A Hands-on Manual from the Trenches by Stefan Wolpers

Version: 3.4 2018-06-06

Agile Transition

Inhaltsverzeichnis

Preface

Welcome to my hands-on guide on turning an existing product delivery organization into an agile, self-organizing one.

As you can easily spot this ebook is work in progress. It is based on various blog-posts I have written since October 2015 which have originally never been intended to form a book, or manual. Hence, there are gaps in the current text I am planning to close over the course of 2017/2018.

So, stay tuned!

Best,
Stefan Wolpers

PS:

Please note that I will not be able to automatically provide you with new versions of this ebook if you unsubscribe from the *Food for Agile Thought* newsletter.

In doing so, you also delete your email address from email-list of readers of this ebook. Thank you for your understanding!

Introduction

The Big Picture of Agile: How to Pitch the Agile Mindset to Stakeholders

Let's face it: While your enthusiasm for the big picture of agile practices is admirable, your stakeholders will most likely be moved by one thought only at the beginning of the transition: "What's in for me? How will I now have my requirements delivered?".

Read on and learn about one way how to kick-off the transition to a learning organization by pitching a simplified version the big picture of agile practices to your stakeholders first.

There Is Just One Spotify

Being a part of the team that builds and scales an organization from scratch over several years is rare privilege. Ben Linders pointed out recently that There Is No Spotify Model, and that you „shouldn't copy it in your own organization".

According to Henrik Kniberg, scaling Spotify…

"…wasn't a big re-make, more like a continuous stream of small iterative improvements to our organization and process. We have been growing for three years, and the way we work today has evolved naturally over time."

(**Note**: He is talking about an organization that has been carrying an agile mindset in its DNA from the start.)

You—on the other side—are more likely to be a member of a corporation that decided to become more agile to improve its standing in the innovation game. (Remember Marc Andreessen's "Why Software Is Eating the World"?)

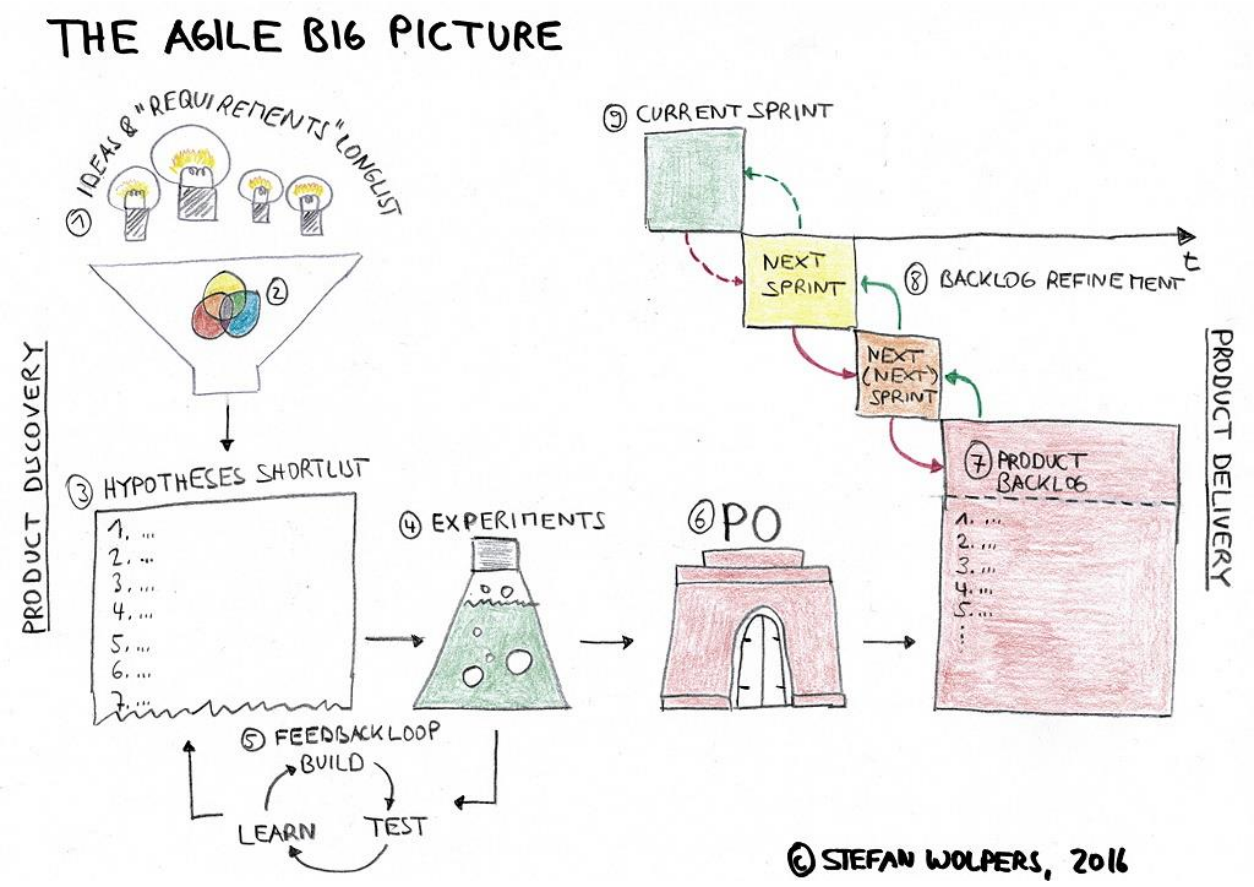
In your case, there is an established organization with its own flavor of structural particularities. There are hierarchies, there is politics and probably one form or another of silo thinking, or local optimization attempts. And cross-departmental communication has long been an issue.

There are likely people who will lose with the move to an agile organization. An organization that is build on top of formulating hypotheses, and running experiments. An organization that embraces failure, self-organization, and the acceptance of accountability. (To my experience, command & control structures can be comfortable, too, at least for some people. Read more in: Why Agile Turns into Micromanagement.)

The Hands-On Approach to Pitch the Big Picture of Agile to Your Stakeholders

No matter the kind of organization, “going agile” cannot be accomplished by an edict from the CEO. It needs to be sold, hearts and minds need to be won, and that is a journey not a destination.

A successful first step in pitching the benefits of agile principles and processes to stakeholders—at least to my experience—is starting the educational process with the following kind of “flowchart”, the big picture of agile:



The flowchart is admittedly handling “agile” at a simplified, reduced level. However, it is designed that way intentionally. There is the huge number of available agile frameworks and methodologies that cover everything from product discovery to continuous product delivery.

If you start with a too detailed version of this new universe, you might lose stakeholders early in the transition process to cognitive overload. The flow diagram does not intend to dumb it all down but prevent stakeholders from being driven into their own, possibly

negative, confirmation bias of "agile". Keep it simple, if you want to preserve an open mind among your stakeholders.

I am using the flow diagram for organizations with an existing product management organization that utilize Scrum to deliver the product. The diagram is intentionally combining several different agile and lean practices to compensate for Scrum's Achilles heel—the missing product discovery part.

Therefore, it starts with the competition of ideas for scare (development) resources on the product discovery side, and ends on the (Scrum-based) product delivery side. (Without specifying the technical nature of the delivery process, e.g. DevOps practices.)

The diagram addresses the following steps in a simplified agile process:

1. The ideas and "requirements" long-list, competing for engineering resources
2. The application of the "valuable, feasible, and usable" filter…
3. …thus identifying the short-list of test-worthy hypotheses
4. Running (lean) experiments to…
5. …validate hypotheses, representing how the learning organization is mitigating risk
6. The Product owner as the gatekeeper of the product backlog, visualizing the hand-over from product discovery to product delivery
7. The product backlog, representing at any given time the most valuable set of tasks for a Scrum team
8. The continuous product backlog refinement process…
9. …leading to the next set of user stories, tech tasks, spikes, or bugs for the Scrum team.

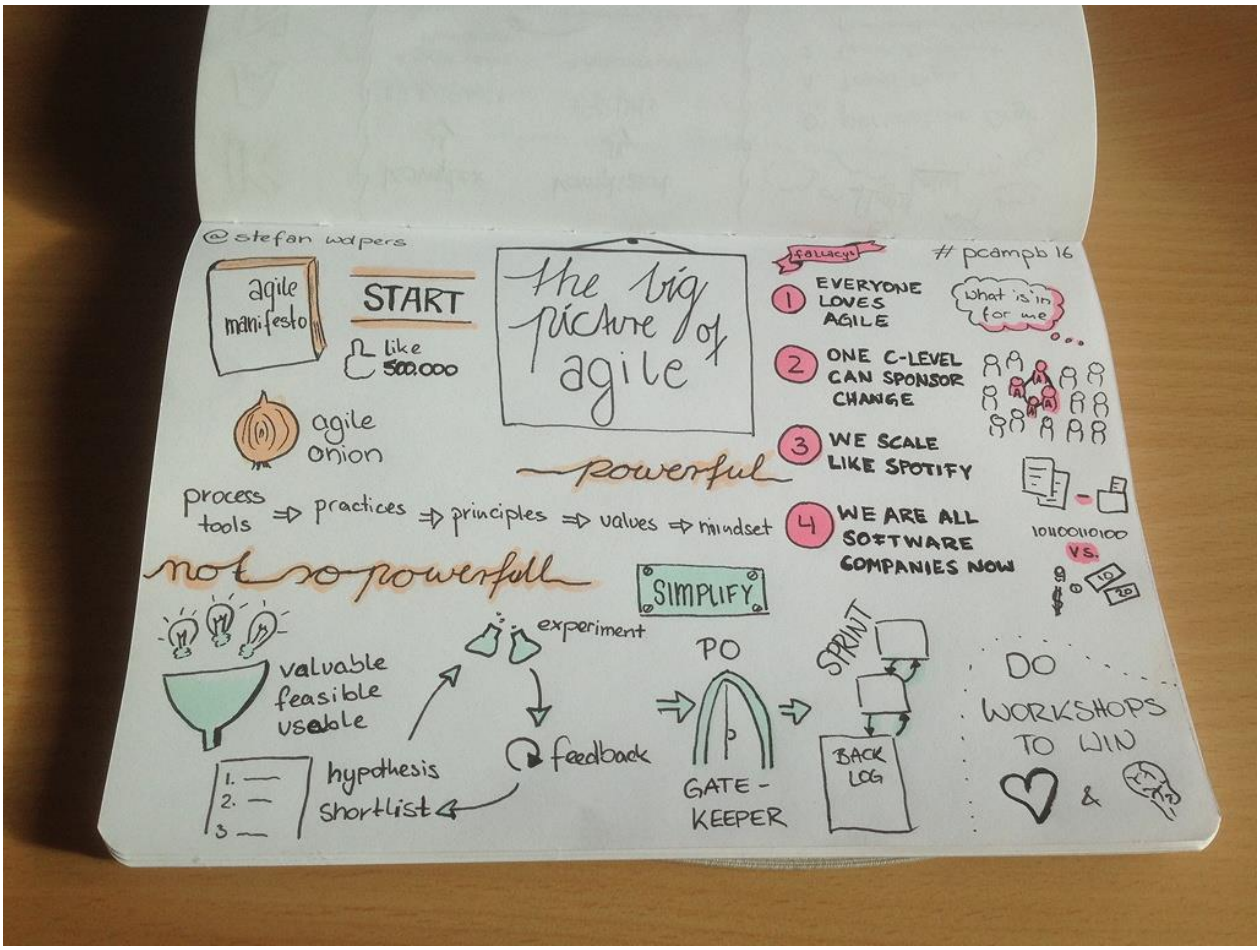
This initial coaching works well together with a follow-up workshop with stakeholders, during which the stakeholders actually build a clickable prototype of an app. (Read more here: App Prototyping with Absolute Beginners.)

Such a workshop greatly improves the future collaboration with those stakeholders that embrace the agile way of thinking—your future change agents.

And the workshop also identifies all those stakeholders that are opposed to the idea—the ones who will try to impede the progress of the transition to preserve their current position.

ProductCamp Berlin 2016

I presented the flowchart at the ProductCamp Berlin 2016, and Sabine Rougk created a great sketch note:



Check out Sabine's other sketch notes at her Instagram account @duda_draws.

Conclusion

Any transition to agile practices needs to be sold to an organization, it cannot simply be ordered. A proven first step to do so, is pitching a simplified big picture of agile practices to visualize the process from product discovery to product delivery, and communicate its benefits to the stakeholders.

Ideally, you follow up with a hands-on stakeholder workshop that teaches beginners to build a clickable prototype of an app.

How to Make Agile Work in Fast-Growing Startups

TL;DR: What Makes Agile in Fast-Growing Startups

From 2010 to 2017, I was working several years in three Berlin-based, fast-growing startups in my capacity as Scrum Master, agile coach, and Product Owner. These are my lessons learned on how to make 'agile' work in a fast-growing startup, and what anti-patterns to avoid at all costs.

Agile Startups: The Berlin Success Story of Zalando

Back in 2015, Zalando – Europe's largest online fashion retailer – introduced its flavor of Agile, dubbed 'radical agility'. It has proven since to be a smashing success, not just fueling the bottom line of the business, but also Zalando's ambition to build an outstanding product delivery organization. Quote:

"Over the last year and a half, we have doubled the technology team from around 800 in 2015 to over 1,600 currently. In addition to changing our business model, we also implemented a unique culture within the technology team called Radical Agility: This has seen monthly technology applications grow from 500 to over 2,000, and allows to ensure that we are hiring only the best quality."

Source: Matteo Bovio, Zalando SE

If becoming agile, as opposed to merely doing agile, is so successful, how come then, that not all organizations are pursuing a similar path as Zalando? Certainly, there are no shortages of agile methodologies, frameworks, and practices to choose from.

Six Fallacies that Prevent Startups from Adopting Agile Successfully

From 2010 to 2017, I was working several years in three Berlin-based, fast-growing startups in my capacity as Scrum Master, agile coach, and Product Owner. 'Fast-growing' in the sense of this article refers to startups of at least 200 staff strong. Each of the before-mentioned organizations doubled in size each year during a period of at least three consecutive years. All startups built double-sided marketplaces, serving B2C as well as B2B customers.

These are my lessons learned on how to become agile as a fast-growing startup, and what anti-patterns to avoid at all costs.

Fallacy #1: 'Agile' Equals More Bang for the Buck

If you ask founders and managers of startups why they want to become an agile organization, they typically name reasons such as:

- To become more efficient in software delivery
- To deliver faster
- To improve the predictability of software deliveries.

If you compare these answers to the actual benefits of becoming an agile organization, such as:

- Outperforming competitors by creating a learning organization
- Creating an outstanding culture by providing room for autonomy, mastery, and purpose, thus attracting talented people
- Mastering continuous product discovery as well as product delivery
- Minimizing risk, and improving the return on investment for product delivery organizations,

you immediately recognize the misalignment of motives and the real agile mindset.

Fallacy #2: No or Limited C-Level Sponsorship

The change to become a learning organization within a startup is by nature fundamental and continuous. It requires to permanently:

- Run experiments
- Embrace failure, and to…
- …abandon the famous "heroic inventor" mental model.

Unfortunately, the latter is often the driving force behind the founders' mental model of entrepreneurship although we all know that Steve Jobs did not single-handedly create Apple out of a void.

To my experience, the challenges of becoming a learning organization can only be handled effectively by self-organizing teams. Their collaboration will lead over time to a 'team of teams' structure. This approach requires at any stage the full backing of the C-level. A limited or lackluster support will render all efforts useless.

Equally futile by comparison to the lack of C-level support is a bottom-up approach by hacking the existing culture. It usually leads to frustration, and talented people with an agile mindset will seek for better-suited organizations elsewhere.

Fallacy #3: Everyone Loves 'Agile'

Change is much less appreciated than innovators commonly believe. Organization have an inherent inertia to change, which is a reason that they are successful: they provide stability. However, stability also breeds self-interest at all levels, but particularly at the level of the middle management.

There is the 'what's-in-for-me' syndrome: why would a middle manager put his or her career on the line by supporting the agile mindset? Taylorism – or supporting command and control structures in siloed organizations – still pays well today. It results in local optimizations and personal agendas. Also, career & CV optimization efforts are a motivation to be reckoned with.

Self-organization, on the other side, needs a different kind of management: teachers, coaches, and mentors. And just relabeling the positions of the old middle management rarely works.

Read more: Why Agile Turns into Micromanagement

Fallacy #4: We Know what to Build

This issue applies to both established as well as new organizations. You will often find a strong cognitive bias at the founder and management level towards the future direction of the product.

The bias tends to be reaffirmed if things work out ("I was right, and I will be right in the future"), or it will be rationalized in the case of failure. ("Something else was wrong no one could not foresee.")

Consequentially, the bias often results in micromanaging the product delivery organization. Also, it will nurture ignorance towards opportunity costs or costs of delay as strategic concepts.

Fallacy #5: Scale Like Spotify

"I don't understand the fuzz about agile coaching, and aligning the rest of the organization with product development. Once we are agile, we will copy the Spotify model and scale accordingly."

There seems to be a belief even among informed stakeholders that scaling an agile organization can be achieved by simply going through the checklist of a successful another startup. One of the favorite blueprints for that purpose is Spotify.

However, according to Henrik Kniberg – one of the head coaches at Spotify– it wasn't that easy:

"[It] wasn't a big remake, more like a continuous stream of small iterative improvements to our organization and process. We have been growing for three years, and the way we work today has evolved naturally over time."

Source: Don't Copy the Spotify Model

The truth is, you cannot just copy the Spotify model. This organization was built with such an idea in mind from the beginning, and yet Spotify needed to find its way. This "way" is what every other organization needs to figure out on its own: How to become an agile organization?

Fallacy #6: We Need a PMO for Product

A PMO – short for project management office – signals to agile practitioners that:

- You have a command & control mindset
- You are a Taylorist at heart
- You consider functional silos to be beneficial
- You neither understand product discovery, nor product delivery in the 21st century.

The idea to install a communication gatekeeper between the product delivery organization and its stakeholders and customers contradicts everything 'Agile' stands for. An agile startup does not require a PMO.

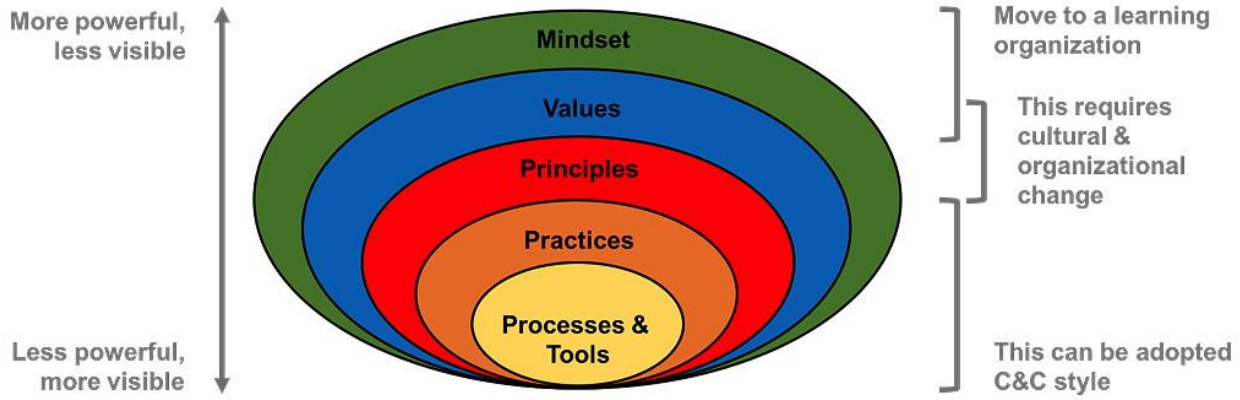
How to Make 'Agile' Work in Startups Then?

Apparently, becoming an agile organization is a tedious, lengthy journey. Therefore, the first question any startup should answer for itself, is simple: is it a sales-driven, product-driven or tech-driven enterprise?

Not all organizations need to become truly agile and may still create a great culture, and deliver a return to investors.

However, if competition is fierce, technology is advancing rapidly, and big players are investing large amounts, then there is nowadays practically no way around becoming a learning organization.

And that process requires several steps as the following graphic visualizes:



Source: What is Agile? **Graphic**: By the author.

Getting from the processes & tools level to the agile mindset is a daring journey, and no one can guarantee that the endeavor will result in the desired outcome.

So, if your startup is a call-center with a homepage, you may want to consider not going down the agile rabbit-hole. You may very likely plateau at level 2, establishing some isolated agile islands within the organization.

So, be honest with yourself: is that worth the effort?

How-to #1: Culture

Great, you want to become a truly agile organization. Then let's start with the most challenging issue right away.

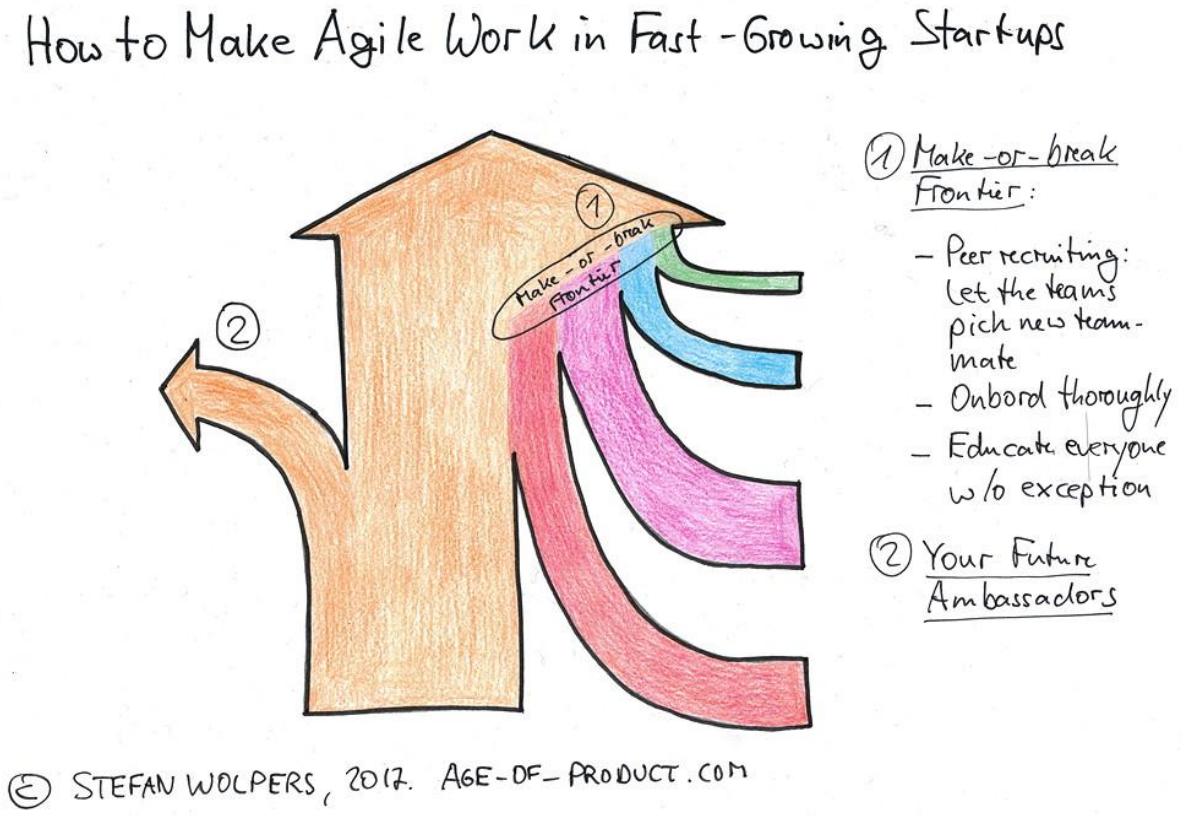
You started out with a small team, and your way of being agile seems to start working. Your startup is getting traction, new funding, and your investors urge you to hire more people, and particularly to hire more "senior people" from larger organizations.

If you are now onboarding five, ten, or 20 people a month, you have a serious issue at hands: how to preserve your original team spirit, your original (agile) culture?

"*Culture eats strategy for breakfast.*" (Peter Drucker)

This observation is the main reason that you need to protect your nascent agile culture at all costs. Don't let culture "evolve" by hiring people from (legacy) organizations. A former BCG manager will likely urge to establish a project management office (PMO) because this is the way she knows how to organize work. And as culture tends to follow structure you will strangle your agile culture exactly in this way.

There are only two means how to avoid this sort of collateral damage. You need to invest both in diversity and education of every new hire.



Ensure that everyone understands why your startup needs to be agile. This learning is not achieved by handing out leaflets. It only works by teaching the big agile picture and winning hearts and minds of everyone within the organization.

Read more: The Big Picture of Agile.

How-to #2: Winning Hearts & Minds w/ Education

How do you teach the big agile picture, thus winning hearts and minds of the new hires? Train everybody – without regard to her future position – hands-on in all value-creating activities of the organization. Start with customer care, and steal from Zappos.

Plus, and that is my favorite activity, run mandatory workshops with all new hires where they have to prototype a new app. The exercise works for everyone — sales, marketing, customer care, finance, HR… — no specific knowledge required.

One of my workshops takes about 6 to 8 hours, and the participants build a clickable prototype of a team-event organization app. They learn how to figure out what app might be valuable as well as how Scrum and user story mapping work.

At the end of the workshop, people either love building products or the product delivery organization earned at least their respect. Both are highly valuable mindsets for an agile startup.

Read more: App Prototyping with Absolute Beginners.

How-to #3: Hire the Right People

As you may have guessed already, recruiting the right people for the organization is the make-or-break frontier for any plan to adopt agile in fast-growing startups.

Sticking to following hiring principles will make the process significantly more manageable:

- Always hire for mindset & cultural fit, never for skills. The latter can be taught, but you'll never turn an a**hole into someone likable
- Look for intrinsically motivated people: they will care for what they help create
- Hire for the best possible team, not for the best fit for a particular position. (If you haven't yet read or watched "Moneyball," do so, and you will understand the principle.)
- Lastly, diverse teams are more innovative.

