have a multilayered product ownership structure. The risk of miscommunication and delayed response is just way too high. The PO role is taken much more seriously at smaller/mid-size software development companies. It is a full-time job and whoever takes it gives it his full focus.

At larger companies, however, having a BA or BA-like person assuming a PO-proxy role is much more common; larger companies are more tolerant of having PO-proxy roles. Meanwhile, the role of actual PO (head PO or chief PO) is given to a person with more stripes on his shoulders.

Here is a typical scenario:

The PO role is given to someone who is positioned high in the organizational food chain—someone who is entitled to make final business decisions. But in most of cases, such a PO is more of a political figure who neither fully understands nor is being held accountable for performing his duties as PO. He is not so much a decision maker as a decision "signer," whereas decisions are recommended by someone else. This "someone" is typically a PO-proxy, a lower-ranking role that is almost immediately introduced after selecting the PO. The PO-proxy role becomes a buffer layer between the PO and real work.

Although there are instances in which the model of a single PO plus multiple PO-proxies makes sense (e.g., the PO is responsible for a product that is being built by multiple feature teams, where each PO-proxy supports each individual team), in general such additional organizational layers create unwanted risks: miscommunication, misunderstanding, and increased cycle time.

In his book *Agile Product Development with Scrum*, Roman Pichler clearly describes how having BAs stepping into PO-proxy roles creates multiple problems that ultimately lead to "decrease of productivity and morale."[2]

Therefore, although becoming a PO-proxy is much more realistic than becoming the PO, it still does not provide a very effective solution for accommodating BAs, as the volume of BAs that each organization harbors today by far exceeds the number of available PO-proxy openings. This is a very basic supply-and-demand dilemma.

PO "candidate"

Let's face it, for a business analyst (or for someone at the same organizational level) to step into a PO's role is nothing but a great opportunity for career growth. It is a step up into the spotlight, gaining more visibility and authority, being presented with more opportunities to network and form useful professional relationships. Today, there are many BAs who are asked to do the heavy lifting for POs, including writing stories, backlog grooming, communicating with feature teams (especially offshore teams)—pretty much everything except making final decisions. This creates a situation in which BAs do a lot of heavy lifting for little recognition.

Elevating BAs to PO-proxies (still much more realistic then becoming the PO), deputizing them to be not just backstage servants but rather front-stage leaders, creates a rewarding situation for them, whereby assuming a more important organizational position becomes very attractive.

Let's look at the opposite situation: An individual who already has a high-ranking job with a company is asked to step into the PO's shoes.

Such an individual already has plenty of visibility, authority, and, most certainly, lots of day-to-day responsibilities. This individual's job has already been defined in "pre-Agile" terms with clearly formulated success/failure indicators, and, crucially, a compensation structure. Naturally, such an individual will *not* genuinely embrace the additional PO role *unless* the following conditions are met:

1. His other day-to-day responsibilities are minimized.
2. His compensation is increased to justify the additional efforts required to perform the second job.
3. There is a hybrid of the first two conditions: reasonable workload, reasonable compensation, no significant loss of organizational positioning.

In most of cases, companies attempt to fulfill the first condition but complete it only partially by simply doing an internal reorganization and appointing an individual for the PO role *without* removing his existing responsibilities. There will always be some resistance along the way.

Therefore:

- For a product manager or a key SME/business user to have more direct interaction with technology usually means performing "dirty" work.
- For a product manager or a key SME/business user to step into a PO's role sometimes means stepping down in the organizational tree and giving up direct reports—something that might be required to avoid conflicts of interest.

Regardless of overall workload for a newly baked PO, the type of work required by the role and how this work ties to organizational positioning and monetary rewards (e.g., potential bonuses) is usually the reason for low support.

Under the second condition (the likelihood of which, by the way, is not high at large organizations since salaries and bonuses are tied to organizational levels), yet another likelihood for failure arises: No extra money would compensate for the extra time and energy that a worker needs to spend on doing a second full-time job. It is highly unlikely that an individual would be able to sustain twice the workload without having it affect his personal lifestyle. It is also unlikely that a company would be willing to increase an individual's compensation twofold to pay for a double effort.

And as has been proven many times, "highly compensated heroics" never have long-lasting effects.

It seems that the only potentially workable option would be to create hybrid conditions under which, on one hand, the PO feels safe that his role within an organization did not depreciate (although the workload would decrease) and, on the other hand, he was able to give the required time and attention to Agile teams as needed. At the same time, the PO's

compensation increases and rewards would have to be within reasonable terms to compensate for significant additional work, yet not cause serious exception to a compensation formula used by a company.