As far as the recruiting process is concerned, let the teams choose their new teammates, and your failure rate will drop significantly. HR will probably not like it but how can you aim for self-organizing teams and patronize them at the same time?

Read more: Peer Recruiting.

How-to #4: Team Building

You cannot overestimate the importance of team building: the team wins, the team loses. And Tuckman's findings is still very valid. The objectives of the team building effort are apparent:

- Let the teams constitute themselves, avoid drafting team members
- All teams shall become self-managing over time
- All teams need to become cross-functional, as feature teams are required, not component teams
- High performing teams are co-located.
- Other team building lessons learned:

- Ensure that the onboarding of new members is done properly. Prepare everything in advance, and hook them up with a buddy from a different part of the organization
- Don't shuffle team members around between teams — a team is not a resource pool
- Line managers and subordinates cannot be teammates
- Use delegation poker to outsource tasks from teams to the management.

Read more: Zappos' Outrageous Record For The Longest Customer Service Phone Call Ever

How-to #5: The Agile Workspace

The agile workspace – the underestimated success factor for agile organizations. It is interesting to observe that even newly designed office of startups that pride themselves to be agile lack proper facilities.

Adopting agile does not come cheap as far as the workspace is concerned. You will not just require more space. You will also need different spaces to support the four modes of creative work:

- Focus
- Collaborate
- Learn
- Socialize

Also, the startup-like feast and famine cycle of stuffing of more people into a once generous space until the next move is no longer an option. Whiteboards and other information radiators require space all the time if an agile startup wants to reap their benefits.

Read more: Agile Workspace: The Undervalued Success Factor

How-to #6: Sound Engineering Practices

Garbage in, garbage out. No matter how agile your product discovery process is, it needs to be matched by similar engineering practices.

You build it, ship it, run it:

- DevOps
- Probably a micro-service architecture
- Test automation (TDD, BDD)
- Continuous integration
- Probably continuous delivery
- You need to keep technical debt at bay

Component teams need to become cross-functional feature teams:

- Encourage a holistic product view
- Focus on end-to-end feature delivery
- There should not be any form of code ownership
- Co-locate all teams. No matter what effort you put into communication between remote teams, they cannot match the effectiveness and productivity of co-located teams.

Lastly, don't let Salesforce just "happen" by accident. It is a familiar story: since engineering is busily building the application for the customers, the internal backend for sales and customer care is left temporarily to Salesforce.

The typical sales pitch goes like: "we need a CRM software, and why would we build something that we can license anyway?" (Which is legitimate.) The initial set-up is small, just a bit customization, but after a short period requirements from sales start emerging that only can be met by custom development.

And this is the moment when a parallel IT universe is created, and people without software competence – not mention an understanding of software architecture – gain a say in designing your application. The situation will get particularly nasty when Salesforce begins to deviate in functionality from the real application, and syncing data back and forth becomes a major task for the engineering teams.

Without proper technical leadership and stakeholder management, Salesforce will irrevocably spread in the organization causing frustration throughout engineering & product, bypassing a lot of agile practices. (All three of the before-mentioned startups suffered from that syndrome.)

How-to #7: Agile Metrics

The general purpose of agile metrics is to understand the current situation better and to gain insight on change over time. An agile metric should, therefore, be a leading indicator for a pattern change, providing an opportunity to analyze the cause for change— and act in time if required.

Contrary to traditional command–and–control organizations, metrics in an agile context are not used to (micro-)manage the individual, the creative worker. In an agile organization, metrics are used to provide the team with insights on how to improve continuously.

So, a simple way to undermine the process of becoming an agile organization is to apply the same bean-counting approach as a command–and–control oriented organization would pursue. Therefore, consider carefully what agile metrics your startup is using:

Good metrics:
- Lead time
- Cycle time
- Ratio of fixing work v. feature work
- No. of bugs on production

Bad metrics:
- Velocity

Ugly metrics:
- Story points per engineer per time unit.

Read more: Agile Transition: Agile Metrics—the Good, the Bad, and the Ugly

Conclusion

There is no one-size-fits-all approach to become an agile organization. There is no checklist that you can apply.

You need to figure out your way. It will take money, brain, and time to do so. You might fail, you will probably plateau at a certain stage, and once your startup stops going forward, it will likely start moving backward.

So, don't lose faith in the process and always question yourself:

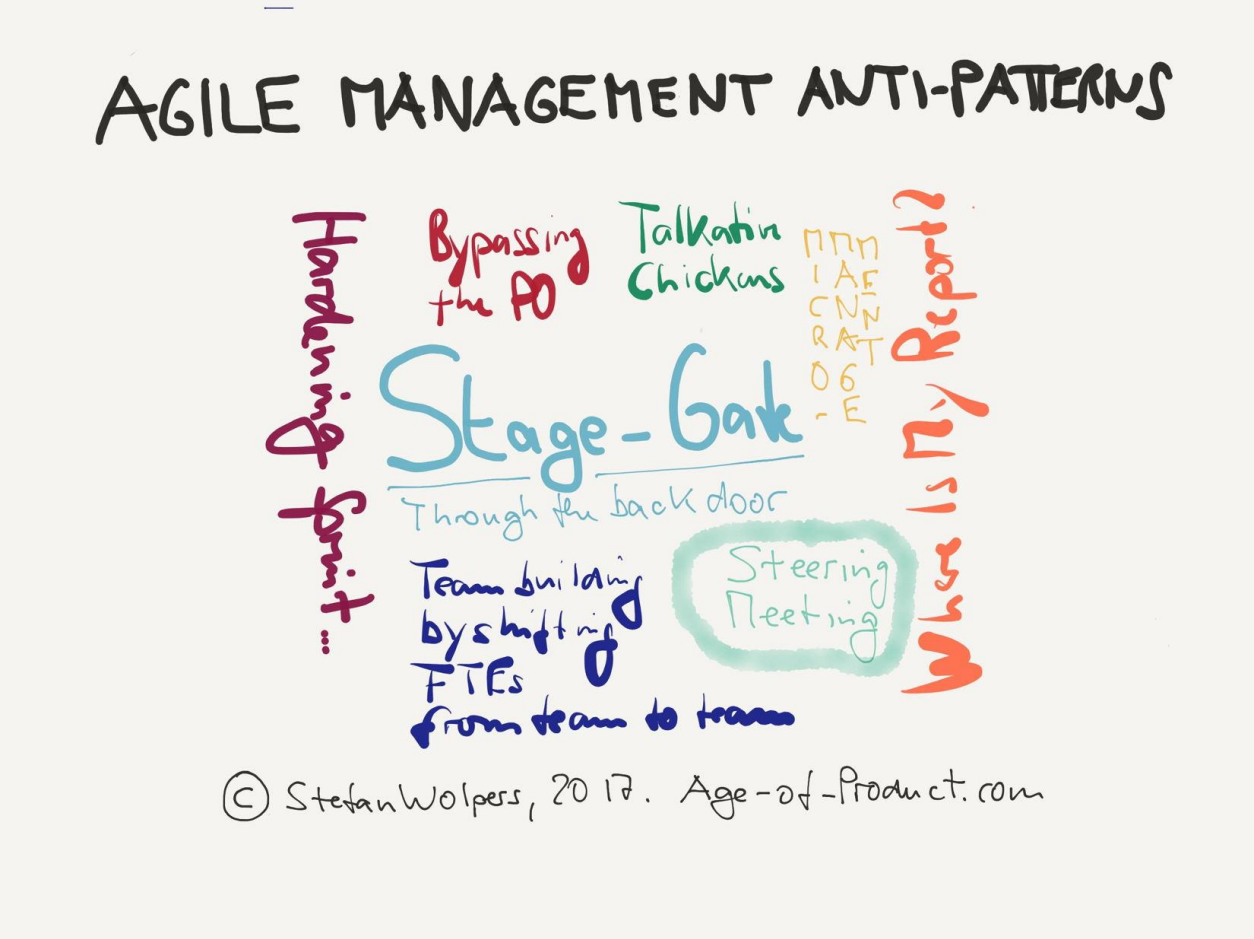
"Whenever a theory appears to you as the only possible one, take this as a sign that you have neither understood the theory nor the problem which it was intended to solve." (Karl R. Popper)

Source: Goodreads on Dogmatism

Agile Management Anti-Patterns — An Introduction

Summary: The Nature of Agile Management Anti-Patterns

Learn more about agile management anti-patterns the aspiring agile manager should avoid during the organization's transition. From stage-gate through the back door to the 'where is my report' attitude.



Agile Management Anti-Patterns in Learning Organizations

The following anti-patterns are a selection of common behavior by traditional managers often found in organizations that haven't yet managed to become (fully) agile:

- **"I am a CSM, too."** A manager should avoid mentioning this level of formal education in discussion with agile peers if she wants to be taking seriously. While everyone will appreciate the effort, participating in a 2-day workshop does not mean that she

mastered navigating the waters of dual track agile. Respect needs to be earned; it is not delivered by rank.

- **Using the budgeting process as a stage gate to exercise control through the back door**: The budgeting process is hard to align with agile requirements like the longevity of teams. Hence, the original process will have to change over time. However, it is a sign of mistrust, though, if the management uses the budget permanently as a means of controlling the teams, making them "pitch" every two months or so. There is absolutely nothing "agile" to be found in this practice. Instead, the management ought to provide the teams with goals and guidance how to achieve these along with funding that is sufficient to meet the objectives.

- **The 'where is my report' mentality**: The manager expects to receive reports regularly instead of participating in ceremonies, for example, the stand-ups or the sprint reviews. A quick look at the Manifesto for Agile Software Development should help the manager, though: 'Individuals and interactions over processes and tools' is a core principle of all agile practices.

- **Steering meetings**: Unimpressed by the agile ways of working, the manager insists on continuing the bi-weekly steering meetings to ensure that the team will deliver all her requirements in time. This one has a quick remedy, though: just do not participate in meetings that have no value for the team.

- **Team building without involving the team**: The manager decides who is joining or leaving the team without involving the team itself in the decision process. Building a high performing team is not accomplished by moving FTEs from one spreadsheet to another. There is a reason why special forces, for example, put so much effort and time into the longevity of teams and any agile organization should follow the example. The least the manager can do is involve the team in all decision that may affect the team composition. **Read More**: "Peer Recruiting."

- **Telling people how to do things**: In the good old times on the shop floor, it was a valuable trait to train newcomers or work groups in the art of assembling a Model T—as the manager probably did herself. Nowadays, as we invest most of our time building products that have never been built before this attitude becomes a liability. Just let the people closest to the job at hands figure out how to do this. Guidance by objectives and providing support when requested or needed will be appreciated, though.

- **Abandoning management all together**: Interestingly, some managers seem to believe that self-organizing teams do not require any form of management anymore. Nothing could be further from the truth: There are always issues at an organizational level that teams love out sourcing to their management. If you like to

figure out what that might be, give Management 3.0's delegation poker a chance.

- **Disrupting the flow**: The manager disrupts the flow of the scrum team by inviting random team members to various meetings of little or no value, or by disrespecting time-boxes of scrum ceremonies, or the sprint itself. Scrum master, you will need to intervene in this case.

Scrum-Specific Agile Management Anti-Patterns

If your organization uses scrum as a part of the product delivery process, there are various scrum specific management anti-patterns. (Please note, that managers and internal stakeholders tend to overlap functionally and are often used in this context synonymously.)

General Agile Software Development Anti-Patterns

These anti-patterns show tendencies to overrule scrum as a 'fine weather sailing' exercise:

- **All hands to the pumps w/o scrum**: The management temporarily abandons scrum in a critical situation. (This is a classic manifestation of disbelief in agile practices, fed by command & control thinking. Most likely, canceling sprints and gathering the scrum teams would also solve the issue at hands.)

- **Reassigning team members**: The management regularly assigns team members of one scrum team to another team. (Scrum can only live up to its potential if the team members can build trust among each other. The longevity of teams is hence essential. Moving people between teams, on the contrary, reflects a project-minded idea of management. It also ignores the preferred team building practice that scrum teams should find themselves. All members need to be voluntarily on a team. Scrum does rarely work if team members are pressed into service. Note: It is not an anti-pattern, though, if two teams decide to exchange teammates temporarily. It is an established practice that specialists spread knowledge this way or mentor other colleagues.)

- **Special forces**: A manager assigns specific tasks directly to engineers, thus bypassing the product owner. Alternatively, the manager removes an engineer from a team to work on such a task. (This behavior does not only violate core scrum principles. It also indicates that the manager cannot let go command and control practices. He or she continues to micromanage subordinates although a scrum team could accomplish the task in a self-organized manner. This behavior demonstrates a level of ignorance that may require support from a higher management level to deal with.)

- **Hardening sprints**: While the idea of continuous delivery is to permanently provide more value to customers by frequently shipping new code, it is fine to continue

thinking in releases. There may be legal reasons for that, or customer care and sales need to be thoroughly trained in advance, or the customers prefer it this way. What is not acceptable in an agile engineering organization, though, is preserving QA as a separate, siloed entity that demands time for end-to-end testing before releasing any new code. That is true cargo cult agile. Instead, move the QA engineers into the scrum teams and focus on test automation and other well-established CD practices.

Agile Management Anti-Patterns: The Stand-up

The daily scrum is a simple ceremony, and yet it can be abused in various ways by managers:

- **Reporting session**: The manager expects that everyone delivers a sort of status report. The anti-pattern has an advanced version: the manager "runs" the stand-up, signaling the team members when it is their turn to speak.

- **Talkative chickens**: By definition, the manager is a "chicken," and chickens are supposed to listen in. Actively participating in stand-ups is reserved for team members. (It is acceptable if managers ask a question during the stand-up, though. Beyond that they can talk at any time to the product owner or scrum master.)

- **Anti-agile**: Line managers are attending stands-up to gather "performance data" on individual team members. (Do not be surprised to find them thinking in the story-points-per-developer "performance" category as well.)

Agile Management Anti-Patterns: The Sprint

There are also attempts to bypass scrum principles, for example, by ignoring the nature of the sprint backlog such as:

- **Pitching developers**: The manager tries to sneak in small tasks by pitching them directly to developers, this ignoring the product owner as well as basic scrum principles.

- **Everything's a bug philosophy**: The manager tries to speed up delivery of certain features by relabeling tasks as 'serious bugs.' (A special case is an "express lane" for bug fixes and other urgent issues. Every manager will try and make his or her tasks eligible for that express lane.)

Agile Management Anti-Patterns: The Sprint Review

It does not take a lot for managers to derail the scrum team's motivation in the most important scrum ceremony — the sprint review. Watch out for the following anti-patterns:

1 **Scrum à la stage-gate**: The sprint review is a kind of stage-gate approval process where managers sign off features. (This anti-pattern is typical for organizations that use an agile-waterfall hybrid. Otherwise, it is the prerogative or the product owner to decide what to ship when.)

2 **No manager in the trenches**: Managers do not attend the sprint review. (There are several reasons why managers do not attend the sprint review: they do not see any value in the ceremony. It is conflicting with another important meeting. They do not understand the importance of the sprint review event. None of the sponsors is participating in the sprint review, for example, from the C-level. To my experience, you need to "sell" the ceremony within the organization to successfully transition to an agile organization.

3 **Starting over again**: There is no continuity in the attendance of managers. (Longevity is not just a team issue, but also applies to stakeholders. If they change too often, for example, because of a rotation scheme, how can they provide in-depth feedback? If this pattern appears the team needs to improve how managers understand the sprint review.)

4 **Passive managers**: The managers are passive and unengaged. (That is simple to fix. Let the managers drive the sprint review and put them at the helm. Or organize the sprint review as a science fair with several booths.)

Agile Management Anti-Patterns: The Retrospective

While managers are not supposed to participate in retrospectives, they might nevertheless negatively influence those. These three agile management anti-patterns about scrum that are notable:

- **Line managers are present**: Line managers participate in retrospectives. (This is the worst anti-pattern I can think off. It turns the retrospective into an unsafe place. And who would expect that an unsafe place triggers an open discussion among the team members? Any line manager who insists on such a proceeding signals his or her lack of understanding of basic agile practices. Note: If you are small product delivery team at a start-up and your part-time scrum master (or product owner) also serves in a management function, retrospectives might be challenging. In this case, consider hiring an external scrum master to facilitate meaningful retrospectives.)

- **Let us see your minutes**: Someone from the organization—outside the team—requires access to the retrospective minutes. (This is almost as bad as line managers who want to participate in a retrospective. Of course, the access need be denied as retrospectives are a safe place.)

- **No suitable venue**: There is no adequate place available to run the retrospective. (The least appropriate place to have a retrospective is a meeting room with a rectangular table surrounded by chairs. And yet it is the most common venue to have a retrospective. Becoming agile requires space. If this space is not available, you should become creative and go somewhere else. If the weather is fine, grab your Post-its and go outside. Or rent a suitable space somewhere else. If that is not working, for example, due to budget issues, remove at least the table so you can sit/stand in a circle. Just be creative. Read More: Agile Workspace: The Undervalued Success Factor.)

Conclusion: Agile Management Anti-Patterns

There are numerous agile anti-patterns of typical managers, covering the whole range from mindset to processes. While the latter can be fixed by providing training and coaching continuously, the former is not trivial to address. While some die-hard command and control-minded managers will never make the transition, others may have a chance to change their perspective from 'how do I make my superior happy' to 'how can I make my team more successful.'

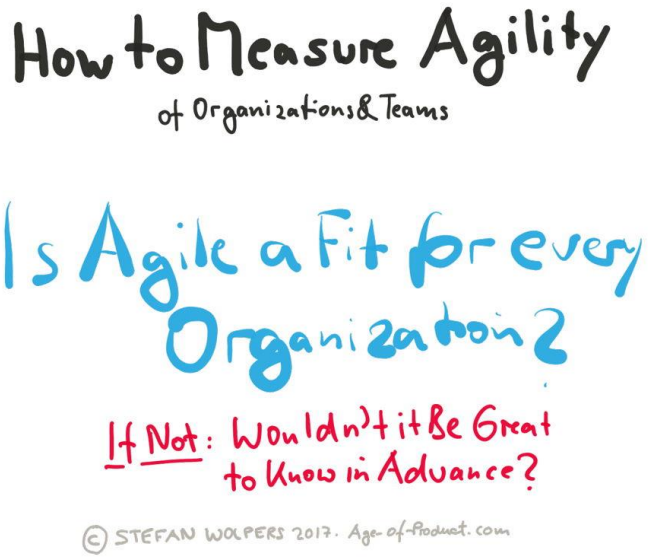
How to Measure Agility of Organizations and Teams

The Agility Assessment Framework—An Introduction

Is every organization suited to become 'agile?' If so: How to measure agility? And if not: Wouldn't it be great figuring that out before embarking on a futile and expensive journey? Back in October and November 2017, I ran a survey to identify contributing factors to an organization's or a team's agile maturity. In total, 86 people participated. Based on their answers, I aggregated a preliminary taxonomy of agility related factors.

This taxonomy was first presented on the Hands-on Agile Berlin meetup on November 30th, 2017.

On February 3rd, 2018, 20-plus people will join a hackathon to build an agility assessment framework based on this taxonomy. The goal of the workshop is to provide the first version of a tool that empowers agile practitioners to measure agility, be it an organization's suitability for agile practices or a team's progress on its path to becoming agile.



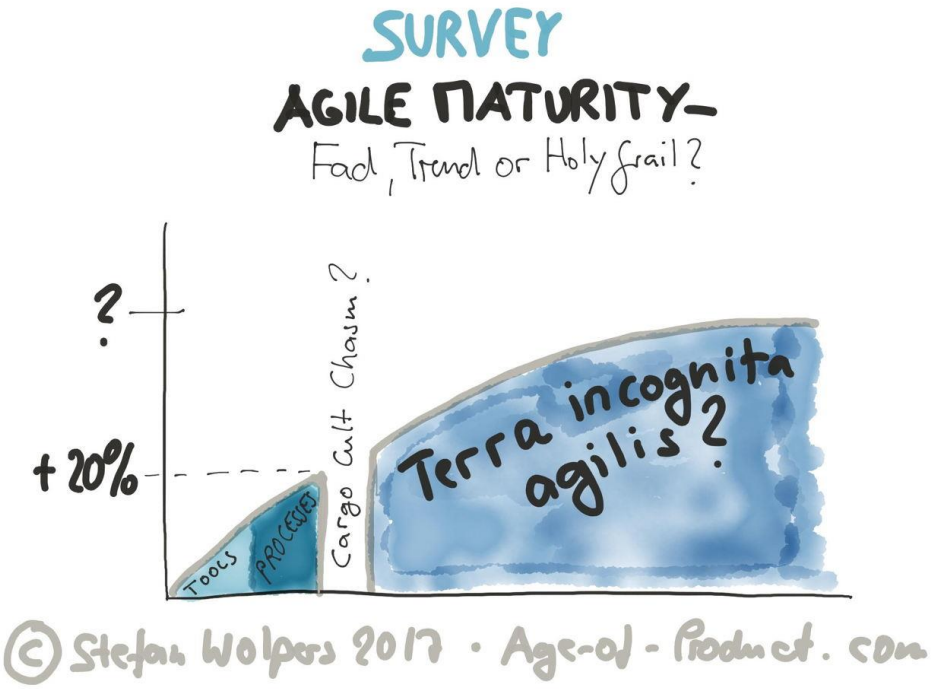
The Current State

Measuring agility is nothing new. There are plenty of tools and approaches available, starting with Crisp's Scrum Checklist to James Shore and Diana Larson's Agile Fluency™ model. Measuring agility of prospective clients has become a valued presales tool for many consultancies, too.

What is missing today, though, is an open-source and thus widely available framework that any agile practitioner can use to get an understanding of her organization's or team's level of agility.

Future Steps: An Open Source Project for Agile Practitioners

On February 3rd, 2018, 20-plus people will join a hackathon to build an agility assessment framework based on taxonomy described below. The goal of the workshop is to provide the first version of a tool that empowers agile practitioners to measure agility.



'Agility' could be an assessment of an organization's suitability for agile practices, providing an idea of the necessary steps for an organization that decided to become a learning organization. Questions that come to mind are, for example:

1. Where are we now?
2. Where do we want to go?
3. What are the necessary steps to get there?
4. Design a plan how to get there

The Berlin hackathon will be an experiment. For example, I wonder if we can apply analytical thinking–such as measuring factors and calculating states—to complex social systems? Or will that approach turn out to be a dead end?

The Original Survey Questions

The 2017 agile maturity survey comprised of four questions:

1. What factors contribute to a team's growing maturity in agile practices?
2. What maturity levels do you see at a team level?
3. What factors contribute to becoming an 'agile' or a learning organization?
4. What maturity levels do you see at an organizational level?

In total, 86 people participated in the survey: 13 from the corporation I am currently supporting and an additional 73 participants from the Age-of Product mailing list.

Assessing Agility: The Preliminary Agile Maturity Indicators

From the answers, I derived the following taxonomy of indicators of agile maturity:

- People and teams: Autonomy, Mastery, Purpose
- Organizational Excellence
- Technical Excellence
- Communication & Collaboration.

The slides of the presentation are available on SlideShare.

People and teams: Autonomy

Self-organization:

1. Empower teams (Decisions, accountability)
2. Focus on outcome
3. Respect Scrum values (Commitment, focus, openness, respect, courage.)

4. Safety to raise & discuss issues
5. The team handles its own problems (No scrum mom.)
6. Supporting each other as team members (Bonding.)
7. Holding each other accountable (Agile is a team sport.)

Accountability (of the individual):

1. Choosing tools & devices (e.g. software)

People and teams: Mastery

Learning:

1. Short feedback loops (User tests, customer development)
2. Use of retrospectives
3. Continuous team coaching (Guilds, code mentors etc.)
4. Stakeholders live up to their responsibilities
5. Hands-on experience over credentialism.

Competence:

1. T-shaped people
2. Active knowledge sharing
    - Continuous learning
    - No withholding of knowledge
    - Knowledge sharing beyond the product and tech realm
3. Budget to attend conferences
4. Center of Excellence for Agile

Team building:

1. Cross-functional teams:
    - No dependencies w/ other teams,
    - End-to-end delivery capability
2. Stable, long-living teams
3. Support by an experienced scrum master

People and teams: Purpose

Inclusion:

1. Product discovery

2. Product roadmap creation
3. Release planning

Organizational Excellence

Culture

1. Embrace and celebrate failure (Validate hypotheses by running experiments)
2. Curiosity as a norm
3. Undogmatic attitude, live Shu-Ha-Ri
    - Transparency:
    - Share information and data at all levels,
    - No more gated information or information brokers

Leadership

1. Focus on innovation, quality and business value (No more HIPPOism.)
2. Supports of 'agile's way of working' fully
3. Enforces 'agile' as the core of the company culture
4. Respect for roles, principles, and processes (The 'real' PO.)

Management

1. Managers to servant leaders
2. Trust in people and teams
3. Provides tools and facilities necessary to become agile
4. Gemba and Kaizen become standard practices.

Organizational Design

1. Abandon functional silos for cross-functional teams
2. Remove redundant middle management layers (Flatten the hierarchy)
3. No more command & control, compliance-driven management
4. HR aligns with requirements forf self-organizing teams
5. The organization morphs into a team of teams.

Clear objectives

1. Shared vision among all actors
2. Clear strategy
3. Clear priorities.

Business value focus

1. Customer centricity mindset
2. Delivering business results
3. Shifting the IT focus business needs
4. From project budgets to product teams.

Technical Excellence

Engineering level

1. Built-in quality:
    - Code reviews,
    - TDD (Test automation, test coverage)
2. Pair and mob programming
3. Practicing Scrum, Kanban, XP.

Process level

1. DevOps: CI, CD (Deployment at will)
2. Regular cadence of releases
3. Identifying suitable metrics:
    - Lead time, cycle time,
    - Number of experiments,
    - Team health
4. Open sourcing code.