In his book *The Art of Product Management*, Richard Mironov[1] describes situations when the role of product owner in Scrum is fulfilled by a person whose day-to-day duties and responsibilities resemble that of a PO: product manager. The product manager is a conventional, outward-facing role of a product business owner. Of all potential candidates for the PO role in Scrum, the product manager seems to be the closest to the PO role.

Mironov goes into detail, outlining how the two roles (PO and product manager) differ, specifically stressing the fact that the speed of crashing/failure is much higher for a PO than it is for a product manager. This occurs because things move much faster in Scrum and time frames to observe antipatterns and shortcomings are much narrower.

The most important thing that the two roles have in common, and the main reason for the lack of success, is *lack of engagement*.

Project manager

One of the most questionable roles in Agile product development is that of the project manager. Is this role still required? This topic is controversial and causes a lot of discomfort when raised.

For many PMs, the uncertainty about how their jobs will be impacted by Agile is the reason why Agile meets resistance in the conventional PM world.

Agile adoption presents different challenges for technical managers and nontechnical managers. It is important to make a note of this distinction.

For mid-level technical managers, the dilemma is primarily psychological and behavioral. Technical managers are expected to have hands-on expertise and, if needed, should be able to roll up their sleeves and produce technical work relatively quickly. This is exactly what counts in Agile product development—producing tangible technical work. The main challenge that technical managers face is their ability to let go of authoritative power and control of their subordinates. In Agile, tactical technical decisions are to be made at a team level, not at a technical management level. Psychologically, loss of such centralized control creates a problem—it is discomforting. The situation may worsen if a technical manager needs to become a member of a feature team, where he starts working side by side with individuals who previously reported to him.

As mentioned in the discussion about POs, organizational flattening and the adoption of an Agile framework may translate into a loss of jurisdictional power for some, and technical managers are not an exception.

Nevertheless, this does not create a "potential job loss" situation for technical managers, as they are always able (or at least expected) to fall back on their primary technical skills and to integrate with feature teams. Alternatively, some individuals can get promoted from mid-level technical management to more strategic technical leadership, where they are not being so much involved in tactical work at the team level but are responsible for more strategic decisions and resource planning across multiple teams. But again, just like the space for "upraising" BAs to PO-proxies is limited, so is the space for uprating mid-level technical managers into more senior positions: Supply and demand rules still apply.

It is important to note that the notion of senior technical leadership does not go away when Agile is introduced, as senior technical management is still needed. It is the abundance of mid-level technical management and the redundancy of their work.

Things are much more different for non-technical managers.

In Agile, let's take the Scrum framework as an example. The responsibilities of the project manager are evenly distributed between the PO and the team. The PO is now responsible for all strategic planning: product vision, product road map, timelines, budget, and scope (which remains flexible most of the time). A team is responsible for all tactical activities: sprint planning, story estimation, task breakdown, work assignment and work flow management, various team ceremonies, etc. A Scrum Master, selected by a team (the ideal case), usually picks up various team logistics, resolving inter- and intra-team impediments, brokering and facilitating ceremonies, negotiating with the PO, protecting a team from undesirable external influence, and escalating problems to executive management.

So is there anything left to do for a mid-level non-technical project manager in an organization that gets leaner and flatter as it undergoes Agile transformation, lightens its processes on all fronts, and gets rid of all its "procedural" waste? It all depends on how easily a non-technical PM can adjust.

Among other less apparent elements that are required for mid-level non-technical PMs to stay afloat in an Agile-transforming organization, the two main adjustment requirements are these:

1. Mental shift away from command and control behavior.
2. Ability and willingness to pick up additional technical skills that would make the PM more valuable in a product development process.

The first is all about behavioral patterns. It is about accepting that a group of skilled professionals, when empowered, can make decisions about their own work better than any outsider who is not doing the actual work, especially if such an outsider is not qualified technically.

Since Scrum (we continuously use this Agile framework as an example, as it is the most structured one of all Agile frameworks) implies self-direction and self-governance, any attempts to force anything upon a feature team will have adverse effects on both parties: on one hand, it will deteriorate Scrum and stall evolvement of a Kaizen culture, and on the other hand, it will marginalize a person who attempts to act as enforcer even further from where the real action takes place.