Communication & Collaboration

Trust & respect

1. Benefit of the doubt for colleagues
2. Safety to disagree
3. Honesty
4. Candid peer feedback.

Conflict resolution

1. Constructive disagreement (Disgree, but commit approach.)
2. Non-violent communication.

Collaboration

1. Zero tolerance for political games
2. No scripted collaboration
3. No incentives to withhold knowledge (Or information.)
4. No finger-pointing, no blame-game.

The Conclusion

Measuring elements of agility at an organizational or team level is nothing new. This 'agility assessment framework' approach, however, is new as it aims to be the first open-source and thus widely available tool for all agile practitioners. It is unlikely that the first planned workshop will deliver more than a rudimentary prototype of the agility assessment framework.

However, it will be a start to gather more insights by applying the framework to real-life work situations and take it from there. Hopefully, we will be able to establish a community around the 'agility assessment framework' in the future.

Agile Audit: How Is Your Agile Transition Progressing?

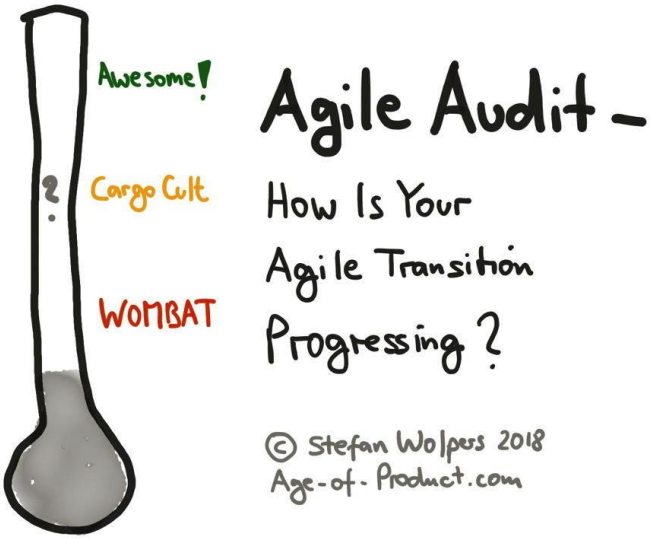
Introduction

Supposedly, becoming agile is a journey, not a destination. Which is a convenient narrative if the viability of your consultancy depends on selling men and materiel. The fuzzier the objective of an agile transition the less likely there will be an agile audit addressing the return on investment question the customer might have.

Moreover, a fuzzy objective such as 'we want to become an agile organization' is probably the reason for applying the same methodologies indiscriminately to every organization—a one size fits all approach for agile transitions.

However, what if not every organization embarking on a transition to agile practices is meant to become a teal organization or a holacracy? What if being late to the agile transition party is instead a deliberate choice than a manifestation of hubris, ignorance or leadership failure?

Read more on why feedback loops in the form of an agile audit are beneficial for organizations and teams alike.



Becoming Agile, Late Majorities, and Laggards

InfoQ applies the 'Crossing the Chasm' metaphor to engineering practices, thus covering a part of the agile movement to create learning organizations. Its recent 'Engineering Culture

and Methods InfoQ Trends Report - January 2018' found that new converts to Scrum, for example, will recruit themselves most likely from the late majority and laggards. (The early majority of organizations is already adopting Lean, Kanban, and Scrum derivatives.)

Regarding the late majority and laggards, my two working hypotheses are:

1. Not every — late majority or laggard — organization embarking on a transition to agile practices now would benefit from automatically following the beaten path of preceding organizations. Why strive to become a holacracy if there is no value for customers to be gained? (Think about markets that are highly regulated.)

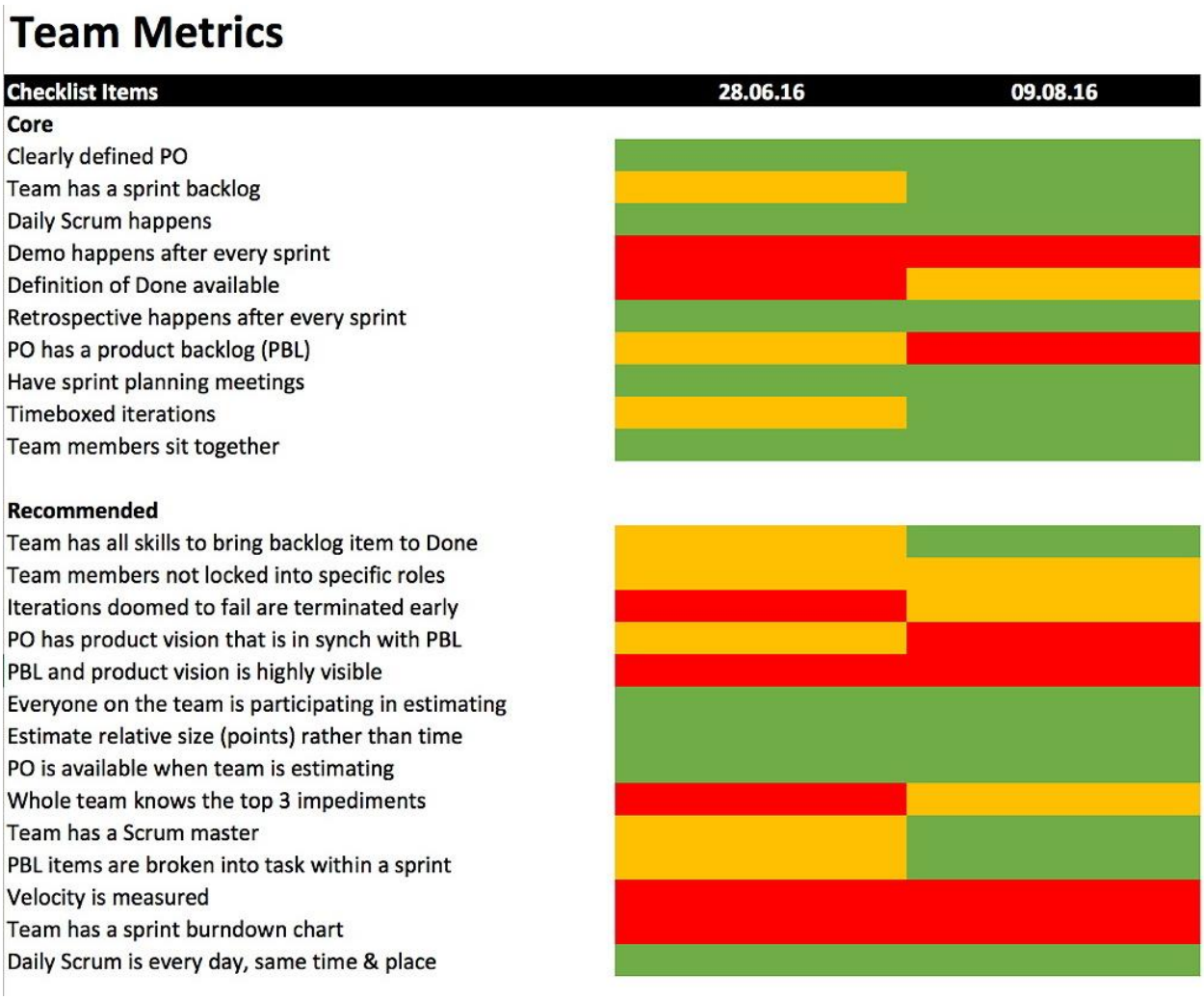
2. Many of the innovators, early adopters, and early majority organizations would benefit from external feedback loops in the form of an agile audit, too. The majority of these organizations are anyway operating somewhere on the product delivery side of the agile equation. They have never progressed to embracing business agility as an organizational concept. Moreover, they probably overspent on their agile transitions as they missed to define a goal at the beginning of their journey. Note, it is not in the interest of a consultancy supporting clients with agile transitions to make itself redundant in the process.

The DIY Agile Audit — Self-Assessment Tools

Ben Linders lists more than 50 self-assessment tools that provide some status check of a team or even an organization.

A lot of these self-assessment tools are suited to provide an initial data set to start a discussion with a team or the organization. (See, for example, my questionnaire Cargo Cult Agile: The 'State of Agile' Checklist for Your Organization.) If applied in a regular cadence, a change of the resulting values over time may support a better understanding of team issues as well as organizational issues and other contributing factors.

One of the best-known self-assessment tools is probably Hendrik Knibergs "Unofficial Scrum Checklist:"



In the example above, we were using a kind of estimation poker to answer each question with one of the three values green, orange, and red. The colors are coded as follows:

- **Green**: It worked for the team.
- **Orange**: It worked for the team, but there was room for improvement.
- **Red**: It either didn't apply, for example, the team wasn't using burn-down charts, or the practice was still failing.
-

If the resulting Scrum practices map is getting greener over time, the team is on the right track. Otherwise, you have to dig deeper to understand the reasons why there is no continuous improvement and adapt accordingly.

What most of these self-assessment tools are lacking, though, is alignment with an overall concept how agile transitions occur. Hendrik Kniberg's Scrum checklist has proven to be a useful tool providing insight into the progress of the adoption of Scrum practices at a team

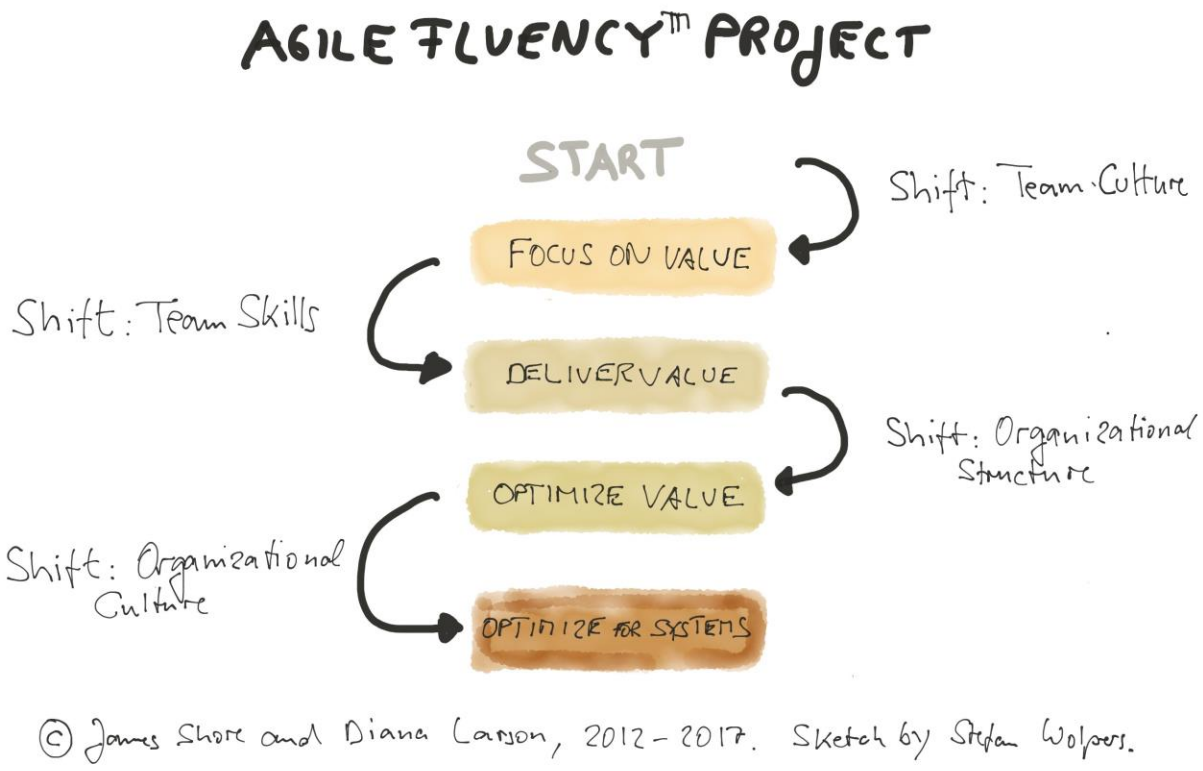
level. However, adopting Scrum, for example, is only a minor part of becoming an agile organization.

This alignment applies particularly to larger organizations where the excitement to experiment with agile ways of working is often sidelined both by the "what-is-in-for-me-syndrome" of the higher levels within the hierarchy as well as the risk mitigation necessity. This is where the Agile™ Fluency Model appears on the scene. James Shore and Diana Larsen initially described the model on Martin Fowler's blog back in 2012: Your Path through Agile Fluency— A Brief Guide to Success with Agile.

The Agile™ Fluency Model by James Shore and Diana Larsen

In October of 2017, I attended the Agile Fluency™ Gathering 2017 which happened to be in Germany. I wasn't sure what to make of the model in advance, but meeting dedicated agile peers in general and Diana Larsen and James Shore, in particular, seemed well-spent time to me.

The gathering started with two days of introduction to the Agile Fluency™ model stressing particularly that it is not a maturity model.



Instead, the steps that may look from the outside like maturity levels turned out to be fluency zones. Each zone constitutes a valid level of organizational agility. Also, a combination of elements of more than one zone can define an organization's agile fluency. That was my 'aha'-moment.

About 60 percent of 'agile' organizations seem to fall into zone one (Focus on Value), about 30 percent belong to zone two (Deliver Value), while five to ten percent can be assigned to zone three (Optimize Value). There is no valid data available on possible candidates for the zone four — Optimize for Systems — as there are only a few companies known to be in that zone. (For example, Semco, or Menlo Innovations.)

Agile Fluency™ Model Zone 1: Focus on Value

Zone 1 of the Agile Fluency™ model is centered around a shift in the team's culture. The team is now focusing on working on the most valuable things at all time although the road to value-based prioritization will likely be a bumpy one. Nevertheless, from a business

perspective progress is already visible. What we observe in this zone, for example, is the application of basic Scrum principles or a switch to Kanban.

These fundamentals will usually take two to six months to master, and approximately 60 percent of all agile teams across organizations fall into zone 1 of the Agile Fluency™ model. (Note that falling in zone 1 does exclude borrowing already some techniques from zone 2.)

Agile Fluency™ Model Zone 2: Deliver Value

Zone 2 of the Agile Fluency™ model addresses team skills, mainly being able to release at will, shipping product on a market cadence, capturing value frequently, and revealing obstructions early. All of this requires investments both in software craftsmanship — from extreme programming to clean code to DevOps — as well creating cross-functional teams. These steps will typically take from five to eighteen months to master, and approximately 30 percent of agile teams fall into zone 2 of the Agile Fluency™ model.

Agile Fluency™ Model Zone 3: Optimize Value

Zone 3 of the Agile Fluency™ model reflects the shift to improve the organizational value of a team by incorporating business expertise. In zone 3 we talk about applying techniques like Lean Startup or Design Thinking to make innovative product decisions. The team will reduce handoffs, improve the flow of work, and address organizational obstacles that impede the team's future success.

Depending on the nature of the organization, these fundamentals will take from twelve months to three years to master, and no more than five to ten percent of agile teams fall into zone 3 of the Agile Fluency™ model.

Agile Fluency™ Model Zone 4: Optimize for Systems

Zone 4 of the Agile Fluency™ model is all about optimizing systems by changing their organizational culture. Probably, this means moving toward sociocracy, to holacracy, or teal organizations. Teams understand business priorities and seek to collaborate with other teams to improve the overall value stream, thus propelling innovation in a self-organized manner.

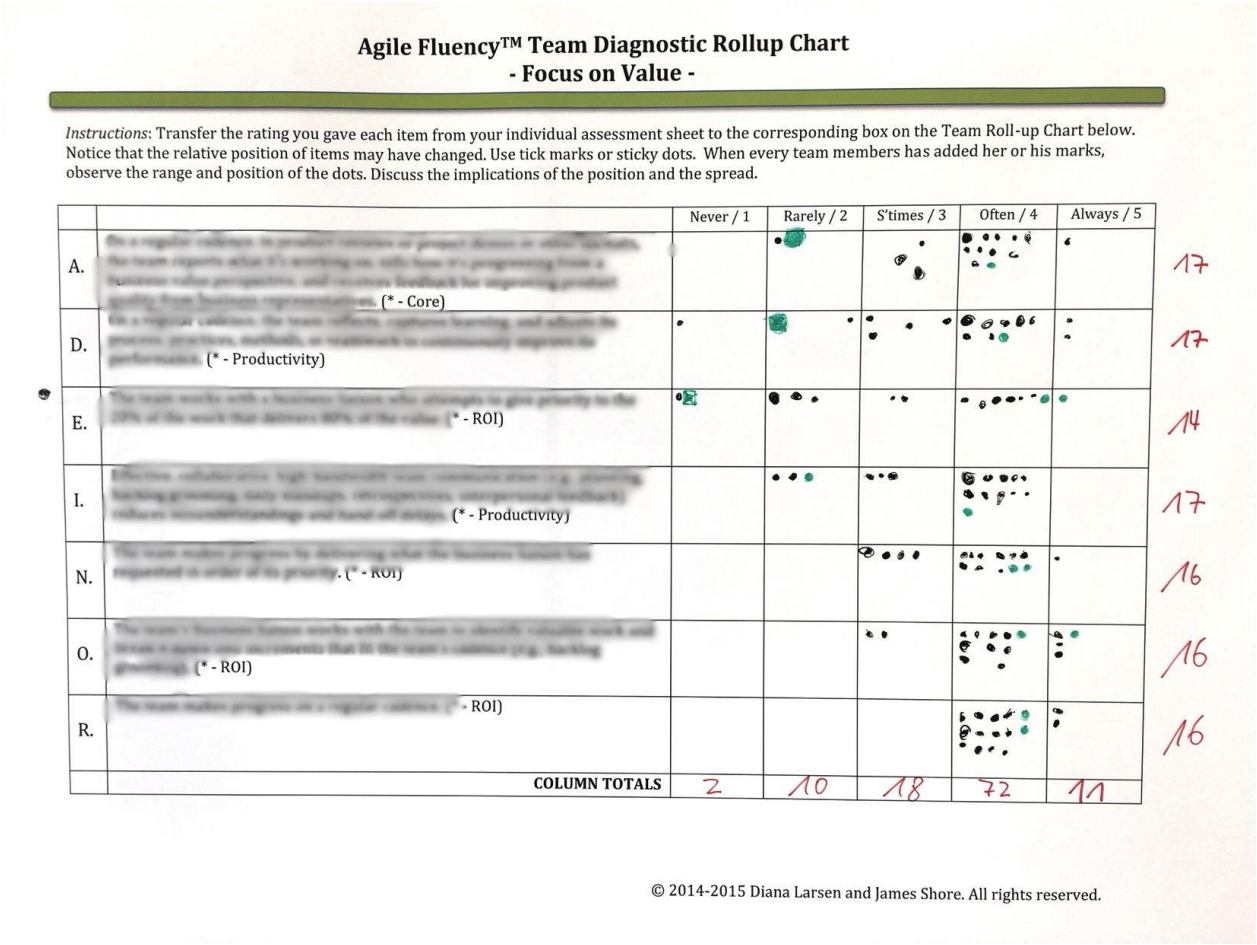
Besides a few (small) startups, there are only very few large companies known to be in zone 4 of the Agile Fluency™ model.

The Agile Fluency™ Team Diagnostic – An Agile Audit Tool

Based on the Agile Fluency™ model, James Shore and Diana Larsen developed a team diagnostic tool.

The Agile Fluency™ Team Diagnostic tool supports an agile team with the identification of the state of its agile journey as well as beneficial next steps. If applied in a regular cadence, patterns will emerge that provide a better understanding whether the team is on the right track improving its capability of delivering valuable products to the customers.

The tool is centered around a questionnaire comprising of seven questions each for the first three zones (from Focus on Value to Deliver Value to Optimize Value) that are answered by each team member. The answers are simple ranging from '1/Never' to '5/Always.' The facilitator then aggregates the results:

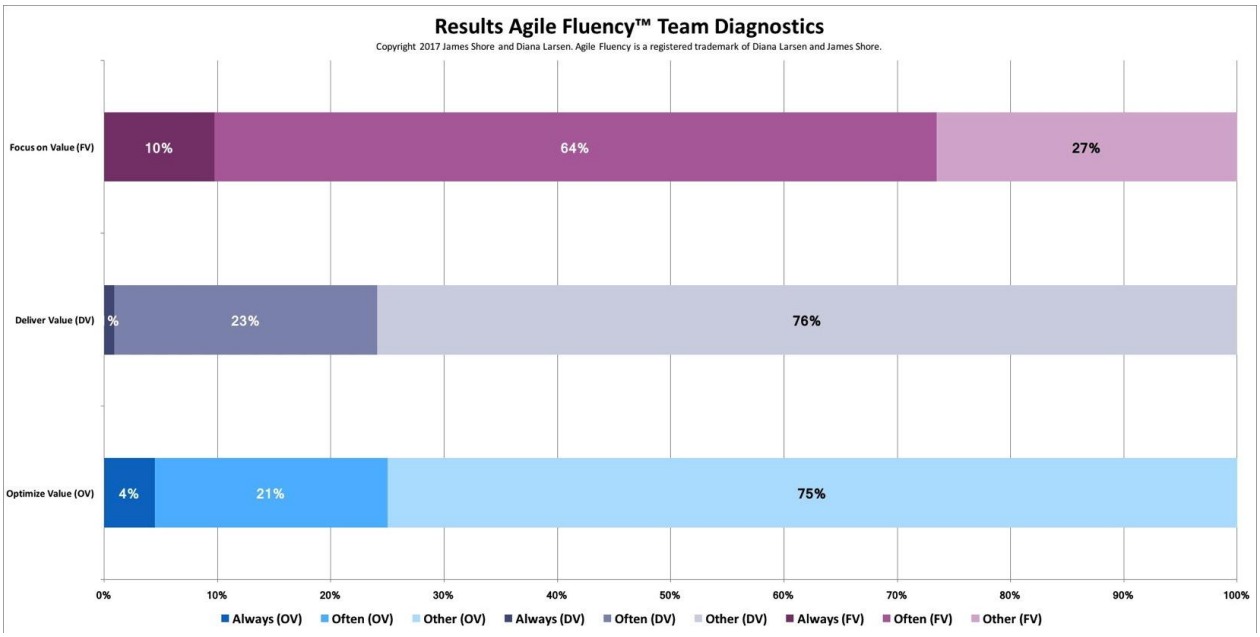


Agile Fluency™ Team Diagnostic Rollup Chart
- Focus on Value -

Instructions: Transfer the rating you gave each item from your individual assessment sheet to the corresponding box on the Team Roll-up Chart below. Notice that the relative position of items may have changed. Use tick marks or sticky dots. When every team members has added her or his marks, observe the range and position of the dots. Discuss the implications of the position and the spread.

		Never / 1	Rarely / 2	S'times / 3	Often / 4	Always / 5	
A.	[illegible] (* - Core)						17
D.	[illegible] (* - Productivity)						17
E.	[illegible] (* - ROI)						14
I.	[illegible] (* - Productivity)						17
N.	[illegible] (* - ROI)						16
O.	[illegible] (* - ROI)						16
R.	[illegible] (* - ROI)						16
	COLUMN TOTALS	2	10	18	72	11	

Based on the distribution of individual answers, the Agile Fluency™ Team Diagnostic tool provides a visualization of a team's current status:



A team is "fluent" in a zone when:

1. Everyone in the team rates the zone's core metric as '5/Always,'
2. At least 75% of the team members' responses (across all seven questions) are '5/Always,'
3. The composite (team) rating for six of the seven items '5/Always.'
4.

Based on this status according to the Agile Fluency™ model the team engages in a discussion to identify issues worth tackling to improve the team's fluency. (This is not different from an overall team retrospective.) The Agile Fluency™ Team Diagnostic exercise is then repeated in a regular cadence, probably every 8 to 12 weeks thus creating a pattern.

Once you run this practice with all teams regularly and you aggregate the team results you will create a data set over time that reflects the progress of your organization's agile transition — the Agile Fluency™ Team Diagnostic can thus become your agile audit.

Note: If you like to learn more about this agile audit tool please me know; I am a licensed Agile Fluency™ Team Diagnostic facilitator.

The Conclusion

Feedback loops have proven invaluable for any undertaking where value needs to be delivered, and risk requires to be mitigated. This fundamental agile principle applies not only to creating products and services. Of course, it does also refer to the process of becoming an agile organization itself.

Hence, feedback loops in the form of a regular agile audit ought to be a part of the toolbox of any agile transition. Not for the sake of collecting data but providing insight and thus guidance promptly for teams as well as the organization.

Organizational Issues

The Agile Workspace: The Undervalued Success Factor

Agile Workspace Means Choice Among a Diversity of Spaces

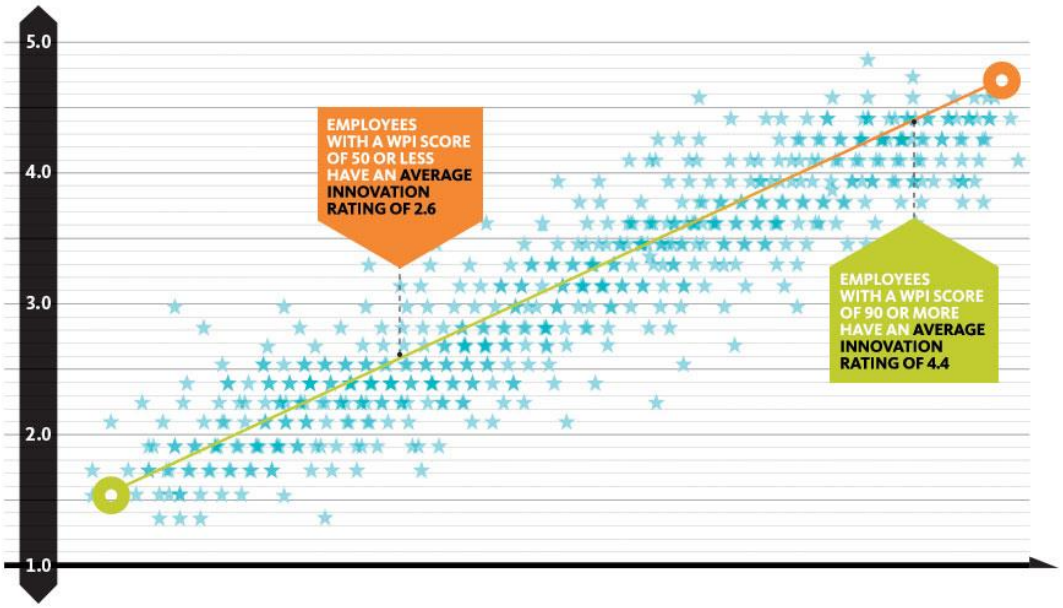
If you want your organization to become agile, adding more whiteboards to the workspace will not suffice. You have to abandon the idea that the workspace is an assembly line for white-collar workers. You need to let go Taylorism. We are now in the age of the creative worker.