At this point, it is worth mentioning one very common anti-pattern that is often observed in large organizations as they undergo Agile transformation: The Scrum Master role automatically gets filled by a former PM. Is this default assignment proper?

Automatic sanctioning of PMs with SM responsibilities may cause more harm than good, and this has proven to be the case on multiple occasions. Having PMs go out and get certified as Scrum Masters is not sufficient. If a PM mind-set remains, the person will never become a Scrum Master. Unfortunately, mind shifting is not something that any certification can change. When the PM steps into the SM role but continues using his command and control tactics, it demoralizes the team and becomes its biggest impediment. The situation becomes even more dangerous if a newly baked SM who used to be a direct manager of other (one or more) team members now becomes their SM. Even if organizational reporting lines are removed, psychological dependency frequently remains and prevents team members from thinking and acting freely.

In his book *Scaling Lean and Agile Development* (coauthored with Bas Vodde), Craig Larman explicitly warns about harvesting "fake Scrum Masters." Larman's quote, "Changing the title of someone to 'Scrum Master' while he acts like—and is encouraged by the organization to act like 'a project manager'"[5] alludes to the fact that simply relabeling old roles into new ones does not cultivate better behaviors and does not improve culture.

Although there are many PMs who are willing and capable of undergoing a mind shift to become SMs, they still represent a fraction. Some PMs also still feel that becoming a SM is a step down in their careers, a demotion in a way, because now they no longer have the power to control the actions of others. They feel discouraged by the situation and start seeking other career paths. Are their employers aware of that? What is also worth mentioning is that, unlike the PO (we still refer to Scrum roles for the reasons mentioned above), the Scrum Master position is rarely a full-time job. Unless the SM is sanctioned to support multiple teams, which is also not always desirable, it remains a part-time facilitator role. Under ideal conditions, the SM role can be picked up by any team member, or, what is even better, by a member of another feature team. (This way, a team will be able to completely avoid a conflict of interest and ensure neutrality.) Since the SM role is not a full-time job in most of cases, it also means that a person who holds it is expected to have specific functional (technical or semi-technical) responsibilities that can

benefit a feature team in product development.

This brings the discussion to the following question:

Is a non-technical PM willing and able to gain additional technical skills and functional expertise to remain an asset within an organization that is undergoing Agile transformation?

For many non-technical PMs, learning technology is not an easy task. It is not always easy to switch from many years of using conventional project management tools and techniques, such as a project plans, project charters, WBS, Gantt charts, and individual task assignment lists to Java console, class libraries, Web services API, automatic test tools, or Agile software development collaboration tools.

Often, project managers select a less technical direction in their transition. They retrain as BAs and embed with teams, where they begin serving the purpose of PO or PO-proxy conduits, by supporting teams with business requirements (backlog items). This shift certainly helps in retaining resources, which is always a positive characteristic of any corporate culture, but it only works if there is a very specific need for having a BA embedded with a team. (This is more practical in distributed Scrum with offshore teams.) Otherwise, creating an extra layer in the business-to-technology communication channel is not recommended.

And again, supply-and-demand rules suggest that there are not enough team BA vacancies to accommodate all requalifying PMs.

Epidemic of certifications

> *"It is not enough to have knowledge, one must also apply it. It is not enough to have wishes, one must also accomplish."*
> *– Johann Wolfgang von Goethe*

Problem statement

Over the last couple of years, the variety of Agile certifications has significantly increased. Specifically, entry-level certifications that attest to basic Agile knowledge (Scrum framework, specifically) are now available in abundance. The type and depth of knowledge that these certifications offer (usually superseded by a condensed training course) are similar and, to a large extent, they cover identical topics.

If we search for reasons why there is such an abundance of basic-level Agile certifications, it will become apparent that it has become more about market share

retention by certifying organizations that operate in the Agile arena than about delivering unique, universally acceptable attestation standards that, on one hand, truly reflect individual hands-on practical expertise and theoretical knowledge and, on the other hand, enable individuals to use earned Agile credentials for securing a competitive job.

Discussion

This discussion is not about comparing one certification to another or suggesting which certification is better. It is also not about sharing research data about which certifications are recognized by the industry. Most certainly, the intention is not to compare pricing or promote any certification—they are all, when packaged skillfully, attractive. If a reader wants to fully grasp the variety of introductory Agile certifications available today, he can always do a Web search that will produce at least a half-dozen options.