To become agile – and reap its benefits such as becoming more innovative –, you need a diversity of workspaces to support all forms of creative work: focus, collaborate, learn, and socialize. Also, you have to let your creative workers choose which space is best suited for a task.

The Agile Workspace: A Worthwhile Investment

Why would you spend on creating a diverse workspace in the first place? Because creating it is a sound investment decision.

Based on their proprietary Workplace Performance Index® methodology, Gensler's "U.S. WORKPLACE SURVEY 2016" uncovers "a statistical link between the quality and functional make-up of the workplace, and the level of innovation employees ascribe to their companies".



Copyright notice: (c) Gensler.

In other words: If your organization is trying to become agile without investing in the workspace, it may realize a lesser return on investment. It may also fall behind a competitor that has invested in a diverse workspace for its creative workers.

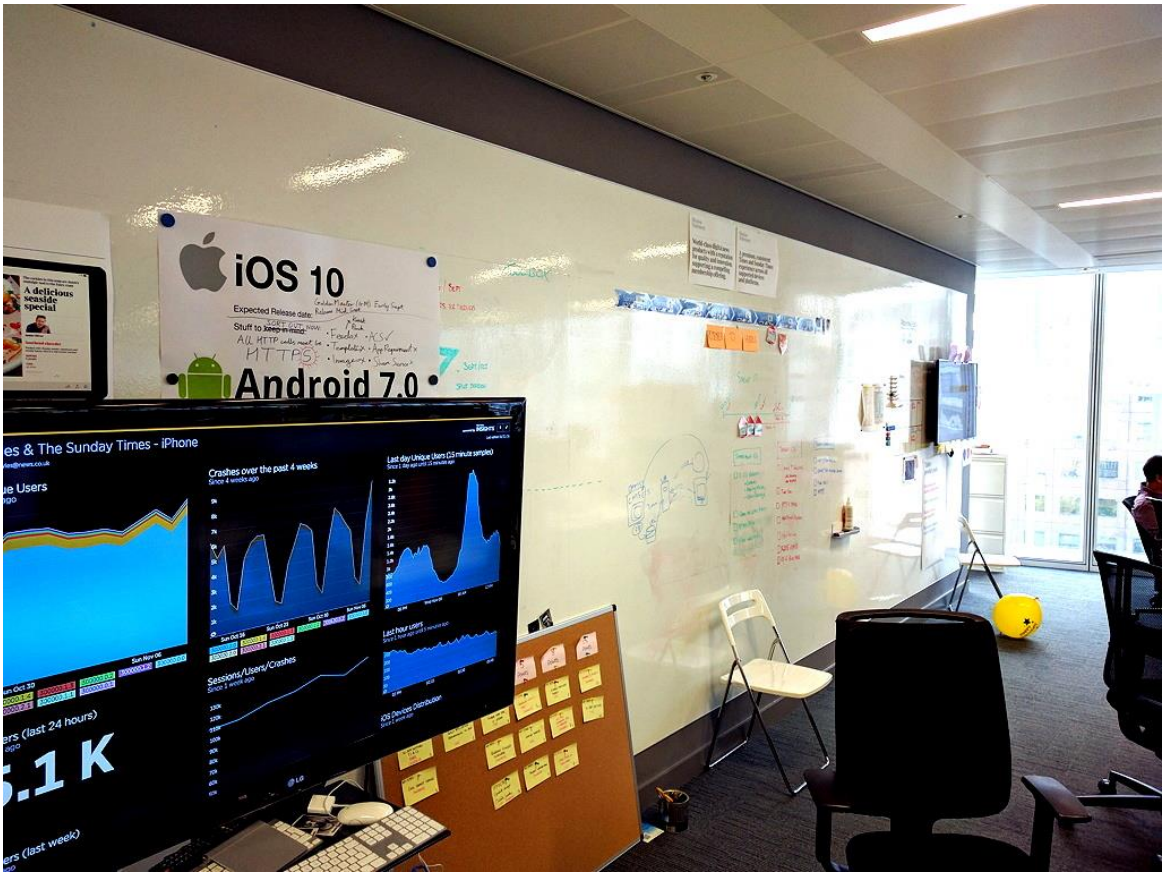
Download Gensler's survey for free.

Agile Workspace Anti-Patterns

Let's have a look at some issues in workspace design that contradict the needs of the creative worker:

- Open floor plans are often considered to be the ideal agile workspace as they nurture collaboration and help spreading news and information, thus keeping everyone in the loop. However, a lot of creative workers feel that their need for focus and deep work is challenged by the noise and sound level that often accompanies open floor plans. (Read more: "Just shut up and let your devs concentrate, advises Stack Overflow CEO Joel Spolsky".)
- Having to grab a new desk every morning: This bean counter brainchild is toxic for collaboration among members of agile teams. The policy is likely to scatter a team all over the place instead of co-locating it. It also deprives everyone of an important part of the psychological safety net – an individual desk – that fosters trust building and collaboration among team members.

- Scarce availability of whiteboards: Most walls within the workspace of an agile organization should be by default whiteboards to encourage instant collaboration. Glass or brick walls may be aesthetically pleasing but will impede the agile transition as they prohibit spontaneous collaboration.



Copyright notice: [(c) Adrian Kerry.]

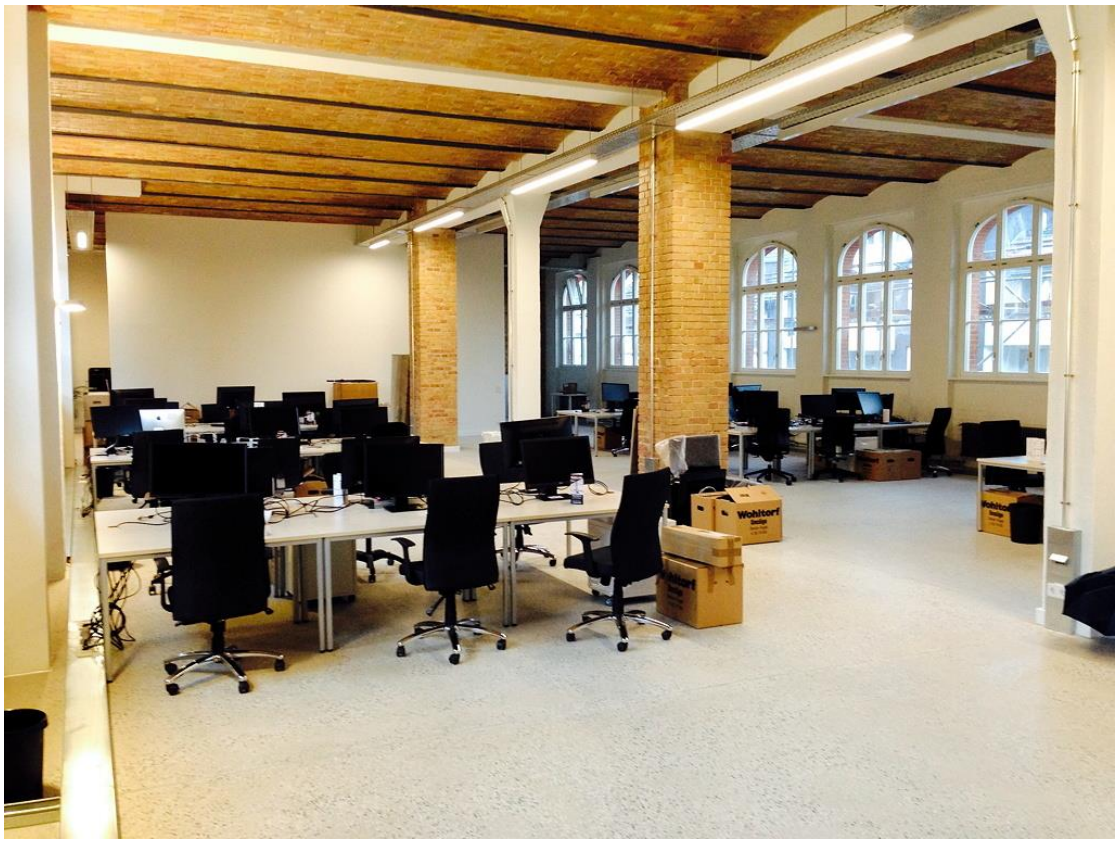
Team members who prefer working from home are a good indicator that your workspace is not up to the agile job. Those team members are by no means unfit for agile (team) work. They probably just need to get deep work done that they cannot accomplish in the office.

Another indicator for an unsuitable workspace is the widespread use of hearing protection devices or headphones by team members.

Being Agile Requires a Diversity of Workspaces

To become agile, your organization needs a diversity of workspaces to support all forms of creative work: focus, collaborate, learn, and socialize:

- Agile requires a large, flexible space for trainings, public ceremonies, e.g. sprint reviews, or workshops. If you want to run user story mapping workshops with ten of your stakeholders, for example, you will need at least 10 meters (or 30 feet) of whiteboard space. (Rule of thumb: one meter per workshop participant.)
- Form follows function: A stylish office may please the eye but does not necessarily satisfy the needs of the creative worker:



- Agile teams require defined team spaces to create a sense of togetherness, not just any area within an open space.
- Agile requires space for collaborating in small teams of 2 to maybe 5 or 6 people.
- Agile requires silent workspace to for deep, focused work.
- Agile requires space for informal, ad hoc meetings of 2-3 people.
- Agile requires social spaces – e.g. cafés – that encourage informal networking. Cafés provide the serendipity of meeting someone interesting unexpectedly.
- Being agile also requires a budget for regular offsite events, such as workshops, to support pushing people gently out of their comfort zone. Familiar settings become less stimulating for innovation over time. Utilize the hardwired (sabre toothed tiger-driven) need of humans to be more alert in unfamiliar places to your advantage.

How can a great agile workspace be created?

You create a great agile workspace by including the teams as early in the planning process as possible. Don't just present the concepts from architects or interior designers. Live up to valued agile principles – such as transparency, interaction, and inclusion – when your organization is designing an agile workspace.

Conclusion

Transitioning to an agile organization requires changing the available workspace in most cases. And buying a few additional whiteboards won't get the job done.

Becoming agile requires an environment that fosters all four forms of creative work: focus, collaborate, learn, and socialize. And it should be up to the creative worker to choose the right space for each task.

Last, but not least: Include your teams when designing an agile workspace. People care for what the help creating.

How to Create an Agile Community of Practice

How to Become an Agile Organization - When the Plan Meets Reality

Typically, the recipe for becoming an agile organization goes somehow like this: you need the commitment from the C-level to change the culture of the organization and thus its trajectory. You also need strong support from the people in the trenches who want to become autonomous, improve their mastery, and serve a purpose. Then - in a concerted effort - the top and the bottom of the hierarchy can motivate the middle management to turn into servant leaders. (Or, probably, get rid of them Haier style.)

Accordingly, an action plan often starts with hiring a consultancy to help figure out a more actionable roll-out plan, mostly comprising of training and workshops, initial team building activities, and probably some audits concerning financial reporting requirements, technology or governance issues.

What this kind of orchestrated initiative often neglects is the grassroots part of any successful change: provide room and resources to the members of the organizations to engage in a self-directed way with the change process itself.

A successful agile transition needs an agile community of practice.

The Purpose of an Agile Community of Practice

The purpose of an agile community of practice has two dimensions:

1. Internally, it serves in an educational capacity for agile practitioners and change agents. There is no need to reinvent the wheel at team-level; regularly sharing what has proven successful or a failure in the context of the transition will significantly ease the burden of learning.

2. Externally, the agile community of practice contributes to selling 'agile' to the rest of the organization by informing and educating its members. The members of the agile community also serve as the first servant leaders and thus as role models for what becoming agile will mean in practice. They bring authenticity to the endeavor.

Winning hearts and minds by being supportive and acting as a good example day in, day out, is a laborious and less glamorous task. It requires persistence - and being prepared not to take a 'no' for an answer but try again. Reaching the tipping point of the agile transition will likely be a slow undertaking with few signs of progress in the beginning. (Moreover, management tends to underestimate the inherent latency.)

A Portfolio of Services and Offerings of an Agile Community of Practice

Internal Offerings, Serving the Community

Improving the level of mastery of the members of the agile community of practice is not rocket science. My top picks are as follows:

- **Sharing is caring**: The hoarding of information is one of the worst anti-patterns of an agile practitioner. Hence share everything, for example, retrospective exercises LINK, to information resources (newsletter, blog posts, etc.) to working materiel and supplies. A wiki might be the right place to start.

- **Training and education**: Organize regular workshops among the agile practitioners to train each other. If not everyone is co-located, record the training for later use. (Webinar software has proven to be helpful with that.) If you have a budget available, invite the Marty Cagans to the organization to train the trainers.

- **Organize events**: Have regular monthly events for the agile practitioners and others from the organization and host meetups with external speakers. Make sure that all practitioners meet at least once a quarter in person for a day-long mini-conference.

- **The annual conference**: Consider hosting an organization-wide yearly 'State of Agile' conference to share lessons learned, success stories and failures.

- **Communication**: Use a Slack group to foster friction-less communication among the community members.

- **Procurement**: Find a workaround to allow non-listed suppliers to provide supplies such as special pens or stickies. (Probably, there is a freelancer or contractor among the practitioners who can help with that.)

External Offerings, Serving the Organization

Generally, what is working for the agile community of practices is also suitable for the members of the organization, probably with a different focus, though. Try, for example, the following:

- **Provide training**: Provide hands-on training classes in close collaboration with the change agents. Consider a less demanding format that a typical day-long training class - a focused one-hour class in the late afternoon may prove to be just the right format for your organization. (Tip: Avoid the necessity for participants to apply somewhere to be allowed to the class. That will massively improve attendance rates.) Also, consider offering a kind of curriculum that is comprised of several of those light-weight classes.

- **Communication**: Consider running a website or blog beyond the agile community of practice's wiki to promote the organization's path to becoming an agile organization. The best means I have encountered so far to foster engagement among change agents and early adopters is a weekly or bi-weekly newsletter within the organization.

- **Make 'agile' mandatory for new colleagues**: Educate all new hires on agile principles and practices to support the repositioning of the company.

- **Gain visibility**: Selling Agile to the organization to win hearts & minds is best achieved by making 'agile' tangible at a low-risk level for the individual. For example, organize regularly lean coffee sessions or knowledge cafés thus providing a safe environment to check this 'agile' thing out. Invite people directly to ceremonies, for example, sprint reviews - if you practice scrum - that might be of interest to them. (Guerilla advertising is welcome.) Lastly, why not offer an informal way of contacting agile coaches and change agents? Some people shy away from asking supposedly stupid questions in the open and may be hard to reach otherwise.

- **Provide transparency**: Occupy a space at a highly frequented part of a building or the campus to show what 'agile' is about and provide an overview of practices, courses, regular events, etc.

- **Host events**: Try to organize regular events for the organization, for example, providing lessons learned from teams that are spearheading the agile transition. Create a schedule in advance and stick to it. Perseverance is critical to fighting the notion that becoming agile is merely a management fad that will go away soon.

Overcoming Resistance to an Agile Community of Practice

So far, I have not yet witnessed open pushback from an organization about the creation of an agile community of practice. More likely, you will encounter complacency or ignorance at the management level. Sometimes, the budgeting process will be utilized - willingly or not - to impede the creation of a community.

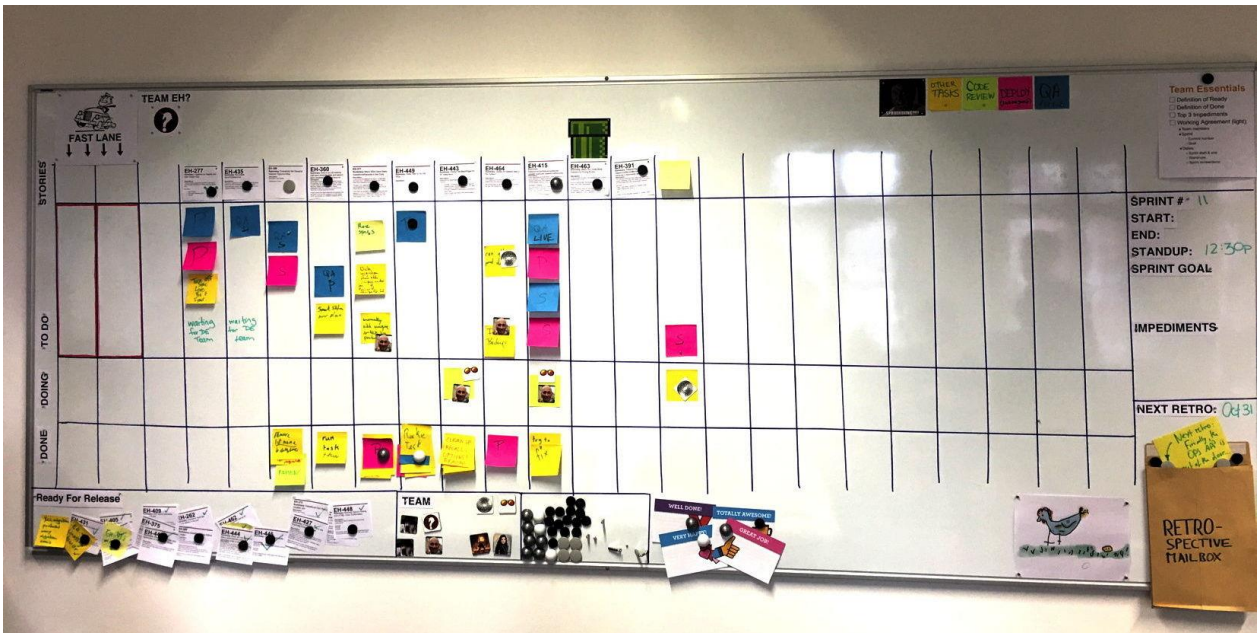
But even when you are financially restrained, there is still enough room to move the agile community of practice ahead. There are several services available for free that provide video conferencing and hosting, blogs, event organization, or newsletter services. In my experience, it is less a question of available funds, but you need to overcome your anxiety and get going without waiting for written approval from whomever. Assuming accountability as an agile practitioner by starting a community and thus moving the transition forward sounds agile to me.

Creating an Agile Community of Practice — Conclusion

Creating an agile community of practice is a vital part of the process of becoming an agile organization. It provides a lot of the groundwork that is necessary to convince the members of the organization that becoming agile is neither hazardous nor a fad but a trend and thus an excellent chance for everyone involved.

Create Whiteboards for Transparency and Collaboration

Whiteboards are magic as they support foundational agile principles such as interaction, collaboration, face-to-face communication, or transparency. They facilitate adapting to change, continuous improvement, and the self-organization of team. You can create meaningful software with a few index cards, pins, pencils, and a drywall—you do not need Jira or any other agile process tools.



Use Cases of Whiteboards

The first application of a whiteboard that comes to the mind of an agile practitioner is probably a sprint board. Too often, though, is our imagination limited to the available 'whiteboard space.' It turns out that whiteboards are not just well suited to display a sprint backlog or the workflow and workload of a team.

Whiteboards make excellent information radiators in general: from product backlogs, product roadmaps and experimentation and hypotheses backlogs, to visualization of technical debt or architecture diagrams, to impact mapping or user story mapping. Whiteboards provide instant transparency and invite collaboration with other team members or stakeholders. Also, whiteboards prove particularly useful in explaining team processes to stakeholders and new team members. By comparison to printed posters that seem to be both static by nature as well as 'approved' before printing, whiteboards invite to challenges the depicted status quo-as they can be changed both quickly and reversible. Lastly, a "whiteboard" in the sense of this article is not necessarily magnetic although those are more versatile. A drywall will do, as will a thick layer of cork.

Getting Practical: How to Create Whiteboards

There are four basic techniques to get offline boards:

1. Install prefabricated magnetic whiteboards on walls. It is the most expensive option with regard to cost per square meter. Moreover, often ready-made boards are either too small, or the boards do not fit the available wall space. Setting them up in the middle of the room can become a wobbly installation.

2. An excellent alternative to ready-made whiteboards is plain non-magnetic drywall. It is great for index cards as long as you are willing to use pins. You can achieve a similar effect with covering a wall with cork or other soft materials.

3. Lastly, another alternative is turning a wall into a magnetic whiteboard. A client recently transformed about 20 meters of drywall into a huge whiteboard - almost 3 meter high - at the cost of about $ 70 per square meter. (We run meanwhile also product backlog refinements in front a part of this mega-whiteboard.)

4. Buy movable whiteboards on wheels and place them somewhere in the room. (There is a variety of choices, from stylish to crude will-get-the-job-done.

Of course, windows and doors are probably usable as improvised whiteboards, too, depending on what is acceptable in your office. (I am aware that asking for forgiveness is presumably easier than asking for permission. However, if you are in doubt, I suggest asking the facility management. Some of those folks are really protective due to numerous safety and governance rules.)

Whiteboard Supplies

There are apparent whiteboard supplies such as magnets, pins, index cards or PostIts/stickies. While index cards need to be principally motived to stick to the surface of the whiteboard, stickies may work temporarily without additional means depending on the surface and the quality of the stickies. (I suggest using 'super-sticky' stickies anyway, but even those submit to gravity sooner or later.)

Whiteboard markers that can be removed from whiteboard without a cleaner should be the only markers around your office. (I am aware, that there are fewer colors available for whiteboard markers.) The trouble regularly starts with mixing whiteboard markers with permanent markers for flipchart paper. Honestly, I find it hard to understand why the latter is still purchased given the high removal costs if they are used on whiteboards without being removed immediately. (Have a bottle of cleaner handy for that purpose; the ones containing alcohol are usually up to the job.)

Drawing lines on whiteboards, for example, to mark columns or swim-lanes can be a tricky thing. Using whiteboard markers is not helpful as you will need to redraw them often. Also, team members tend to wipe lines off the whiteboard when working with the board. While it is easy to remove whiteboard markers from whiteboards, removing a black or red whiteboard marker from a white shirt or blouse is a different beast. My favorite line marking tools is hence a very thin, flexible yet sticky tape that is used for painting jobs.

Another essential component of useful whiteboards is magnetic avatars. (We have them printed by a service provider.) We use them for every team members to mark the tasks someone is working on during a sprint. We also use them on the product backlog board to match tasks with teams. (We have a unified product backlog that multiple teams use.) Finally, we use them to visualize the architecture of the application.

How to Use Sprint Boards

Given that whiteboards probably are most often used for sprint backlogs or Kanban boards, here are some practices that have been useful in the past:

- **Team name**: Put the team name or team logo on the board.
- **Team members**: Create a list of team members. If you use avatars, match names and avatars here.
- **Basic dates of the current sprint**: Start with the sprint number, start and end date, the standup time, and probably other ceremonies as well.
- **Sprint goal**: What are we fighting for this time? Make it visible prominently, written in large readable letters.
- **Swim-lanes**: Create swim-lanes to make information intake easier: These can be regular lanes, or probably your team likes to separate non-functional tasks and bugs from other tasks. It may be useful to have one swim-lane per tasks to avoid cluttering the board with unaligned stickies or cards.
- **Use templates**: Standardize the cards or stickies on the board by creating a basic design. This design applies to color-coding cards and markers. For example, always use the same color for bugs and write or print the ticket number at the same spot on the index card or sticky. Reduce the cognitive load and thus make it easy to read a card or sticky from a distance.
- **Daily scrum**: Run standups in front of the boards and move cards. (This should be a low-brainer but does not seem to be one. There is a reason why salespeople want you to touch new clothes before buying them. The resulting haptic experience already instills a level of perceived ownership. Putting an avatar to a card, moving a card makes it your card, too, supporting self-organization and creating a sense of accountability.)
- **WIP - limit the work in progress**: There are several ways to limit the amount of work in progress physically. For example, limit the number of magnetic avatars and make it a rule that you cannot work on a ticket without putting an avatar on the stickies. Or limit the available space to place cards - no stacking of cards allowed, of course. Mark stickies with red dots that are not moving to visualize the work item age. This way, the team can reach out to remove the impediment that is preventing progress in time.
- **Leading system**: If you are using offline and online boards, make sure that there is a leading system and that the boards are synced regularly, for example, right after the standup. This sync is a job of all team members, not a task of the scrum master, by the way.)

Whiteboard Anti-Patterns

As always, you can observe anti-patterns when teams use tools from the agile toolbox. In the case of whiteboards, for example, the most critical whiteboard anti-patterns are:

- **Abandoned boards**: A team started using a board and then stopped doing so, leaving the old board to rot.
- **Dropouts**: It is necessary to remind some team members regularly to take care of the board(s).
- **Lack of transparency**: Not all cards are placed on the board; there is a shadow accounting, blurring the picture.
- **Lack of motion**: Stickies are not moved regularly, and hence your team experienced sync failures with the other boards. (If this whiteboard anti-pattern persists it renders both boards unusable.)
- **Leading system ignored**: The working agreement on the lead system is not respected by every team member.
- **WIP limit ignored**: The team members do not honor the working agreement - reflected in the design of the board - on the work-in-progress limit.

Conclusions

Whiteboards are one of the most versatile tools from the agile toolbox: simple to create, even quick to improvise, radiating information, and inviting everyone to collaborate and thus becoming a part of the solution.

10 Proven Stakeholder Communication Tactics during an Agile Transition

Stakeholder communication: It is simply not enough for an agile product development organization to create great code and ship the resulting product like a clockwork. You also need to talk about it, particularly in the beginning of your agile transition. Marketing the agile journey of product and engineering to the rest of the organization—and thus getting their buy-in—is a critical success factor to step up the game: You want to become agile, not "do agile".

So, learn more about ten proven stakeholder communications tactics that contribute to making this happen.

Stakeholder Communication Channels During an Agile Transition

Do good and talk about it—a simple necessity, particularly if your agile transition is supposed to be embraced by the whole organization over time. Deciding early in the process how to communicate with internal stakeholders usually makes the difference between "doing agile" and "becoming agile" in the end.

Keep in mind that a lot of stakeholder ego, as well as personal agendas, is tied one way or another to executing "the plan". And you're trying to sell them that quitting this plan will turn out to be mutually beneficial.

A quick mental exercise: Try walking in your stakeholders' shoes, and ask yourself: Would you entrust your career a bunch of hoodie-wearing nerds, promising a big reward?

Developing empathy for stakeholders is particularly relevant, when engineering and product don't have the best standing within the company at the beginning of the transition. The good news is, that no matter whether both are perceived as a black hole by others, a well-orchestrated communication strategy has a good chance to win over the rest of the organization.

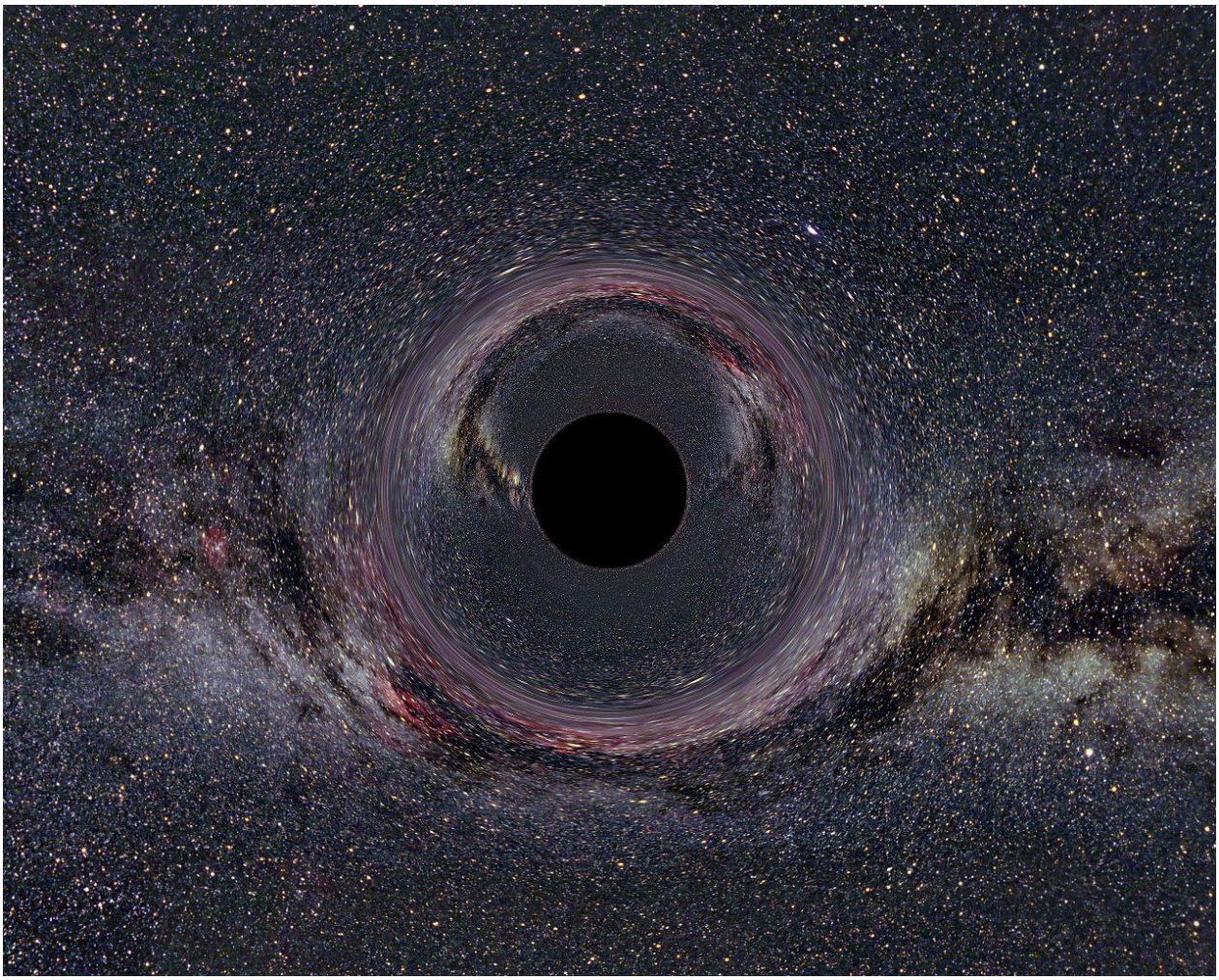


Image from wikimedia.org

Judging by my experience, the following stakeholder communication tactics have proven useful, if actively pursued from the beginning:

I. Agile Ceremonies

Sprint Reviews

A great opportunity to show to the whole organization what value has been delivered during recent sprint(s) and align the stakeholders with the upcoming sprint(s).

The cadence of this event depends on the size of your engineering organization, if the teams' sprints are aligned, or whether you're releasing several times a day anyway.

A weekly sprint review tends to wear off quickly, as not much novelty can be provided. (Except you've just started building something from scratch, then a weekly demo is great for team-building in general and creating a common spirit. You need to celebrate every victory.)

The basic rules are simple:

1. Lead by example: The engineering and product team should be present. Why would a stakeholder invest her time, if your buddies don't consider the sprint review a worthwhile investment of their own time?
2. Show, don't tell. Sprint reviews are a zone free from death by PowerPoint
3. Invite stakeholders to take the helm
4. 5 minutes per team are usually sufficient
5. Initially, it might be required to lure stakeholders to the sprint review. Bribe them, if you have to: Giving a few stories points away during the review to those who attend and can present a convincing issue works well. (But don't make this hack a habit…)

Further reading: Stop Calling Your Sprint Review a Demo—Words Matter!

Daily Scrums

Invite stakeholders to the daily Scrums, once the teams feel comfortable with the idea, to passively participate. Be warned, though, that you need to be firm in dealing with assertive stakeholders, who otherwise might try to take over the stand-up and turn it into a reporting session. If a team does not warm up to the idea, restrain yourself from making the invitation.

II. Educational Initiatives

Aggregate Information in Dashboards

„When you put problem in a computer, box hide answer. Problem must be visible!" (Hideshi Yokoi, former President of the Toyota Production System Support Center in Erlanger, Kentucky, USA.)

In that spirit, be transparent with all information, but visualize it in a way that stakeholders can actually make sense of it. (Often referred to in an agile context as "information radiators".)

Usually, it is more helpful to aggregate information across teams in a sort of stakeholder dashboard than having a board for each team. (Tracking the movement of sub-tasks, for example, normally proves to be a bit too granular for stakeholders.)

The advantages are plenty: Stakeholders rarely read reports, but they are willing to have a look at dashboards, and they appreciate a chat with product owners and engineers. On the plus side: It is a controlled and safe environment. And you can choose the venue, too: Put the dashboards where stakeholders can see them.

Be careful with simple burn down charts, though. Those easily get torn out of context and might cause a Pavlovian reflex with some stakeholders, triggering urges to micromanage teams. Further reading on this matter: Scrum: The Obsession with Commitment Matching Velocity.

Write Release Notes

Really helpful, but remember: Bait the hook and feed the fish. And the fish usually doesn't click on Jira links to learn about recent successes.

You need to tell a story. And a story has a hero (your product, team, organization…), a villain, an obstacle, and how the hero overcame the obstacle—in short: a story arc.

Invest some time to get the release notes right and they will positively influence the standing of product and engineering within the organization. (Rumor has it, they can be even fun to read.) Further reading: Slack: A little thing about release notes.

Institute Ambassadors

Start training interested individuals from the other departments' operative trenches—identified in a collaborative initiative with the respective department heads—to act as liaison officers to product and engineering. Ambassadors collect feedback, track feature requests, and check bugs prior to reporting.

Meet with them on a regular schedule, perhaps even weekly. They're excellent sparring partners, often act as your proxies within their own departments, and are well liked participants in user-tests.

A note of caution: Don't bypass reluctant heads, though. That hack might back trigger and result in the opposite effect of the whole idea.

Organize Training Workshops

Invite other colleagues to a training workshop, and teach them hands-on how a product is build nowadays: Lean startup, user story mapping, versioning, prototyping—you name it. Apparently, this kind of workshop does not include coding. Prototypes are built with paper, crayons, pencils and a prototyping app, for example from InVision.

Find a post on lessons learned when working with marketing people, sales and customer care agents, and other bloody beginners here: App Prototyping with Absolute Beginners.

III. Regular Meetings

Offer AMA Sessions in the Form of a Lean Coffee

Organize regular ask-me-anything session about agile practices, that is addressed to your colleagues outside the product development organization. A Lean Coffee "…is a structured, but agenda-less meeting. Participants gather, build an agenda, and begin talking. Conversations are directed and productive because the agenda for the meeting was democratically generated."

It is the ideal format to not just communicate your message, but also to identify those interested in the approach in general. They will probably become valuable allies at a later stage.

Working Operationally in Stakeholder Departments

Everyone–developers, QA engineers, product managers to mention the obvious–should regularly work, for example, in customer care. Nothing can create rapport more effectively than joining your customers in their operative work, while having your own dog-food.

In a very short period of time, your product backlog, your user story creation and prioritization process will become more aligned with solving real, and quite often trivial, problems. The whole product development organization will start rising from an anonymous group to valued colleagues.

Why? Because you serve in their trenches, too. A user story or a bug report is now no longer a ticket with a number in Jira, but it is associated with a name, and a face, and a story. Stakeholders will also learn more about your background, and why solving something that looks trivial to them might cause a major engineering headache.

Note: I am not referring to other regular meetings at stakeholder level here, for example such as portfolio management, product roadmap planning, or user story mappings, as those would exceed the scope of the post.

IV. Media Channels

Daily Newsletter

Create a daily newsletter of 5 to 6 of the most important news related to your company, and your industry. Throw in some startup & technology related posts—Elon Musk's part 2 of his manifesto would qualify for that—, for example, as well as an occasional post from product or engineering. Or the company blog.

Treat it as a real newsletter and use perhaps the free plan of Mailchimp for that purpose. You will need its analytic capabilities to optimize the newsletter over time. (You can do so by checking click-rates of popular links, or run different headlines.) In other words: Be useful to your colleagues.

Opening rates tend to be above 40% and can go higher, if word gets around that the CxO level is actively reading the newsletter. Encourage people to provide you with interesting links, and point out to those contributors within the newsletter editions. Sometimes, you might be lucky and even start a friendly battle between individuals to provide useful links. To my experience, this exercise will not require more than 30 min per day.

Side-effect: The PR department might be irritated (in the beginning). Just ignore them…

Product & Engineering Blog

Start blogging about your daily work: Technologies, processes, methodologies, frameworks, in other words: How you manage to ship a high-quality product, day after day. It greatly contributes to other components of the communication strategy sketched above if you can link to a detailed blog-post.

A great example from Berlin is Zalando's tech blog: "Visit our tech blog to learn more about the engineering and ideas driving Zalando's exciting work. Here's where we post technical talks and tips, news about our open source projects, event photos, and much more."

A product & engineering blog is also a corner stone of a great recruiting strategy, going hand-in-hand with regular events, thus crossing the chasm from online to face-to-face communication with like-minded people you would like to hire.

Conclusion

If you fail at communicating your agile transition in the right way to internal stakeholders, "Agile" might suffer the fate of being regarded as a mere local process, while the rest of your organization stays entrenched in silos and command & control structures.

Once that state is reached, resisting any future agile improvement attempts—waving the banner of "we tried in the past, and it isn't working in our organization"—often become the prevailing attitude for those, who are not supportive of agile and lean practices. (Read more in: Why Agile Turns into Micromanagement.)

This is the reason, why considering the right stakeholder communication should be addressed on day #1 of your agile transition journey.

Roles, Recruiting and Teambuilding

The Scrum Master in 70 Theses

Introduction

The following 70 scrum master theses describe the role of the scrum master from a holistic product creation perspective.

The scrum master theses cover the role of the scrum master from product discovery to product delivery in hands-on practical manner. On the one side, they address typical scrum ceremonies such as sprint planning, sprint review, and the retrospective. On the other hand, the scrum master theses also cover, for example, the relationship with the product owner, they deal with agile metrics, and how to kick-off an agile transition, thus moving beyond the original scrum guide.

The Role of the Scrum Master

This first set of the scrum master theses addresses her role in the scrum process:

- Scrum is not a methodology, but a framework. There are no rules that apply to every scenario — just practices that have worked before in other organizations.
- Successful practices of other organizations cannot simply be copied to your own. Every practice requires a particular context to work.
- You need to determine for yourself what works for your organization — which is a process, not a destination.
- The role of a scrum master is primarily one of leadership and coaching. It is not a management role.
- A scrum master should recognize that different stages of a scrum team's development require different approaches: some, teaching; some, coaching; and some, mentoring.
- A scrum master should be familiar with the Shu-Ha-Ri concept of learning new techniques.
- A scrum master's principal objective should be to remove themselves from daily operations by enabling the scrum team to be self-organizing and self-managing.
- Being a scrum master does not entail, and should never entail, enforcing processes.
- Scrum is not designed for bean counters, although some metrics are helpful in understanding the health of a scrum team. Generally, insisting that the team achieve specific KPI (for example, commitments vs. velocity) does not help.
- Scrum doesn't elaborate on the process that enables a product owner to add valuable, usable, and feasible user stories to the product backlog. Product discovery

using the Design Thinking, Lean Startup, or Lean UX methodologies may help, but in any case, a good scrum master will want the scrum team to be a part of this process (whether participating in user interviews or running experiments).

- A scrum team's communication with stakeholders should not be run through a gatekeeper (e.g., solely through the product owner) because this hurts transparency and negatively affects the team's performance. Sprint reviews, conversely, are a good way to stay in close contact with stakeholders and to present the value delivered by the team during each previous sprint.

Product Backlog Refinement and Estimation

The second set of the scrum master theses focuses on the importance of the product backlog refinement:

- Estimation and backlog refinement are essential tasks for every scrum team. Although the product owner — at least officially — is in charge of keeping the product backlog at 'peak value delivery', they need the assistance of the entire scrum team to do so.
- A cross-functional and co-located scrum team working independently of other teams is an ideal scenario. The reality is that most scrum teams will often be dependent upon deliveries from other teams (e.g., API endpoints) and deliverables from the UX or UI department.
- There are two essential ingredients for good scrum team performance:
    - *Writing user stories as a team*: When something should be built, the product owner first explains why and provides the necessary background (i.e. market intelligence, results from experiments, user interviews, statistical data). Writing user stories, then, is a corresponding and collaborative effort involving the entire scrum team. The process should create a shared understanding of what will be built and for what reasons (the product owner providing the 'why', the scrum team detailing the 'how', both defining the 'what'), and a shared sense of ownership among team members.
    - *Sharing a 'definition of ready'*: To ensure a flow of well-drafted user stories for the development process, the scrum team, and the product owner should consider agreeing on a definition of ready for these user stories. This definition is an agreement about what needs to be provided for a user story to be considered ready for estimation. If one of the defined requirements is not met, a user story isn't ready for estimation. A user story without a previous estimation is an unknown entity and, therefore, not ready to be made part of a sprint backlog because a scrum team can't commit to an unknown entity in a sprint. Consequently, the scrum team must learn to say 'No'. (However, beware of utilizing the definition of ready as a kind of stage-gate process. That constitutes an anti-pattern.)

- A well-refined product backlog probably has user stories detailed for about two or three sprints, and probably less than half of these stories conform to the scrum team's definition of ready. There may also be additional user stories that no one except the product owner is working on.

Sprint Planning

The third set of the scrum master theses covers the sprint planning:

- It used to be that a product owner would explain high-value user stories in a product backlog to the scrum team during sprint planning. The team would then turn these into more detailed user stories, and estimate the subsequent stories. There is now, however, a consensus among agile practitioners that working on these high-level user stories in separate backlog refinement and estimation meetings — before sprint planning — improves the quality of the stories and thus the outcome of the team's work.
- Sprint planning shall create a sense of ownership among a scrum team's members by enabling them to make a valid commitment to the items in the sprint backlog. However, this only happens if a scrum team's uncertainty about the quality of the user stories they're receiving is eliminated. To relieve the scrum team from this uncertainty, a scrum master should support the product owner in running weekly product backlog refinement and estimation sessions, only allowing into sprint planning those user stories that meet the team's definition of ready standard.
- Sprint planning should normally be divided into two parts:
    - *Sprint planning I*: During the first part of sprint planning, a product owner presents to the scrum team the product owner's choice of the most valuable user stories from the product backlog as a ranked list. The team then selects from the top of the list down those stories it can commit to delivering by the end of the sprint — taking into consideration their present constraints including, for example, available capacity, or the required technical tasks such as refactoring that need to be addressed during the same sprint.
    - *Sprint planning II*: During the second part of sprint planning, the scrum team adds detail to the user stories in the sprint backlog (e.g. splitting the stories into tasks, identifying parts of the stories that need further clarification, and agreeing on who will be working on what tasks). The product owner does not necessarily need to participate in this second part of sprint planning but does need to be available to answer questions that the team may have.
- If user story preparation is handled well, an entire sprint planning session might be completed within less than two or three hours.

- Productive sprint planning meetings require a healthy scrum team. Dysfunctional teams will not achieve the level of cooperation required. A sprint planning with dysfunctional teams will only result in a futile and painful exercise.
- A scrum team should usually avoid allocating more than 80% of their capacity to new tasks — including user stories, technical tasks, bugs, and probably spikes. Flow theory shows that a 90% or higher allocation of available capacity will not lead to a team achieving their peak performance.
- Bugs, refactoring, and technical research require regular attention to avoid building-up technical debt. An effective scrum team allocates at least 25% of their capacity to these tasks.
- Incomplete and poorly prepared user stories seriously hamper the effectiveness of a scrum team. These stories should never be selected for the sprint backlog, but instead sorted out during product backlog refinement and estimation meetings.

Standups

The fourth set of the scrum master theses addresses standups:

- Standups are meetings well suited to discuss a current sprint's progress: is all going as planned, or does the scrum team need to adjust?
- Standups are a convenient time for a scrum team to meet and communicate with a project's or product's stakeholders.
- Standups cannot fix, among other things: a dysfunctional organization, a dysfunctional scrum team, an inadequate product backlog, a sprint planning session gone wrong, low-quality user stories, or a missing product vision.
- Standups are valuable if the scrum team is already collaborating well and the basics — such as the product backlog, and sprint planning — are in order.
- The more experienced a scrum team, and the better the internal communications, the more a standup will seem a time-consuming ritual of little value.
- An advanced scrum team may consider virtual meetings instead of real meetings using, for example, a Slack channel.
- A two-person scrum team does not necessarily need a formal standup — meeting for coffee would be a practical alternative.
- There is something wrong with a scrum team whose members do not communicate impediments before each standup. It's possible they're acting more like a group of people being in the same place at the same time pursuing a similar goal than a scrum team.
- Standups are not reporting sessions for the benefit of product owners or participating stakeholders.
- Offline boards are valuable: physically taking a card and moving it instills a sense of ownership of a user story. (This is the same mental model why sales-people want you to touch the merchandise before a making purchase.) If you must let go of either an online or offline board and you are a co-located team, consider letting go of the

online board.

Retrospectives

The fifth set of the scrum master theses deals with retrospectives:

- Retrospectives should encourage self-expression, thereby making it easier for a scrum team to uncover the concerns and frustrations that its members may be harboring so that strategies may be devised to overcome them.
- Retrospectives will only improve a team's collaboration and performance if the team considers these meetings a safe place to provide honest and constructive feedback.
- The blame game is not helpful. During a retrospective, the members of a scrum team should focus on how to improve a situation — and avoid blaming one another.
- Some scrum teams always include the product owner in their retrospectives, while other teams insist that the product owner should be expressly invited.
- It's best not to hold retrospectives at a team's workplace. Distance makes it easier for team members to reflect on the sprint. It's also helpful to regularly change locations for the meeting. Being in a new locale helps to prevent boredom (and team members 'checking out').
- The format for a scrum team's retrospectives should be changed regularly. The same format should not be run more than twice.
- Smartphones, tablets, and laptops should not be permitted at retrospectives so that the members of the scrum team are not distracted, and can focus on contributing to the meeting.
- All issues, concerns, and frustrations, should be documented — even if just temporarily using sticky notes. Though it's always better to keep a formal document or file.
- According to Diana Larsen and Esther Derby in their book "Agile Retrospectives: Making Good Teams Great," there are five stages to running a retrospective:

    - Setting the stage,
    - Gathering data,
    - Generating insights,
    - Deciding what to do,
    - Closing the retrospective.

- A retrospective should set SMART goals for action items (the tasks to be done):

    - Action items should be specific and measurable ("do X more often" does not meet that criteria),
    - A single member of the scrum team should be made responsible for each action item,
    - Each action item should include an estimate of when results can be expected,

    - Action items should be placed on a board to make tracking progress visual and more prominent.

- Every new retrospective should start with reviewing the status of the action items decided upon during the previous retrospective.

Agile Metrics

The sixth set of the scrum master theses addresses on agile metrics:

- The purpose of metrics, generally, is to understand a current situation better and gain insight on how it's likely to change over time.
- A metric is a leading indicator for a pattern, providing an opportunity to analyze the cause for change — and act appropriately in due course.
- Metrics in an agile context are not used to manage, and certainly not micromanage, an individual (particularly the creative worker) — contrary to traditional command-and-control management structures.
- Metrics in an agile organization should be used to provide the scrum team — agile practitioners all — with insights on how to continuously improve, helping them achieve their goals:

    - Agile practitioners strive for autonomy, mastery, and purpose as explained by Daniel Pink.
    - Agile practitioners address personal development with metrics by applying methods like Objectives and Key Results (OKR).

- The experienced agile practitioner realizes that autonomy and accountability are equally important for self-organized scrum teams. Without metrics, both autonomy and accountability are limited.
- The metrics most suitable to agile reflect either a team's progress in becoming agile or the organization's progress in becoming a learning organization.
- Both qualitative and quantitative metrics are used:

    - Qualitative metrics typically reveal more than quantitative metrics when applied to the scrum team.
    - Quantitative metrics provide more insight than qualitative metrics when applied to the organization.

- Any metric used for agile must be tailored to the organization.
- The metrics that the scrum master should be tracking are only those that apply to the scrum team. Metrics that measure the individual should be ignored.
- A metric's context should always be recorded to avoid misinterpretation.

- Parameters that are easy to follow should not be measured for that reason alone — especially if a report is readily available in the project management software being used.

How to Kick-off a Transition to Scrum

The seventh set of the scrum master theses covers agile transitions:

- There is no checklist or master plan readily available, or that could be made readily available, that would ensure a successful transition to agile practices such as scrum. This also applies to SAFe, LeSS, Nexus, DAD, XSCALE, or the so-called 'Spotify methodology.'
- The 'best practices' of and 'lessons learned' by other organizations during their transition to scrum may indicate a direction to take when transitioning, though the context of their transition may not be comparable: what worked for Spotify may not work for General Motors.
- Every transition should start with understanding the 'why': why should the organization become agile?
- Reasons typically given by management for transitioning to scrum and other agile practices include:

    - Making the organization more efficient;
    - Helping the organization deliver faster; and
    - Improving the predictability of delivery dates.

- The recognized benefits of transitioning to scrum and other agile practices are:

    - Outperforming competitors by creating a learning organization;
    - Creating a great workplace culture by providing room for autonomy, mastery, and purpose; and
    - Mastering continuous product discovery and delivery (thus minimizing risk).

- Agile practices' benefits need to be sold to an organization before beginning a transition — agile is not everybody's darling, and personal agendas will be affected by a successful transition.
- A transition will encounter inertia and resistance to change directly proportional to the size of the organization.
- How an agile transition should be undertaken depends upon many factors, including an organization's industry, regulations and compliance rules, the size and age of the organization, workplace culture, the maturity of an organization's products and services, team sizes and the number of teams, and current project management practices.

- How a transition is undertaken should be determined by the goals of the organization — what is hoped to be achieved.
- A successful agile transition requires the backing of C-level executives; a bottom-up approach is futile.
- The first step of any transition to scrum is the creation of the first scrum team.
- Transitioning to agile practices requires training and educating of most members of an organization — not just future scrum team members — in agile practices and principles. Training and education are essential for a successful transition. **Read more**: Prototyping with Absolute Beginners
- There is a huge difference between 'doing Agile' and 'being agile'. Transitioning successfully means becoming — and being — agile.
- In an organization transitioning to scrum, future scrum masters should be agents of change rather than drill sergeants — this is by design, given their lack of proper authority.
- Creating a 'happy agile island' for the product and engineering department is a valid objective. However, in comparison to breaking up functional silos and creating a learning organization, it is likely to deliver a lesser return on investment.

Conclusion

To be a successful scrum master, you will need to have a holistic understanding of the product creation process. You will need to be a part-time agile coach, too, dealing with organizational impediments and constraints, while training other members of the organization and looking for prospective change agents that may join your course. You need to sell 'agile.' You need to win hearts & minds within the organization to overcome its inherent resistance to change. Hence, organizing scrum ceremonies is just a small part of a scrum master's job.