The main purpose of this discussion is to stress the following point: There are *a lot* of comparable certifications to pick from, and this creates unnecessary competition in a space where a universally recognized accreditation standard would be much more desirable.

Competition between organizations that takes the form of "my Agile is better than yours" creates confusion for those who want to get certified and capitalize on their fairly earned practical experience. It also must be noted that by harvesting so many redundant certifications, the industry creates favorable conditions for "certifications collectors"—individuals who obtain certifications for the sake of being certified. By doing so, such individuals skew the count accuracy of those who are really qualified for a job and fade the distinction between true practitioners and acronym collectors. At times, we see names of professionals in the Web space where certification abbreviations by far exceed the length of an individual's first and last names, combined. This is truly ironic. As practice shows, holders of multiple redundant certifications have little or even no practical hands-on experience in Agile space. Finally, there is no statistical proof that would support a claim that any company-employer recognizes *only* one type of certification over other types.

This creates yet another challenge for professionals seeking employment, as they do not know which certification to pursue to increase their chances of being hired. If a company is looking for a certification abbreviation next to a name, instead of experience, it will most likely end up with an underqualified candidate who will further discredit a certification by not being able to live up to a company's expectations of his subject matter expertise.

Conclusion

Let's restate the initial purpose of this discussion. The goal was to make a reader come out of his comfort zone and think about issues that are often omitted from "happy path" Agile themes. The goal was neither to suggest any conclusiveness on the subject matter nor to steer the reader toward any actions.

After reviewing this discussion, each reader should be able to develop his own objective perception on a situation, his own independent view and perspective.

Still, how everyone perceives this information will, to a large extent, depend on such factors as these:

- Is the reader's perspective personal or organizational in nature?
- How close to Agile transformation activities is the reader positioned today and how soon (if at all) will he be impacted by them?

Each factor by itself is influential, as are both in conjunction.

To some, this reading could be a great eye-opener and motivator to make personal adjustments to ensure job security and competitiveness in the job market. This could take on the form of becoming more valuable to one's own organization through pursuing education or training, or by becoming more selective of accreditations and certifications, or by planning an exit strategy to a workplace where conditions for non-Agile activities are favorable.

To others, especially to those whose views are more aligned with organizational ones (e.g., high-level executives, C-level officers), this writing could serve as a push toward internal adjustment: reorganization and/or retraining of resources, reconsidering employees awards and incentives models, revisiting contracts and SLAs with internal and external partners/vendors. By the same token, introducing inspection and validation checkpoints to ensure that there is continuous and gradual advancement toward long-term strategic goals would be another likely outcome of reading these pages.

Further, there will be a certain percentage of readers who will downplay the significance of this writing because of confidence in their "safety zone" away from Agile transformation (e.g., a move to a different department or company) or because they just do not expect a full impact of Agile any time soon (e.g., due to its slow adoption by the organization they work for).

Lastly, there will be parties (varying from individual employees to service providers to client companies) who will react to this discussion with disapproval and defensiveness for self-serving reasons. Most likely, such a reaction would be proportional to their level of involvement with Agile, from the perspective of business development, commerce, and enrichment. Here, fears of becoming a subject of scrutiny and an "ethical audit,"

should some of the issues that are being raised here find their way via wider broadcasting channels, will be the main driving force.

Endnotes and references

1. Mironov, Rich. *The Art of Product Management: Lessons from a Silicon Valley Innovator.* Createspace, 2008.
2. Pichler, Roman. *Agile Product Management with Scrum: Creating Products That Customers Love.* Addison-Wesley Professional, 2010.
3. Poppendieck, Mary; Poppendieck, Tom. *Leading Lean Software Development: Results Are Not the Point.*Addison-Wesley Professional, 2009.
4. Cockburn, Alistair. *Agile Software Development: The Cooperative Game* (second edition). Addison-Wesley Professional, 2006.
5. Larman, Craig; Vodde, Bas. *Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum.* Addison-Wesley Professional, 2008.
6. Logan, Dave; King, John; Fischer-Wright, Halee. *Tribal Leadership*. HarperBusiness; reprint edition, 2011.
7. Reinertsen, Donald G. *The Principles of Product Development Flow: Second-Generation Lean Product Development.* Celeritas Publishing, 2009.
8. Smith, Preston G.; Reinertsen, Donald G. *Developing Products in Half the Time: New Rules, New Tools* (second edition). Wiley, 1997.
9. Mezick, Daniel. *The Culture Game: Tools for the Agile Manager*. FreeStanding Press, 2012.