The Scrum Product Owner in 56 Theses

Introduction

The following 56 theses describe the role of the scrum product owner from a holistic product creation perspective.

The 56 theses cover the concept of the product owner role, product discovery, how to deal with external and internal stakeholders, product roadmap planning, as well as the product backlog refinement. The theses also address the product owner's part in scrum ceremonies such as sprint planning, sprint review, and the sprint retrospective.

The Role of a Product Owner

This first set of theses addresses the product owner's role in the scrum process:

- A product owner embraces, shares, and communicates the product vision. They represent both the customer and internal stakeholders.
- With respect to process, a product owner is the gatekeeper of the product backlog, and thus "owns" the product on behalf of the organization as well as customers.
- A product owner is responsible for maximizing the value that the product provides to both customers and the organization.
- To maximize the value of a product, the product owner must be empowered to make all product-related decisions on behalf of the organization.
- If a Product "Owner" is not empowered to "own" the product, they are not a product owner per se, and the organization is not practicing scrum.
- A product owner is the sole representative of the stakeholders, internal and external, insofar as the development team is concerned.
- A product owner owns the "why", and influences the "who" and "what", but should never be concerned with the "how". Progress in product development is always a collaborative, team effort.
- A product owner must work closely with the core scrum team, and particularly the scrum Master or agile coach — their natural ally.
- A product owner should actively participate in scrum ceremonies, especially the product backlog refinement, sprint planning, and sprint acceptance ceremonies.
- A product owner needs to be co-located with the scrum team to avoid delays, communication errors, and other problems caused by distance.
- Contrary to popular belief, a product owner is neither a user story author nor a requirements engineer, but rather a communicator and facilitator between the stakeholders and the scrum team.

Product Discovery and External Stakeholders

Theses theses concern what's required of the product owner with respect to product discovery and product management:

- A product owner must have a holistic understanding of problems and opportunities: in the market, inherent to the product itself, with the organization and its strategy, and of concern to the various stakeholders.
- The job of a product owner is to create value for both customers and the organization while mitigating risk.
- A product owner creates value by embracing a continuous product discovery process built around learning and experimentation.
- During the product discovery process, a product owner validates hypotheses by continuously running experiments.
- Frameworks and methodologies suitable for the product discovery process include A/B testing, Business Model Canvas, Continuous User Testing, Design Sprints, Design Thinking, Lean Startup, Lean UX, and Rapid Prototyping — to name just a few.
- A product owner must be capable of thinking in terms of systems to deal with complexity.
- The earlier a product owner is involved in a product's lifecycle, the more valuable that product owner will be to the organization.

Internal Stakeholder Management

The following theses concern specific aspects of the relationships between product owners and their internal stakeholders:

- A product owner needs to gain the trust and mandate of all internal stakeholders.
- A product owner must be able to explain to any internal stakeholder at any time how the stakeholder's requirements are accommodated by the product vision.
- Regular feedback from internal stakeholders is crucial to a product owner's work being successful — specifically their success with creating the hypotheses funnel that they use to run experiments.
- Close cooperation between a product owner and their organization's customer care and sales teams is particularly beneficial for product success.
- A product owner must be empowered by the organization to say "No" to a stakeholder's requests — no matter how powerful the stakeholder is.
- A product owner's communication with internal stakeholders needs to be transparent and regular to encourage these stakeholders to be engaged with the scrum team.
- The scrum master is a good ally for a product owner in the pursuit of internal stakeholder engagement.

- Good opportunities for a product owner to engage internal stakeholders are scrum ceremonies, like the sprint acceptance (demo); workshops, such as user story mappings; or training sessions, whereby stakeholders are taught how to better communicate with the scrum team.
- Internal stakeholders make excellent members of customer development or user research teams, and a product owner should seek to secure their participation in these activities.

Set 4: Product Roadmap Planning

The theses in this set concern one of the most contentious topics in the profession: "How do we build agile product roadmaps that actually work?"

- A product owner's role requires that they also act as a product manager, which means that they must define the product vision, perform strategy and market research, develop business models, manage the product lifecycle, and facilitate product portfolio and roadmap planning. (For more detail, refer to Roman Pichler's *The Scrum Product Owner Role on One Page*.) http://www.romanpichler.com/blog/one-page-product-owner/
- An agile product roadmap is a high-level plan that describes how the product vision is likely to be accomplished. It should facilitate experimentation and learning.
- An agile product roadmap is based on objectives, and is usually theme- or goal-oriented.
- An agile product roadmap is not a prioritized list of features with fixed shipping dates for months to come. (For more detail, refer to my *7 Best Practices on How to Build a Product Roadmap*.) https://age-of-product.com/7-best-practices-on-how-to-build-a-product-roadmap/
- Usually product roadmap planning in larger organizations with several products requires that each product owner align their efforts with other product owners to synchronize product development.
- A product roadmap addresses strategic aspects of product planning. A product backlog addresses technical development issues.
- Roadmap planning is, like product backlog refinement, a continuous effort — just at an increased cadence.
- Product owners communicate "big product pictures" by using techniques like user story mapping. (For more detail, refer to Jeff Patton's *User Story Mapping*.) http://jpattonassociates.com/user-story-mapping/
- A product owner needs to be familiar with the five levels of agile planning: defining the product vision, defining the product roadmap, release planning, sprint planning, and accounting for the outcome of daily scrums.
- Relying on a committee of stakeholders for product discovery and portfolio management is the most common reason that agile product delivery initiatives fail.

The Product Backlog and User Story Creation

The theses in this section concern a product owner's home turf: the product backlog, and user story creation.

The Product Backlog

- A product owner is much more than the "project manager of the product backlog", and must do more than churning out user stories on behalf of stakeholders (a.k.a. "ticket monkey syndrome").
- Product backlog refinement is a continuous process that needs to be in sync with the product discovery process.
- Typically, a scrum team will collaboratively refine product backlog items for the upcoming two or three sprints.

User Story Creation

- Creating user stories does not equal breaking down requirements documents received from stakeholders into smaller chunks.
- Writing user stories is a collaborative effort involving the entire scrum team. The process should create a shared understanding of what will be built, and for what reasons.
- Because it's a collaborative effort, a user story is a subject of discussion for the scrum team. This might take up to 10% of the team's availability during a sprint.
- A product owner will need to come to an agreement with their team as to what standards user stories need to achieve before being considered suitable for the sprint backlog (i.e. defining and achieving the "Definition of Ready").
- Planning poker — the process of estimating user stories — is, most importantly, knowledge transfer. It supports the creation of a shared understanding within the scrum team of what needs to be built.
- User story estimation is a critical part of the risk mitigation strategy for the scrum team.
- With respect to the "Definition of Ready": a candidate for the role of product owner should have heard of Bill Wake's INVEST acronym (from the article *INVEST in Good Stories and SMART Tasks*). http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Sprint Planning, Reviews, and Retrospectives

The final set of theses concern product delivery: the sprint itself.

- A product owner defines the scope of upcoming sprints by identifying and prioritizing the most valuable user stories in the product backlog.

- A product owner should participate in all scrum ceremonies related to sprints.
- The product owner is the person responsible for defining a sprint's goal.
- A product owner understands that, in addition to user stories, technical tasks, bugs, and research need to be addressed in every sprint. (For more detail, refer to Barry Overeem's *The Backlog Prioritisation Quadrant*.) http://www.barryovereem.com/the-backlog-prioritisation-backlog/
- A product owner should be available on short notice to clarify any questions that the scrum team may have during a sprint.
- A product owner is responsible for accepting user stories into each sprint, and for deferring user stories that require additional work to meet the 'Definition of Ready' standard. This does not apply to user stories that are related to technical or refactoring tasks. The decision on those is the prerogative of the scrum team.
- A product owner is responsible for deciding whether to release a product increment at the end of each sprint.
- A product owner should host the sprint review, which is an event meant to provide the scrum team an opportunity to demo the outcome of each sprint to the product's stakeholders.
- A product owner must embrace the sprint review as a vital *inspect and adapt* feedback loop — for both the development team and the product's external and internal stakeholders.

Conclusion

I believe that the scrum product owner role is the most demanding of all three scrum roles. It covers a lot of ground, from product discovery, politics, and stakeholder management to systems thinking, and maximizing of the return on investment for his or her team.

It is also the scrum role that the dark side can misuse simpler than the two other roles.

Peer Recruiting: How to Hire a Scrum Master in Agile Times

Peer Recruiting is the new hiring: In the near future, all creative, technology-based organizations will need to abandon the command & control structures that served the industrial world of the 20th century so well. Instead, they will reorganize themselves around autonomous teams to deal with the complexity and pace of innovation of the 21st century.

In such an agile world, recruiting will become a team decision, and the role of the human resources department will change into a supportive one. Recruiters will need to become servant leaders or facilitators, guiding the peer recruiting process.

The following guide to peer recruiting is based on my own experience in participating in the recruiting of such team members with Scrum-related roles over the last five years. This first article will cover the Scrum Master role.

Peer Recruiting and the New Role of HR

In the near future, all creative, technology-based organizations will need to abandon the command & control structures that served the industrial world of the 20th century so well. Instead, they will become self-organized structures, built around autonomous teams. (Think General Stanley McChrystal: "Team of Teams: New Rules of Engagement for a Complex World").

Note: A good intro in the idea of the "Team of Teams" from an organizational point of view is Culture First's podcast: Team of Thrones.

In such an agile world, recruiting will become a team decision, and the role of the human resources department will change into a supportive one. Recruiters will need to become servant leaders or facilitators in this peer recruiting process.

The Supportive Role of the HR

Peer recruiting does not imply, that HR will be rendered obsolete. On the contrary, HR will in the future continue being a major contributor to the success of the whole organization. However, HR's role will change from choosing someone from the candidate pool and present that individual to the team as the new teammate. Instead, HR will support the team picking the "right" candidate and ensure that the legal and administrative side is being taken of.

Typical tasks of the peer recruiting process, that HR will provide to a team therefore comprise of:

1. Creating the remuneration package for the position in question (in compliance with the organization's principles)
2. Handling contractual and administrative issues (social security, visas, work permits etc.
3. Supporting the team creating a job advertisement (if required)
4. Placing job advertisement and run corresponding campaigns
5. Doing background checks and pre-screenings of applicants
6. Organize interviews and trial days (from travel arrangements, and meet & greet to introducing the organization)
7. Collecting the team's feedback after interviews or trial days
8. Handling the signing of the contract
9. Finally, kicking-off the onboarding process for the new teammate.

These steps hold a significant opportunity for HR to become a change agent for the organization, contributing to its agile transition by ensuring that new hires will have the required agile mindset.

Why Bother with the Inclusion of the Team at all?

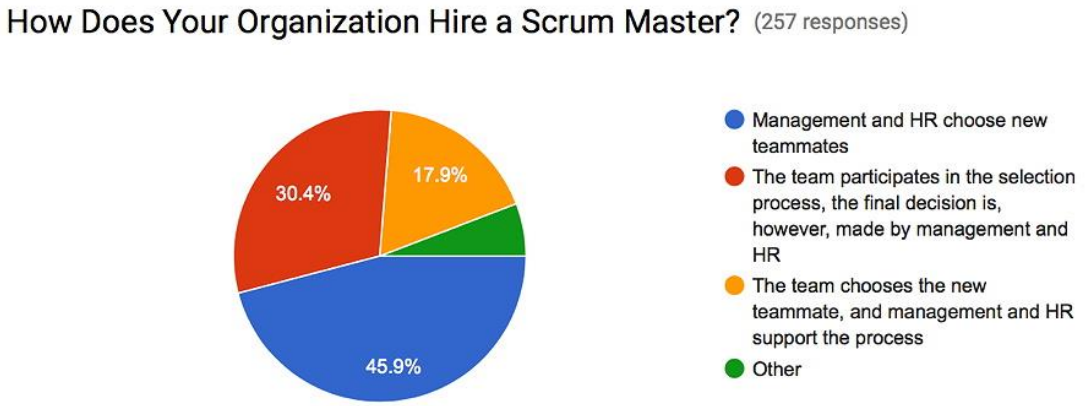
You may wonder why a change of process will be required in the first place?

There are plenty of reasons, my top three being:

- It's consequent. On the one side, the team is empowered to make decisions that directly impact the return on (product) investment. On the other, they are being patronized by deciding on new teammates for them?
- It also means the team has skin in the game. And they will be motivated to go the extra mile to make the new connection work. Now, it is their responsibility, too.
- Last but not least, not involving the team immediately signals to all candidates, that your organization isn't agile, but merely "doing Agile"—a weak value proposition in the war for talent with an agile mindset.

The Current Status: The "How to Hire a Scrum Master" Survey

For about three weeks prior to this article, I have been running a poll "How Does Your Organization Hire a Scrum Master?", and so far almost 240 participants have contributed: It turns out that about 18% of organizations, that are supposedly agile, delegate the final hiring decision to the team itself.



Note: The survey is still open for voting, please join and enhance the sample size.

Now, let's have a detailed look at the proposed process, which has been proven to be motivating as well as successful several times so far in different organizations.

The Eight Steps Peer Recruiting Process of Hiring a Scrum Master

1. What kind of Scrum Master Are You Looking for?

This is the question you will need to answer in the first place:

What is the purpose of building autonomous teams in your organization? Does the organization want to become (or stay) agile? Or is the organization just "doing agile"?

Given that the original Scrum motto of "inspect and adapt" meanwhile turned into a quasi religiously followed set of principles—as taught in any official Scrum certification training—, this question is less trivial than it sounds.

The majority of applicants for a Scrum Master role, I have met so far, were following exactly this dogmatic set of principles. And I am skeptical that those applicants would make a good addition to any agile organization. It is all about mindset, not skills.

(**Read also**: Scrum Master Anti Patterns: Beware of Becoming a Scrum Mom or Scrum Pop.)

So, in the "team of teams" universe, you should always hire for mindset. While you can easily teach skills, training someone unsuitable in the right mindset will be futile most of

the time. Which is also the reason, that in this agile, and complex world formalized, certified experience rarely matters.

Hence, the following description is targeting organizations that want to become agile. (See also “Scrum Master Certifications—A Necessity?” paragraph below.)

2. Create a Job Advertisement

In an ideal world, there wouldn’t be a necessity to run a job ad. Someone in the product development organization would personally know a suitable individual and introduce her to the team. (And the organization.)

Unfortunately, truly agile people—those that are not merely sticking to the letters of their Scrum certification manual—are in short supply. So there probably will be the need to create a job ad for the website, as well as other channels.

I strongly recommend to kick-off the collaboration between the team and HR at this point. Most ads that HR departments produce for agile jobs are simply awful. Their usual “I don’t know what this is all about, but I have to come up with an ad by 12 pm, so I copied the text from a competitor” approach scares away suitable candidates, because they sense the lack of competence.

Instead, I suggest to sit together with the whole team, share a coffee, and get copy of the ad right, by being authentic, and human, and reflecting the culture of the organization.

3. Run the Job Advertisement

That is the HR department’s job. However, the team may have good suggestions outside the typical LinkedIn approach. Why not try Reddit, for example? Or sponsoring some meetups of the local agile community?

4. Pre-Screening Applicants

It would be helpful for the team, if HR could pre-screen applicants. This could be the standard background check. Or a first analysis if a candidate is suitable for formal reasons or in compliance with internal programs of the hiring organization, for example, diversity initiatives.

Unsuitable candidates should then be at least flagged, and probably be removed from the pool. (Although the team should be aware of that for transparency reasons.)

5. Discuss Suitable Applicants with the Team

The team members should then be provided with access to all suitable candidates, preferably in the form of anonymized CVs: No photos, no age, no gender, no ethic group, no religious believes—any information on candidates that might trigger a bias of whatever kind should be excluded.

Also, you will to normalize information on a candidate's public profile, e.g. blog or Twitter or other accounts. A summary such as "A is actively contributing to the Scrum community by running a meetup as well as creating a newsletter with 800 subscribers" will suffice for the selection process.

If this approach requires a scissor, glue, and a Xerox machine, so be it. Please keep in mind that these biases are triggered on autopilot, and that there is not willpower known to mankind that could prevent biases from interfering with the selection process.

Then have a joined meeting—HR & all team members—and discuss whom to invite for the in person interviews. (A simple dot-voting will suffice in the end.)

6. Running the Interviews

The "Hiring: 38 Scrum Master Interview Questions to Avoid Agile Imposters" PDF provides a large set of questions (and possible answers), spanning five categories.

Those are the starting point for the interviews. The purpose of the interviews is to identify those that will be invited for a trial day with the team.

Instead of one or two team members having a long interview with a candidate each, I recommend to run interviews of 30 minutes each with as many team members as possible.

The trick is that you split the questionnaire evenly among all of the interviewers and later aggregate the answers. Thus, you will obtain more constructive feedback from all the interviewers.

Tip: Create interview teams of two teammates each: One is asking questions, while the other is taking notes. After half of the interview has passed, they switch roles. The reason for that is that most people are not good at leading the interview and at the same time take meaningful notes. Two people, however, will have a much better chance to recognize signals on the candidate's side, for example, particular answers or body language.

Note: It is important that exactly the same procedure applies to all candidates otherwise the results are less comparable.

It is a good practice to run these debriefings to aggregate the answers right after an interview round with a candidate. Target for objectivity and have HR handle this task. They are the professionals.

The most important question to answer, however, is the "Would you like work with the candidate?" question. And that one should be asked the next morning. Sleeping on it will sober the interviewers, and thus provide a path to a better decision.

Tip: Go with your first thought, and walk away from any candidate who will have lost the "yes" overnight. Don't rationalize your decision, as people can be taught news skills, but they won't change their personality. The trial day is an expensive exercise, and should not be wasted.

Candidates that are not considered for a trial day should receive an answer, detailing the reasons for the decision. I know that legal departments tend to freak out over this. They usually fear legal action, for example, on grounds of discrimination legislation. However, respect and transparency are vital values of the agile community, and should be honored accordingly in my eyes.

Finally, invite the candidates that the team would be interested in working with for a trial day. Let the team make a suggestion for a date, as they need to align a trial day with their own sprint rhythm.

7. Have a Trial Day

Given my holistic view on being agile, a trial day for a Scrum Master should not merely focus on basic Scrum mechanics. If you do that, you might risk ending up choosing someone who is comfortable with "doing Agile by the book". (Whatever book that is…)
Hence, the purpose of the trial day is in my eyes to get an empirical understanding how the future Scrum Master can support the whole organization in becoming (more) agile. The three main areas, I focus trial days on, are as follows:

The Team

This is the simple part. Good exercises for hands-on learnings are:

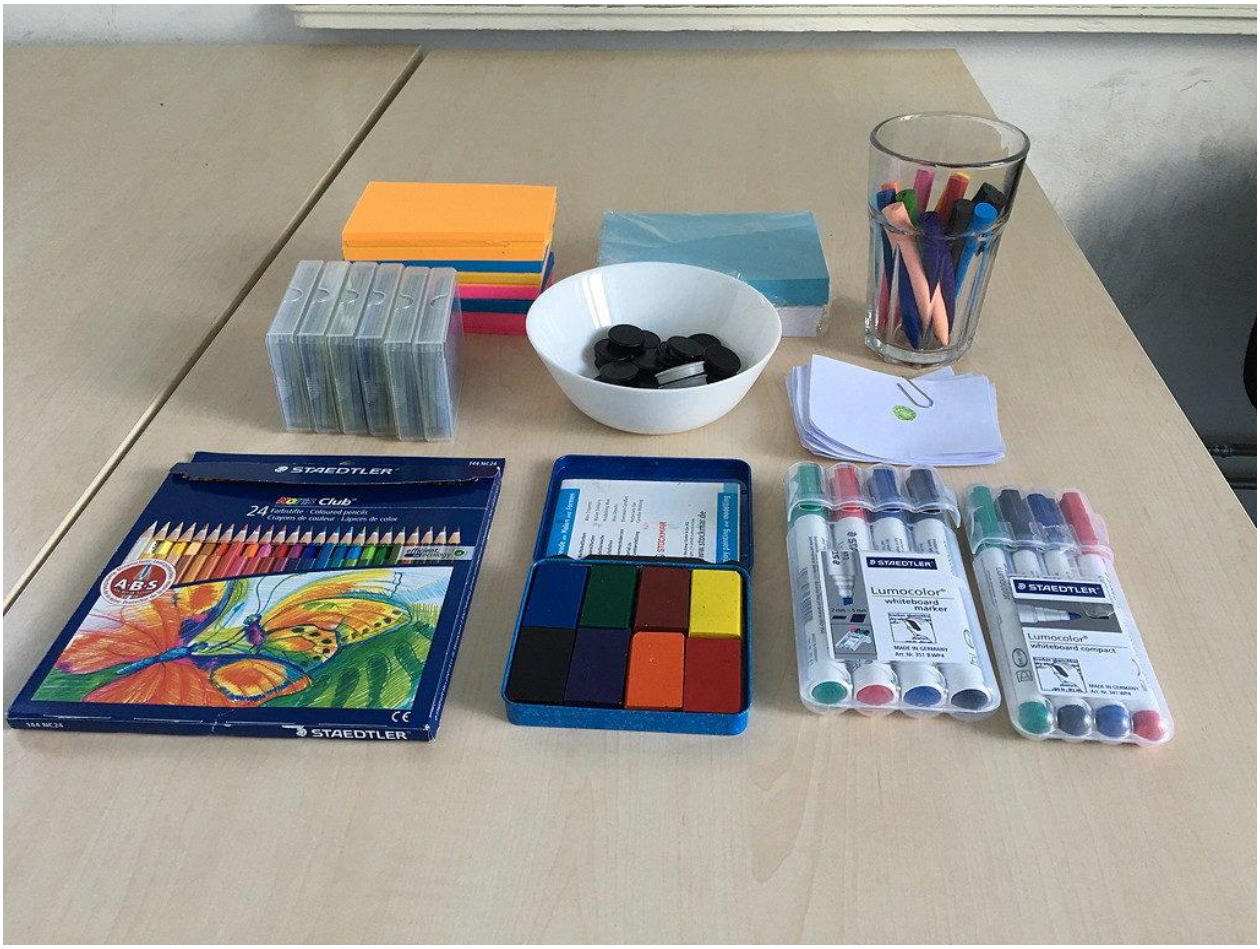
A. UNDERSTANDING THE CURRENT STATUS OF THE TEAM

Have an introductory session with the complete team, a kind of "ask me anything" session for the candidate. A simple questionnaire will do the job, for example: 20 Questions a New Scrum Master Should Ask Her Team to Get up to Speed..

B. RUNNING A RETROSPECTIVE

Ask the candidate to run a retrospective with the team in question. 30 minutes to prepare for the exercise should be more than generous. Actually, I would expect a seasoned Scrum

Master to have prepared retrospectives at hands. (Retromat offers a wealth of exercises to choose from, see also: How to Curate Retrospectives with Retromat.



Scrum Master survival kit…

By "retrospective" I am not referring to the basic "good, bad, and 2 actions items" 30 min version. I would expect something a bit more sophisticated along the lines of Esther Darby's and Diana Larsen's book: "Agile Retrospectives: Making Good Teams Great".

C. CREATING A DASHBOARD WITH AGILE METRICS

Visualizations are key to stakeholder communication, and I believe the Scrum Master should take care of collecting data, aggregating information, and finally providing the gained knowledge in a way that helps the organization grow.

In this exercise, the candidate would be asked to create an initial version of such a dashboard and start collecting the first data. Typical metrics, that are easily available by questionnaires or polls are:

1. Team happiness,
2. Perceived value delivered to customers during the last sprint,
3. Perceived current level of technical debt,
4. The agile health level in the organization:
    a. The Scrum checklist of Henrik Kniberg works fine for this purpose
    b. Alternatively, the 'State of Agile' Checklist for Your Organization
5. Last, but not least, Team velocity as well as a burn-down chart are both particularly well suited to better understand the candidate's mindset of being agile.

Note: Acquiring stakeholder feedback on level of appreciation of the production delivery organization will most likely not be possible on a trial day.

The Product Organization

Good exercises for the product organization are:

1. UNDERSTANDING THE CURRENT STATUS OF THE TEAM FROM THE PRODUCT OWNER'S PERSPECTIVE

Have an interview with the Product owner on the current situation from her perspective. Again, a simple questionnaire will do job, for example: 20 Questions from New Scrum Master to Product Owner.

A good candidate will be prepared for that interview, and provide her ideas how to she can contribute to improve the agile product discovery and delivery process.

2. PARTICIPATING IN A BACKLOG REFINEMENT

The candidate should participate in a backlog grooming session, helping the team to improve the product backlog of the product owner—garbage in, garbage out, right? The candidate should demonstrate best grooming practices during the exercise, addressing for example:

1. How to deal with large product backlogs,
2. Bill Wake's INVEST principle to create user stories,
3. How to handle acceptance criteria (for example, use Gherkin…)
4. The whole estimation vs. estimates process: estimation poker, knowledge transfer, #noestimates, predictability as an agile key metric.

A good candidate can ask the right question during refinement session without having detailed knowledge about the product backlog itself. Handling the process and its principles are in the focus of this exercise.

3. STAKEHOLDERS AND THE ORGANIZATION BEYOND THE PRODUCT & ENGINEERING

This part of the trial day assesses the future Scrum Master's communication capabilities. "Selling" the product and engineering organization to stakeholders and the rest of the organization is a not just a valuable, but an essential trait to either further an agile transition or maintain its dynamic.

It will be particularly important in organizations with silos and legacy command & control structures outside of the product and engineering organization. Or in fast growing startups with a lack of organizational structure to begin with, particularly when those are sales- or marketing-driven.

The task for the candidate will be to design a basic communication strategy with stakeholders that is suited to support transparency, interaction, and collaboration. (Read more on this topic here: 10 Proven Stakeholder Communication Tactics during an Agile Transition.)

A worthwhile trial day usually requires a full working day, as well as the attention of the whole team. Which is a pretty significant investment. So, choose the candidates carefully.

Tip: Invite the candidate—as well as the whole team—for lunch. It will be pretty much impossible for the her to play a role for 60 minutes, when interacting socially with several other people at the same time. Having food together brings out the true colors…

Note: Menlo Innovations takes the trial process even a bit further: "So we bring people in and get them to speed date with our own staff. The question is always: would you like to work with this person? If the answer is yes, then we bring them in to work with us for a day, then a week and then a month. If the answer is still, "Yes, I would like to work with this person," then they are hired."

8. Gather Feedback from the Team the Day after the Trial Day

Collect the feedback from the team members the day after the trial day with a simple questionnaire:

- "How would you rate the candidate's competence level on a scale from:
  - 1 [Awesome!] to
  - 6 [Thanks, but no thanks.]"

- "Did the candidate do anything to impress you positively?" (Free text field.)
- "Did the candidate do anything to impress you negatively?" (Free text field.)
- "Would you consider working with the candidate as your new teammate?" Three options:
  - Yes
  - No
  - Don't know
- "Should we make the candidate an offer? Three options:
  - Yes
  - No
  - Don't know

If the feedback is not unanimous, it is HR's task to take over. Either by entering the contract negotiation, or provide the negative feedback from the team, and continue the search.

If the feedback is not unanimous, the team should discuss—under the moderation from HR—whether the differences are surmountable or not. In the latter case, the candidate should not be forced upon the team. The team always has a veto right.

Scrum Master Certifications—A Necessity?

What about Scrum Master certifications, Scrum Alliance's CSM, for example?

Let Jilles van Gurp answer this question:

"I congratulate Ken Schwaber on his well oiled business of scrum training large parts of the industry (me included) but I believe that he doesn't honestly believe a two day training is enough either."

And Dan North describes today's situation as follows:

"Modern Scrum is a certification-laden minefield of detailed practises and roles. To legitimately describe oneself as a Scrum Master or Product Owner involves an expensive two day certification class taught by someone who in turn took an eye-wateringly expensive Scrum Trainer class, from one of the competing factions of "Professional" or "Certified" (but ironically not both) schools of Scrum training".

As both Jilles and Dan mention, the agile-industrial complex feeds its followers well. Nevertheless, it's fallacy to believe that two or three days of Scrum training will be even remotely successful in teaching the participants about Scrum.

Scrum is a framework to begin with, easy to understand, but hard to master. Scrum needs to be adapted to each organization depending on its culture, size, or the kind and maturity of its products, just to name a few aspects of this process.

So, a training—leading to a CSM or equivalent certificate—can only cover the smallest common (Scrum) denominator of all organizations: Artefacts, meetings and procedures. All issues that make a transition to agile complex and hard to master need to be experienced first hand. You cannot compensate a lack of experience by applying a dogmatic process to the letters of a book. That results in cargo cult Scrum.

Therefore, a Scrum Master certificate is more of a personal branding or advertising than a sign of expertise. However, I don't blame smart people for rising to the occasion and satisfying the corporate demand for agile management methodologies by creating a certification standard. (Or enhancing their LinkedIn profiles with the right search-terms, as I did myself.)

Conclusion

If your organization shall become agile, switching the hiring process to peer recruiting will be a necessity. It won't make HR obsolete but its role will change to facilitating others choosing the right candidates. HR will thus become a change agent, contributing to the agile transition of the organization.

Trying to stick with the traditional command & control process on the other side would signal everyone with an agile mindset that your organization isn't agile, but merely "doing Agile".

Hiring: 38 Scrum Master Interview Questions to Avoid Agile Imposters

Demand Creates Supply and the Job Market for Agile Practitioners is No Exception

Maybe, "Agile" in general is a management fad and not trend at the moment. But what we can say for sure is that Scrum has become very popular for software development purposes. A seasoned Scrum Master is nowadays in high demand. And that demand causes the market-entry of new professionals from other project management branches, probably believing that reading one or two Scrum books will be sufficient.

If you are looking to fill a position for a Scrum Master (or agile coach) in your organization, you may find the following 38 interview questions useful to identify the right candidate. They are derived from my ten years of practical experience with XP as well as Scrum, serving both as Product owner and Scrum Master as well as interviewing dozens of Scrum Master candidates on behalf of my clients.

Download the PDF including answers and background information here.

38 Scrum Master Interview Questions

Scrum is not a methodology, but a framework. There are no rules that apply to each and every scenario, just best practices that have worked in other organizations before. Hence, you have to figure out on your own what is working for your organization–which is a process, not a destination.

So, the role of the Scrum Master or agile coach in my understanding is primarily about leadership and coaching, but not about management. And most definitely, the Scrum Master role is not about "process enforcement". Which is also the reason, that the repository contains to a large part questions that are addressing a candidate's soft skills.

The PDF contains additional background and contextual information, as well as guidance on the range of suitable answers as well. These answers should enable an interviewer to deep dive into a candidate's understanding of Scrum and the candidate's agile mindset.

However, please note that:

1. The answers reflect the personal experience of the authors and may not be valid for every organization: what works for organization A, may be failing in organization B
2. There are not suitable multiple choice questions to identify a candidate's agile mindset given the complexity of applying "agile" to any organization

3. The authors share a holistic view on agile methodologies: agile equals product discovery (what to build) plus product delivery (how to build it) centered around self-organizing teams.

The questions are group into five sets:

I. The Role of the Scrum Master

- The Agile Manifesto says "People over processes". Isn't the Scrum Master – a role to enforce "the process" – therefore a contradiction?
- What are good indicators that "Agile" is working in your organization, that your work is successful?
- Are there typical metrics that you would track? And if so, which metrics would you track and for what purpose?
- Your team's performance is constantly not meeting commitments and its velocity is very volatile. What are possible reasons for that? And how would you address those issues?
- Shall the Scrum team become involved in the product discovery process as well, and if so, how?
- The Product owner role is a bottleneck by design. How can you support the Product owner so that she can be the value maximizer?
- How do you ensure that the Scrum team has access to the stakeholders?
- How do you spread an agile mindset in the company across different departments and what is your strategy to coach these non-IT stakeholders?
- How do you introduce Scrum to senior executives?
- You already performed Scrum training to stakeholders. After an initial phase of trying to apply the concepts, when first obstacles/hurdles are encountered, you see that these colleagues build serious resistance in continuing with Scrum adoption. What are your strategy/experience to handle such situations?

II. Product Backlog Refinement and Task Estimation

- The Product owner of your team normally turns stakeholder requirement documents into tickets and asks to estimate them. Are you fine with that procedure?
- What kind of information would you require from the Product owner to provide the team with an update on the product and market situation?
- Who shall be writing user stories?
- How shall a good user story look? What is its structure?
- What should a "Definition of Ready" comprise of?
- Why aren't user stories usually simply estimated in man-hours?
- The Product owner of your Scrum team tends to add ideas of all kind to the backlog to continue working on them at a later stage. Over time, this has lead to over 200

tickets in various stages. What is your take on that: Can the Scrum team work on 200 tickets?

III. Sprint Planning

- How can you as a Scrum Master contribute to the sprint planning in a way that the team is really working on the most valuable user stories?
- On what metrics would you base the assessment of the value of a user story and what metrics would be not acceptable?
- How do you facilitate the user story picking progress in a way that the most valuable stories are chosen without overruling the team's prerogative to define the team's commitment?
- How much capacity would consider to be adequate for refactoring, fixing important bugs, exploring new technologies or ideas?
- How do you deal with a Product owner that assigns user stories or tasks to individual team members?
- How do you deal with cherry picking tasks by team members?
- A user story is lacking the final designs, but the design department promises to deliver on day #2 of the upcoming sprint. The product owner of your Scrum team is fine with that and pushed to have the user story in the sprint backlog. What is your take?
- A member of the Scrum team does not want to participate in the sprint planning meetings but considers them a waste of time. How do you deal with that attitude?

IV. Standups

- Would you recommend standups for all teams no matter their size or experience level?
- Do you expect experienced team members to wait until the next standup to ask for help with an impediment?
- How do you handle team members that "lead" standups, turning them into a reporting session for them?
- How do you handle team members that consider standups to be a waste of time and therefore are either late, uncooperative or do not attend at all?
- The standups of your Scrum team are not attended by any stakeholder. How do you change that?
- How do you approach standups with distributed teams?
- Can you draw a draft of an offline Kanban board for a Scrum team right now?

V. Retrospectives

- Who shall participate in the retrospective?

- Do you check the team health in a retrospective or isn't that necessary? If so, how would you do it?
- What retrospective formats have you been using in the past?
- How can you prevent boredom at retrospectives?
- A team is always picking reasonable action items, but is later not delivering on them. How do you handle this habit?
- How do you recommend to follow up on actions items?

How to Use the Scrum Master Interview Questions?

Scrum has always been a hands-on business, and to be successful in this a candidate needs to have a passion for getting her hands dirty. While the basic rules are trivial, getting a group of individuals with different backgrounds, levels of engagement and personal agendas to form and perform as a team, is a complex task. (As always you might say when humans and communication is involved.) And the larger the organization is, the more management level there are, the more likely failure is lurking around the corner.

The questions are not necessarily suited to turn an inexperienced interviewer into an agile expert. But in the hands of a seasoned practitioner they support figuring out, what candidate has actually been working the agile trenches in the past and who's more likely to be an imposter.

So, go for a pragmatic veteran who has experienced failure in other projects before and the scars to prove it. Last, but not least: Being a "Certified Scrum Master" – or having any other certification of a similar nature – does not guarantee success.

Download the PDF including answers and background information here.

42 Product Owner Interview Questions to Avoid Hiring Agile Imposters

42 Interview Questions that Will Benefit Your Organization

With more than 5,000 downloads of the first publication in our Hands-on Agile Fieldnotes series, 38 Scrum Master Interview Questions to Avoid Hiring Agile Imposters, and the continuing demand for it, we've confirmed that it's critical to hire people with an agile mindset for an agile transition to succeed.

This second publication in the Hands-on Agile Fieldnotes series is therefore focused on hiring for the other critical Scrum role: that of Product Owner.

Co-authored with Andreea Tomoiaga, *42 Product Owner Interview Questions to Avoid Hiring Agile Imposters* represents the most important learnings of our more than 20 years combined hands-on experience with Kanban, Scrum, XP, and several product discovery frameworks. We have worked as Product Owners, Scrum Masters, agile coaches, and developers in agile teams and organizations of all sizes and levels of maturity. We have each participated in interviewing dozens of Product Owner candidates on behalf of our clients or employers. The questions and answers herein are what we have learned.

Scrum's Product Owner Role Is Tricky to Grasp

Scrum is not a methodology, but a framework. There are no rules that apply to each and every scenario — just best practices that have worked in other organizations before. As somebody hiring for an agile team, you need to determine for yourself what works for your organization — which is a process, not a destination.

The role of the Product Owner itself makes the hiring process difficult to handle. The Product Owner is the least well defined role within the Scrum framework and — at the same time — the role with the most facets.

Product Owners are innovators at heart and thus value creators for both their customers and their organizations — if given the chance to work in an agile manner. Theirs is the most vulnerable Scrum role. Turn a Product Owner into a (ticket-system of your choice) monkey, or deprive them of the ability to say "No" (by making them the gatekeeper of the product backlog), and they quickly become the Achilles' heel of any agile organization.

The Product Owner role depends upon the size of the organization, the industry it operates in, and the lifecycle stage of it's products. But most importantly, overlap with the product manager role must be considered (spoiler: they aren't identical).

These 42 interview questions are neither suited nor intended to turn an inexperienced interviewer into an expert on agile software development. But in the hands of a seasoned practitioner these questions will provide ample support when needing to determine who of the candidates has actually worked successfully in the agile trenches. Remember: "agile" is a mindset, not a methodology. There is no checklist that will drive your recruiting success.

42 Product Owner Interview Questions to Avoid Hiring Agile Imposters provides contextual information including guidance on suitable answers and instruction as to how these interview questions are best used. The questions are grouped into six sets covering the most important work areas.

Set 1: The Role of a Product Owner

This first set addresses a candidate's conceptual understanding of the Product Owner's role in the Scrum process:

- What's the purpose of being agile in the first place?
- How would you characterize your role as a Product Owner? Are you a facilitator, a coach, a manager, a visionary, a tactician, a coordinator, or a driver?To what extent is the Product Owner a "product manager"?
- When was the last time you said "No" to a stakeholder? How did you approach the situation? What was your reason for saying "No"?
- Your product backlog is guarded by a 'product committee' who meet regularly to approve new features. Can you act as a credible Product Owner if you're not in control of the product backlog?
- What titles would you think suitable on your business card when you think of your role as a Product Owner?
- How do you cooperate with the Scrum Team?
- Would it bother you if your Scrum Master suggests a course of action concerning product development?

Set 2: Product Discovery and External Stakeholders

The questions in this set concern what's required of the product owner with respect to product discovery and product management:

- Do you think Scrum adequately addresses the product discovery process?
- How do you learn about new ideas and requirements?
- How do you include user research in the product discovery process?
- How much time do you allocate to user research and understanding your customers' needs?

- How would you design a process to handle product ideas from stakeholders — and the organization generally?
- At what stage do you involve the Scrum team in the product discovery process?
- How do you avoid misallocating resources to features or products that no one is really interested in?

Set 3: Internal Stakeholder Management

The questions in this set concern specific aspects of the relationships between product owners and their internal stakeholders:

- Your organization has recently decided to 'go agile' in product development. How do you educate your stakeholders about the implications?
- How do you organize a Scrum team's collaboration with stakeholders — and improve it over time?
- How do you communicate with uncooperative stakeholders?
- A new feature is overdue and has been drastically underestimated because of unexpected technical debt. Nevertheless, your most important stakeholder insists on 'finishing it' because so much effort has already been invested. How do you deal with this?
- How do you deal with pet projects?
- The sales department often sells new features to close deals without talking to you first. How do you deal with that?

Set 4: Product Roadmap Planning

The questions in this set concern one of the most contentious topics in the profession: "How do we build agile product roadmaps that actually work?"

- Product vision and strategy are kept confidential in your organization to prevent competitors from stealing the ideas. Will that impede your work as a product owner?
- Aren't product portfolio and roadmap planning anachronisms in an agile organization?
- What is your approach to creating product roadmaps?
- How often should product roadmap planning be done?
- How do you connect teams to the product vision — and show them how their contributions bring that vision to life?
- Who should participate in product roadmap planning?

Set 5: The Product Backlog and User Story Creation

The questions in this set concern a Product Owner's home turf: the product backlog, and user story creation:

- What's the idea behind product backlog refinement?
- How would you organize the process of refining the product backlog?
- How many user stories can you work on in parallel while ensuring their continued relevance to customers and the organization?
- At what stage do you include other team members in the refinement process?
- How do you handle bugs and technical debt when there are a lot of valuable new features competing for resources?
- What should a good user story look like? What is its structure?
- What are the most common pitfalls of product backlog refinement?
- When would you remove a product feature?

Set 6: Sprint Planning, Reviews, and Retrospectives

The questions in this final set concern the sprint itself—planning, delivery, and closure:

- How do you ensure the Scrum Team will be working on the most valuable user stories?
- Is it necessary for the Product Owner to set the goal for a sprint?
- You are pushing for an important user story to be selected for the next sprint. Unfortunately, the final designs are missing — but the designers promise to deliver no more than two days into the sprint. The Scrum Master, however, rejects the story because the 'Definition of Ready' has not been achieved. What can you do?
- Should the Product Owner attend the entire sprint planning ceremony?
- Your Scrum Team regularly estimate user stories at the upper end of the possible range. You believe they're playing safe, creating buffers for rainy days. How do you address this?
- When do you accept user stories?
- Does a Product Owner have veto over the release of user stories?
- During a sprint review, the development team demos new functionality you've never seen before. How do you react?

How to Use These 42 Interview Questions

Scrum has always been a pragmatic business, and to succeed in this business a candidate needs to have a passion for getting their hands dirty. Although the basic rules are trivial, getting a group of individuals with different backgrounds, levels of engagement, and personal agendas to continuously deliver value by creating a great product is a complex

task. And the larger the organization is — the more levels of management there are — the more likely failure, in one of its many forms, is lurking around the corner.

These interview questions are neither suited nor intended to turn an inexperienced interviewer into an expert on agile software development. But in the hands of a seasoned practitioner these questions will provide ample support when needing to determine who of the candidates has actually worked successfully in the agile trenches. They'll also help you determine who's most likely to be an imposter.

You want to avoid inviting imposters for a trial. Look for the pragmatic veteran who has experienced both failure and success with previous projects, and who carries the scars to prove it. Certifications, including the CSPO® (CSPO stands for "Certified Scrum Product Owner" and is a registered trademark of Scrum Alliance, Inc.) certification, are by no means an indication that you've got the "right candidate".

Download the PDF

The free *42 Product Owner Interview Questions to Avoid Hiring Agile Imposters* will be available from January 2017 on.

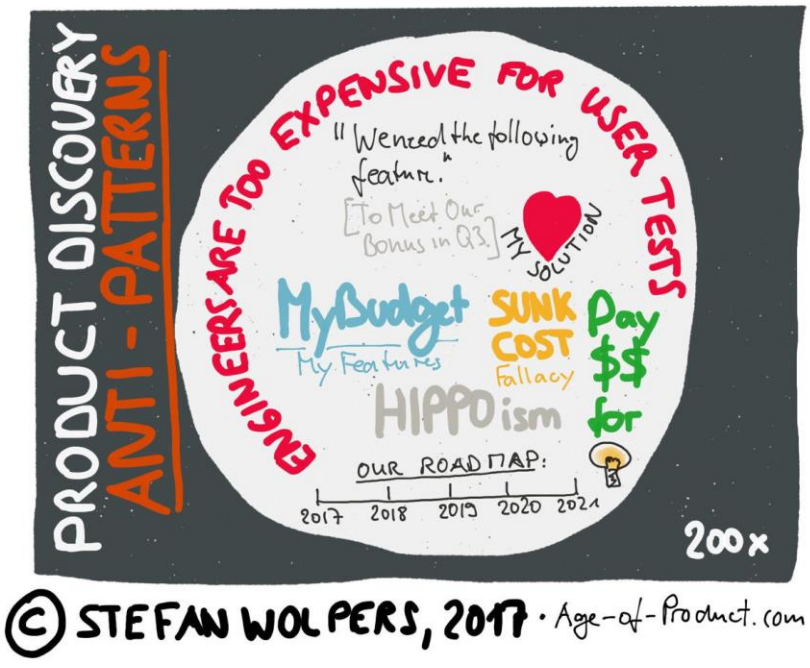
Product Discovery

Product Discovery Anti-Patterns Leading to Failure

Introduction

Scrum has proven to be an effective product delivery framework for digital products like applications or apps. However, scrum is equally suited to build the wrong product efficiently as its Achilles heel has always been the product discovery part. What product discovery part, you may think now. And this is precisely the point: The product owner miraculously identifies what is the best way to proceed as a team by gating and prioritizing the product backlog. How that is supposed to happen is nowhere described within the scrum guide.

It is this void on the product discovery side, or better: the attempts to fill it in a supposedly meaningful way that often results in scrum teams chasing the wrong objectives. Learn more about the numerous product discovery anti-patterns that can manifest themselves when you try to fill scrum's product discovery void.



Scrum's Achilles Heel: Product Discovery

In the attempt to fill scrum's product discovery void, product delivery organizations regularly turn to other agile frameworks like lean UX, jobs-to-be-done, lean startup, design thinking, design sprint—just name a few. The wave of agile transition projects particularly in large, established organizations over the course of the recent years has provided those frameworks with a tremendous tailwind. Some ideas have meanwhile gained buzzword status in the process causing the occasional collateral damage along the way. (I stopped correcting stakeholders of my current project—mainly from the business side—when they use "MVP" in an incorrect way, for example.)

Generally, it is safe to assume that agile product discovery anti-patterns result from a broader set of issues by comparison to the anti-patterns of a particular framework such as scrum. (**Download for free**: The Scrum Anti-Patterns Guide.)

The main contributing variables to various product discovery anti-patterns are:

- Existing organizational dysfunctions, for example, the organization is structured in functional silos,
- A substantial degree of ego issues among individual players—the what-is-in-for-me-syndrome—resulting in personal agendas being pursued,
- A complex, multi-layered reporting structure within organizations that filters as well as delays the flow of information, thus impeding communication and decision-making.

Additionally, particularly larger organizations struggle with the necessary transition at the core of the process of becoming an agile, learning organization: The move from allocating budgets to projects staffed with temporary teams group of FTEs to building and funding lasting product teams.

I do believe that a separation between product discovery on the one side (product management), and product delivery (engineering) on the other side is nowadays no longer a viable approach. To prevail in today's game of an accelerated innovation-based competition—software is eating the world—, every organization needs to acquire a holistic understanding of an agile product creation process:

A vision leads to a strategy which (probably) results in a portfolio of products (and services). Each of those products has a product roadmap that needs to be reflected in the product's backlog, which ultimately provides the sprint backlog. This core agile product creation process is agnostic to the product delivery framework, be it scrum, XP, or any other framework.

The organization needs to inspect and adapt every layer of this hierarchy continuously within appropriate time intervals. Apparently, there is a difference in the inspect & adapt cadence when strategy and sprint backlog are compared to each other. (**Example**: The Secret Tesla Motors Master Plan (just between you and me) from August 2nd, 2006.) Keeping this holistic product creation process in mind, you can often observe some of the following product discovery anti-patterns in practice.

Product Discovery Anti-Patterns: The Fallacy of Knowing Better

- **'We know what to build'**: The is no user research, nor any other interactions of the product delivery organization with customers. (There are several reasons causing this phenomenon ranging from a founder or entrepreneur who pursues his or her product vision without engaging in customer discovery activities. Or the product delivery organization is solely briefed indirectly by key account managers. Probably, the sales department deems a direct contact of the product people with customers too risky and hence prevents it from happening. What these patterns share is either a bias that is hurting the learning effort or a personal agenda. While the former can be overcome by education, the latter is more difficult to come by as the culprits typically reject the idea that they are guided by selfish motives. For becoming an effective product delivery organization it is essential that the team directly communicates with customers at a regular base.)

- **Loving the solution**: This is a variation of the 'we know what to build' syndrome. (There are no ugly children at least not from their parents' perspective. This bias seems to apply to products and services, too, think of Ikea: people value what they created over other products. Which is also the reason that you avoid asking strangers for their opinion: They might not be sharing your conviction. The solution to this product discovery anti-pattern is simple: Fall in love with the problem, not your solution.)

- **No shared understanding**: The product delivery organization does not make any attempts to create a shared understanding among the team and the stakeholders on what to build for which reason. (The best way to secure an engaged and effective communication with internal and external stakeholders is to include them in the product creation process. People care for what they help create. The best way to do so in my experience is Jeff Patton's user story mapping. It is not just a great way to turn 'requirements' into feasible user stories at a tactical level. User story mapping is also very well suited to come to an understanding on product roadmap and release planning.)

- **Persona-driven self-fulfilling prophecy**: You create the personas first based on what you believe to be the customers or users of our next [add an adjective with a positive connotation here] product or service will be. (This is a potentially dangerous mental model as you may fall victim to a self-fulfilling prophecy. It is not

unlikely that you unconsciously create user tests in a way that will verify your personas in the process. Instead, you should flip the process: first, you observe potential customers in their natural habitat to figure out a problem worth solving. Then you create a prototype addressing that previously identified problem. After the prototype, you create preliminary personas from the insight you gained during the related research. "Preliminary" needs to be stressed, too, as the persona design needs to be constantly calibrated with what you learn while continuously running user tests. There are no static personas. **Read More**: "Creating Personas.")

- **Annual roadmaps**: Product roadmaps are planned once a year for twelve months in advance. (Contrary to popular belief in traditional command & control organizations product roadmaps are subject to continuous planning. The agile product delivery process is centered around the idea to work solely on those ideas that provide the highest value to customers and the organization at any given time. To meet that standard the product roadmap needs to be adapted to the learnings from running product experiments regularly in an appropriate cadence.)

- **The incrementalism trap**: You get so focused on improving incrementally that you miss the available innovative leap ahead. Remedy: Apply Elon Musk's first principles thinking model and break down a situation into its elementary pieces to put it back together again in a way that serves your purpose better.

Product Discovery Anti-Patterns: Silos at Work

- **Design as a silo**: There is a UX/UI design team/department that is organized as functional silo outside the development team. (UX/UI competence is essential for a lot of cross-functional teams. If those competencies are only available by sourcing them from a different entity the organization creates an artificial hand-over. The interruption will impede both speed and quality of the team's work as crucial information and learnings will only be relayed but not experienced first hand.)

- **User tests/user research as a silo**: User tests need to be handed over to a functional silo. The researchers will be reporting back once the results are in. (User tests are an integral and thus continuous part of the product discovery process. Hence, user research shall be run by the team itself by embedding someone with the necessary competencies into the team. Read More: How to Run Lean User Tests.)

- **Don't waste engineers on user tests**: The engineers do not participate in user tests because they are deemed too valuable/expensive. (They are supposed to code as it is easier to hire an additional UX designer by comparison. This is the wrong mental model: the earlier engineers participate in identifying a problem worth solving they better the ROI on their engagement will become as they will bring the technical expertise on feasibility to the table. Building the wrong thing due to isolating the engineers is for several reasons incredible expensive. It is a huge waste of resources,

and there will be significant costs of delay—the team could have been built something more valuable in the meantime. Lastly, it is incredibly frustrating to see your effort go to waste, no matter what you were paid to make it. (**Read More**: (1) Daniel Pink: Drive. (2) In Chapter 2—"The Meaning of Labor"—of his book "The Upside of Irrationality," Dan Ariely describes an experiment dubbed the 'Building of Bionicles' (Paperback edition 2011, pages 66 to 77) where Harvard University students were paid to assemble Lego figures. The experiment aimed at a better understanding of people's reactions to small reductions in the meaning of their work. His conclusion: "The translation of joy into a willingness to work seems to depend to a large degree on how much meaning we can attribute to our own labor." (See above, page 74.))

- **No engineers or designers working in customer support**: Engineers or designers do not work with customer support from time to time. (Have you heard of "support-driven development," that everyone in a company should spend five percent of his or her time in customer support? This is not a new idea, and a lot of successful companies practice it to simplify the flow of information from a client's problem to its solution. It proves to be an effective way to discover pain points and create products that clients love. Kayak.com co-founder Paul English: "Customers are a big source of my e-mails. Anytime anyone contacts us with a question, whether it's by e-mail or telephone, they get a personal reply. The engineers and I handle customer support. When I tell people that, they look at me like I'm smoking crack. They say, "Why would you pay an engineer $150,000 to answer phones when you could pay someone in Arizona $8 an hour?" If you make the engineers answer e-mails and phone calls from the customers, the second or third time they get the same question, they'll actually stop what they're doing and fix the code. Then we don't have those questions anymore." **Read More**: Everyone on Support at Basecamp)

- **A sole source of truth**: The product delivery organization relies solely on either quantitative or qualitative feedback. (You need to take both sources into account to come to an informed decision. Data does not provide any information on 'why' things are happening. Data only addresses 'what' is happening. At the same time, focusing exclusively on qualitative feedback may make you vulnerable to bias and unfavorably mental models. The road to self-fulfilling prophecies is paved with good intentions.)

Product Discovery Anti-Patterns: Cargo-Cult Discovery

- **Would you use this feature?** Your product discovery process equals asking prospective customers what problems they have. ("It's really hard to design products by focus groups. A lot of times, people don't know what they want until you show it to them." The famous Steve Jobs quote that everyone in product design and management has heard of before. Fact is, that it is useless to ask prospective customers in advance what features they would appreciate in a new product.

Whether you send questionnaires or you ask them directly in an interview, they will usually fail to imagine both the situation as well as the new product in a way that will provide you with actionable insight. The problem is called "psychological interference" due to the high level of required abstract thinking. **Read More**: Four Lessons Learned from Making Customer Value Your Priority.)

- **Selling non-existing features**: What features do you need us to provide to close the deal? Sales managers chase sales objectives by asking prospects for a feature wish-list and provide those to the product delivery organization as requirements. (The problem with customers is that they usually lack the depth of knowledge required to provide useful answers to this question. Most of the time, they also lack the level of abstract thinking necessary to come up with a viable, usable, and feasible solution. As the saying goes: if the only tool you are familiar with is a hammer every problem will look like a nail. Pursuing the sales process in such a way will lead the product into a feature comparison race to the bottom, probably inspired by bonuses and personal agendas. This is the reason why product people like to observe customers in their typical environment using a product to avoid misallocating resources on agenda-driven features. At a systems level, reconsidering individual monetary incentives for salespeople is helpful, too. In a learning organization, teams win not individuals.)

- **What features do you need this year?** To be fair, some product managers behave in a similar way like sales when they ask internal customers what features are required. (A call to send in your requirements typically ends up with an epic, Christmas-like wish-list of "requirements." If the resources of the development team are basically free of charge there is an incentive to grab as much of those as possible by extending the list whether those requests are valuable to the organization or not. You might also observe that stakeholder define requirements that they do not expect to be fulfilled just to gain leverage in future "negotiations" with the product delivery organization. This submission process is normally motivated on the stakeholder side by optimization efforts leading to a local maximum. (Note: Internal stakeholders acting this way behave perfectly rational under such conditions.) Instead, the product delivery organization needs to maximize the outcome of the whole organization. This maximization effort requires entering a competition of ideas across all internal stakeholders to channel resources to the most valuable ideas.)

Product Discovery Anti-Patterns: Ideas, Hypotheses, and Feedback

- **#NoTransparency**: There is no transparency how the product delivery organization chooses ideas for the hypotheses shortlist, and how those ideas make it from that shortlist to the experiment backlog. (Establishing the principle that internal stakeholders have to enter a competition of ideas to avoid creating local maxima at the expense of the organization is in itself challenging. The prerequisite for a

successful transition to that principle is transparency on the one hand, and a competition watchdog on the other hand. Otherwise, stakeholders might opt out of the process as their efforts appear futile, and rumors might arise: Are HiPPOs driving the selection process? Or will those who shout the loudest be privileged when ideas make it into the backlog of experiments? Do I have to pitch my ideas at 2 a.m. in a bar to the product owner to be heard? Make sure that everyone understands how and why decisions are made. Visualizing both the hypotheses funnel as well as the backlog of experiments has proven to be supporting.)

- **No feedback process**: There is no process how the product delivery organization deals with suggestions for new features or other ideas, and feedback is provided randomly at best. (It's a good idea to have a process that actively involves stakeholders and members of the organization in the product discovery process. People like to have a purpose and to be a part of something larger than themselves. Providing everyone in the organization an opportunity to contribute, without regard to rank, makes working as a product owner easier. Such a process doesn't require fancy technology—a simple shared spreadsheet or form is enough to get it started. Initially, a template to suggest new product features could consist of questions that address the why, the how, the what, and the for whom. It could address the tactical or strategic nature of the suggestion, a possible time-frame, or an estimate of the expected return on investment. Most importantly, designing a process to handle product ideas should be kept agile: the process should start with a simple solution, and then be improved once initial feedback from stakeholders on the process has been analyzed.)

- **Ignoring sales**: The sales organization is not effectively included in the hypotheses creation process. (Beware, though, of the tendency of salespeople to demand new features once the quarter is nearing to meet sales targets to secure bonuses. This opens the discussion ns of bonuses is general; a learning organization should not provide individual financial incentives. The possibility of a moral hazard is otherwise too large: Using the organization's resources for free to meet objectives that are personally beneficial is all too human. Do you remember the origins of the financial crisis from 2008?)

- **Ignoring customer support**: The customer support organization is not adequately included in the hypotheses creation process. (To my experience, cooperating closely with the customer care team is most beneficial for any product delivery organization. Failing to include them is one of the worst product discovery anti-patterns.)

- **Hypotheses selection without engineers**: The product management does not include engineers early in the process of identifying potential hypotheses to test. (Idea is worth nothing, execution is. Usually, a product delivery organization is flooded with ideas from everywhere: customers, sales, internal stakeholders, HiPPOs, friends, and family—you name it. The most important job of the product people is

hence bringing order and transparency to the chaos by establishing a process that turns the long list of ideas and requirements into a short-list. The purpose of this short-list is to run experiments that verify or falsify short-listed ideas as only valuable, usable, and feasible ideas can become a part of the product backlog. To do so, it has proven useful that a triumvirate from product management (business), design and engineering decides which of the items on the hypotheses shortlist make it into the backlog of experiments. Excluding the engineers from that decision process flaws the whole process from the beginning.)

Product Discovery Anti-Patterns: Egos at Work

- **HiPPO-ism**: The product discovery process is at least partly driven by the beliefs of individuals of the higher management caste. (This can be best described with the famous quote from Jim Barksdale, the then CEO of Netscape: "If we have data, let's look at data. If all we have are opinions, let's go with mine." Think about it: how does the pay-grade, and hence the individual's position in the social hierarchy of the organization, qualify that person to have the 'right' idea? The Macintosh was a brain-child of Steve Jobs, and he deserved credit for that. He also almost killed it prematurely by refusing to add expansion slots to the design. I do believe that the Macintosh II made the iPhone possible.)

- **'My budget' syndrome**: Stakeholders do not pitch for development resources but claim that they allocate "their" budget on feature requests as they see fit. (That process leads to the creation of local optima at a silo or departmental level. The effect can be observed particularly in organizations, that tie additional benefits to individuals. Instead, resources need to be allocated in the spirit of optimization for the whole organization. Note: 'Pet projects' also fall in this category.)

- **Sunk cost fallacy**: "We have already invested so much. Let's finish it." (No matter how much has already been spent, past expenditure does not justify continued work on a product that provides little or no value.)

- **Bonus in limbo**: We are nearing the end of a quarter. Bonus relevant KPIs (key performance indicators) are at risk of not being met. The responsible entity demands product changes or extensions in the hope that those will spur additional sales. (This behavior is comparable with the 'what features do you need to close the deal' anti-pattern, but it demanded in more pressing fashion, typically four weeks before the end of a bonus period.) Financial incentives to innovate: The organization incentivizes new ideas and suggestions monetarily. (Contributing to the long list of ideas and thus the hypotheses short-list should be intrinsically motived. Any possible personal gain might inflate the number of suggestions without adding value.)

Conclusion

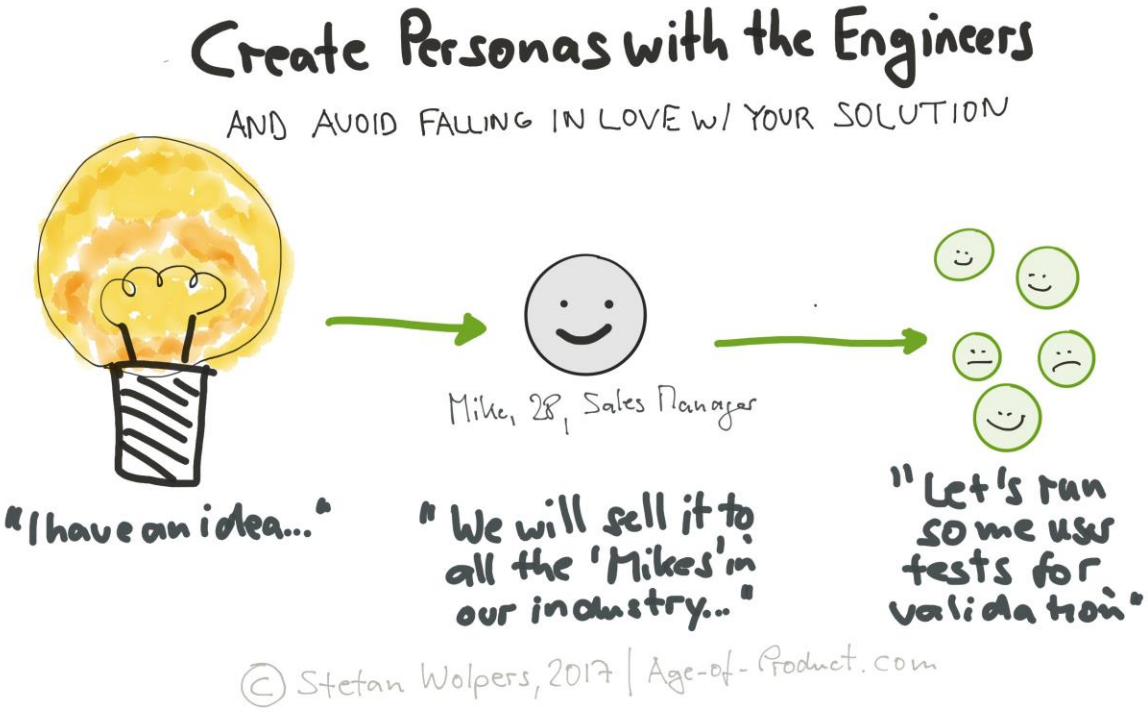
A holistic product discovery & delivery process is at the heart of any learning organization. Filling the product discovery void of the product delivery framework of your choice becomes, therefore, is a necessity. And there are plenty of agile frameworks and methodologies to choose from for that purpose. Watching out for the product discovery anti-patterns listed above will save your organization significant resources in doing so.

Create Personas with the Help of the Engineers

Introduction

Creating valuable software requires knowing the customer—we all agree on that, right? The first question that then comes to mind is how to support this product discovery process in a meaningful manner in an agile environment? And the second question follows swiftly: who shall participate in the process—designers and business analysts or the engineers, too?

Read on and learn why personas are useful for product discovery purposes, how to create personas, and why the complete team—including the engineers—needs to participate in their creation.



Definition: What Is a Persona?

The term was introduced by Alan Cooper in his book *The Inmates Are Running the Asylum (1998)*. (Source: Wikipedia.)

A persona is a fictitious character based on user research that represents a group of users or customers interacting with the product or service.

Benefits: Do We Need Personas?

Personas prove to be beneficial supporting your team by understanding: Whom are they building (a product or service) for? Personas create empathy by showing a human "face" and giving an anonymous entity a name and a background. Personas introduce the "customer" to the day-to-day conversations of the product team. In other words: personas are a useful part of a product's narrative.

Contrary to a piece of literature, though, the narrative of a product is neither "fixed," nor is it created in advance by an author. It is vague in the beginning, and it may become more concrete over time—if you are lucky to achieve product-market fit. Hence, the product narrative is always subject to change of the life-span of the product. Moreover, the best product story is created by a diverse group of observers and authors—the whole team, including the engineers.

Critique: Do We Really Need Personas?

The most common critique is that creating personas would eliminate the opportunity to find innovative use-cases. No one needed Twitter. Hence, no user research, no focus group ever presented this disruptive chance to anyone. Personas thus tend to favor waterfall-ish incrementalism instead of disruptive innovation. Lastly, personas come at a price. Preparing and running the research results at the cost of delay as well as opportunity costs. Nevertheless, the author is convinced that in a majority of all situations the advantages of creating personas as an integral part of the product discovery process outweigh the costs.

Make Personas a Team Effort: The Persona Team Workshop

Preparing a Personas Workshop

If personas are a worthwhile investment, what are the prerequisites of a persona team workshop? It turns out we need:

- Upfront market research, both quantitative (➜ surveys) as well as qualitative research (➜ user tests)

- An analysis of the research findings
- Probably, a persona template tailored to support the product discovery process
- And all stakeholders need to attend the workshop to eliminate the risk of bias.

Designing a Personas Template

A personas template for a B2B product could comprise a subset of the following elements:

- A persona photo: it is most engaging part
- A biography: name, age, education
- A quote to provide insight into the feelings and personality taken from user research
- A company: name, size, and industry
- The current role as a unique identifier among all other personas
- The context: where will the application be used?
- A goal: why is the customer hiring our product?
- A journey: To achieve his or her goal, what will the customer do now?
- The motivation and inhibitors of the customer.

Additional questions for the persona template could be:

- The persona's relationship with other products/brands
- The persona's previous working experience
- The persona's most significant failures in previous jobs or projects
- The persona's career plans.

It has also proven helpful to create a persona matrix that relates all personas of the application to each other.

Lastly, Dave Gray's empathy mapping has proven to be an alternative route to creating personas instead of using a persona template.

How to Run a Persona Workshop with the Whole Team

People care about what they help create—this principle also applies to every part of the product discovery process. Therefore, it is vital that whole team—is a smaller set-up—or a fair number of representatives from each team participate in the workshop that creates personas. The following workshop agenda has proven to be useful:

- Start by dividing the large group into smaller groups of two to four participants each to introduce either your organization's persona template or consider Dave Gray's empathy mapping
- Now ask every group to create "their" persona. Set the time-box to 20 minutes

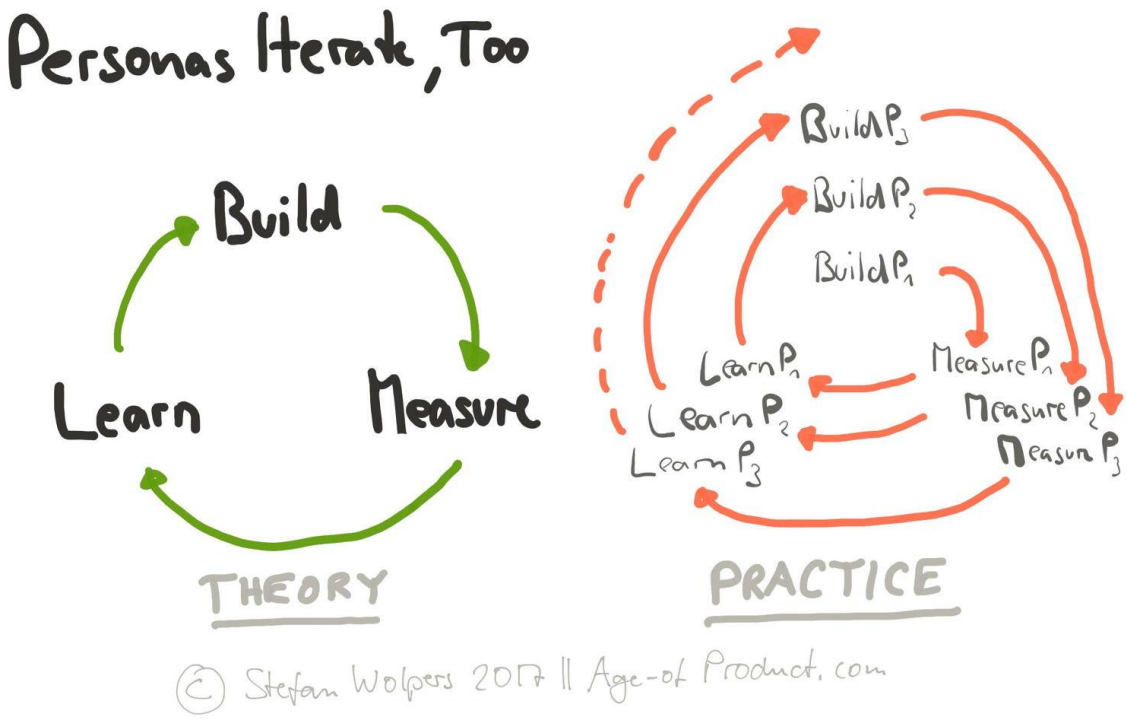
- After the 20 minutes, all groups shall present their persona drafts by telling that persona's compelling story. This moment will take two to three minutes per group
- Next, compare all the different personas and identify patterns. For this purpose, all small group come together again as one large group
- Finally, merge similar drafts in a joined effort to form your persona in version 1:
    - Merge two draft personas ➜ your persona v1
    - Merge four draft personas first into ➜ two draft personas and then those into ➜ your persona v1
    - Merge six draft personas first into ➜ three draft personas and then those into ➜ your persona v1.

Making Your Persona v1 'Real'

Now that you managed to create version 1 of your persona you must not hide it in a box but make it visible in the office:

- Print its LinkedIn profile on a poster and put the poster up in a prominent spot in the team-space
- Use the persona's name in daily team discussions
- Based on the persona, reassess both the product roadmap as well as the product backlog regularly
- And most importantly: validate the persona during user interviews.

Note: Ensure that the team iterates on each persona alongside the product development. No persona is a static artifact; personas develop in sync with the product:



Beware of Persona Anti-Patterns, Fallacies, and Self-Fulfilling Prophecies

"I have a killer product idea, and I know exactly what to build for whom. Follow me." There are many ways to fail at product discovery. However, admittedly, falling in love with your solution instead of the customer's problem is the one reason all too human. While it is impossible to rule out bias, we can limit the adverse consequences by obeying a few simple rules:

- Never let one individual define personas upfront
- Never limit participation in the persona workshop to business analysts and UX designers
- Always collaboratively merge drafted personas
- Always iterate on personas over time
- Always embed personas in the team's operational work.

Note: It is essential that you practice checks & balances with the whole team to avoid unconsciously selecting "users" that "validate" your beloved product idea. This is the

rationale behind including all stakeholders—even the “expensive engineers“—in the persona creation and iteration process.

Conclusions—Create Personas with the Help of the Engineers

‘Take it to the team’ is a proven tactic that applies to everything related to product creation with a cross-functional team that works in an agile way. The idea to create personas without the help of the engineers is hence not only counterintuitive. It also reveals that the organization in question seems to be clinging to outdated functional silos. Agile up—the earlier the engineers are involved in the product discovery, the sooner the team will learn to abandon stupid product ideas.

Product Delivery

How to Kick-off Your Agile Transition – Team #1

The Big Picture of an Agile Transition at a Fast-Growing Startup

Similar to all other journeys, transitioning to agile software development with Scrum starts with a first step: Creating Scrum team #1 from the existing engineering and product organization.

I have started this journey recently with a product delivery organization that is about 30 people strong. And I like to share some of my lessons learned along this path over the coming weeks.

This organization is part of a startup with great potential. This is reflected both in its growth, as well as in the quality of the investors, and its funding. However, like most other fast-growing startups, it hasn't been a smooth ride all the time. And this effects both the engineering and product team on the one side, and the rest the of the startup organization on the other side.

One effect of scaling a product delivery organization quickly by hiring new members has proven to become particularly challenging: The predictability of any product delivery is deteriorating, and allocating more manpower to the problem—without changing either mindset or processes—is not solving the issue.

This problem has led to the perception within the organization, for example, that the engineering and product team isn't really aligned with the rest of the startup. And the consequence is well-known: "Us vs. them" silo thinking, local optimizations efforts, and politics.

And this is where the agile mindset comes into play. Contrary to today's popular approach to introduce agile tools and processes primarily to increase efficiency, the intention of this startup's management is a different one:

1. Embrace self-organization and permanent improvement at team-level to enable a sustainable scaling of the organization
2. Better the value proposition for the customers
3. Further the organization's future product discovery capabilities
4. Advance the product delivery organization's general standing with all other stakeholders by inclusion, transparency, and building trust.

Getting started with the Agile Transition: Identifying Patterns of Failure and Dysfunctions

Where to start then? A good rule of thumb for an agile transition is to go deep first, before you try to scale. My preferred choice is therefore being the observer and listener. I am participating in all meetings, and try to understand what has contributed to the current culture.

I also ask to have one-on-one interviews with as many team-members as possible, too. I intentionally don't run a script during these interviews, but rather let the interviewees talk, and in some cases: rant.

I like to stress at the beginning of these interviews, that I am just taking some private notes, and that no one will ever get to read those. I also encourage my interviewees to "interrogate" me first, thus building a minimum level of thrust.

Interestingly, it seems to be a myth that engineers are uncomfortable with being part of that. On the contrary, they like to point straight at the problems of the organization.
If the interviews are evenly covering all roles—from engineers, QA, UX/UI, to product managers—it will take about 10 interviews before patterns of problems, failures, and dysfunctions appear.

Merge those patterns with the most pressing technical and business issues, and you end up identifying the most likely first objective for the agile transition. This could, for example, be a project or specific software release, that is late, or plagued with technical or functional issues.

If your stakeholders agree, it will be the problem that Scrum team #1 will be tackling.

Looking for Volunteers for Scrum Team #1

Dealing with organizational problems at such a level won't work with people pressed into service—at least not to my experience.

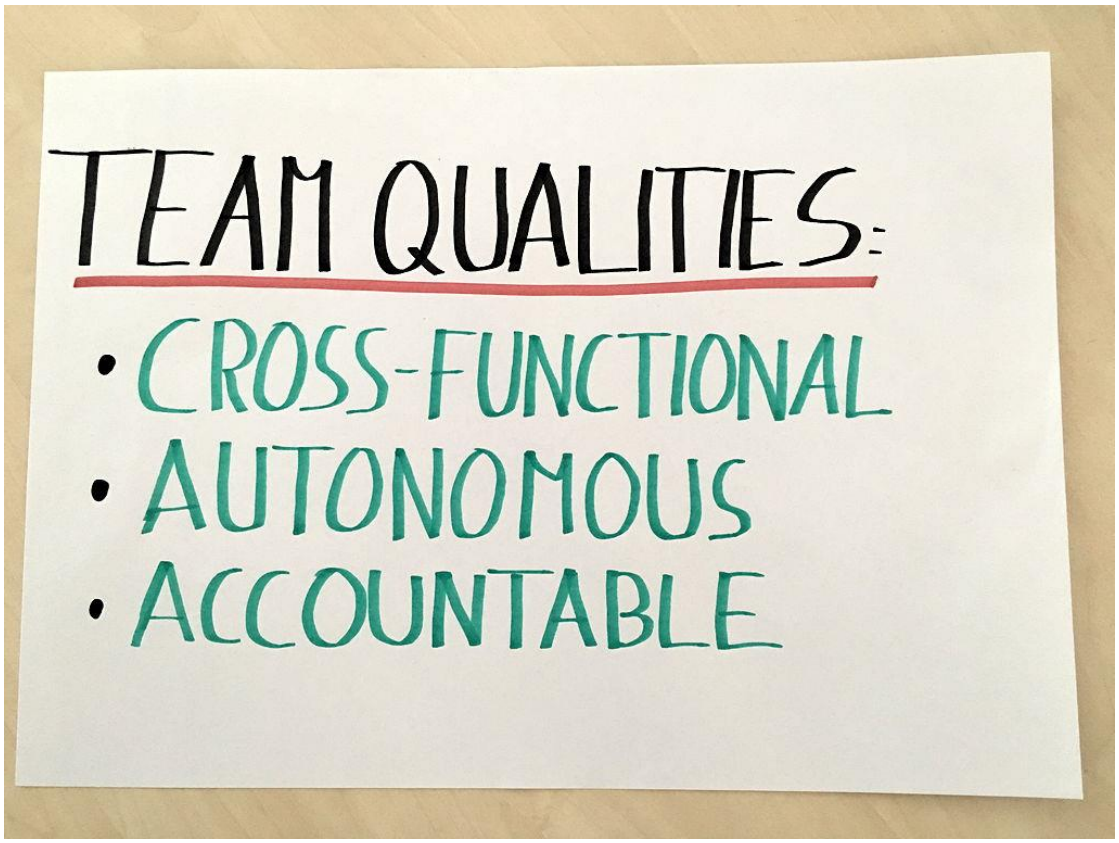
You need volunteers instead, who fully understand the challenge ahead of them. Volunteers, who are eager to prove that there is a better way to reach the objective than being "managed" by someone from the hierarchy.

You need volunteers, who are willing to move out of their comfort zone, embrace the idea of self-organization, and accept accountability. You want those, who are aware today that focusing on delivering value will move the entire organization forward and create a scalable, competitive and sustainable business in the end.

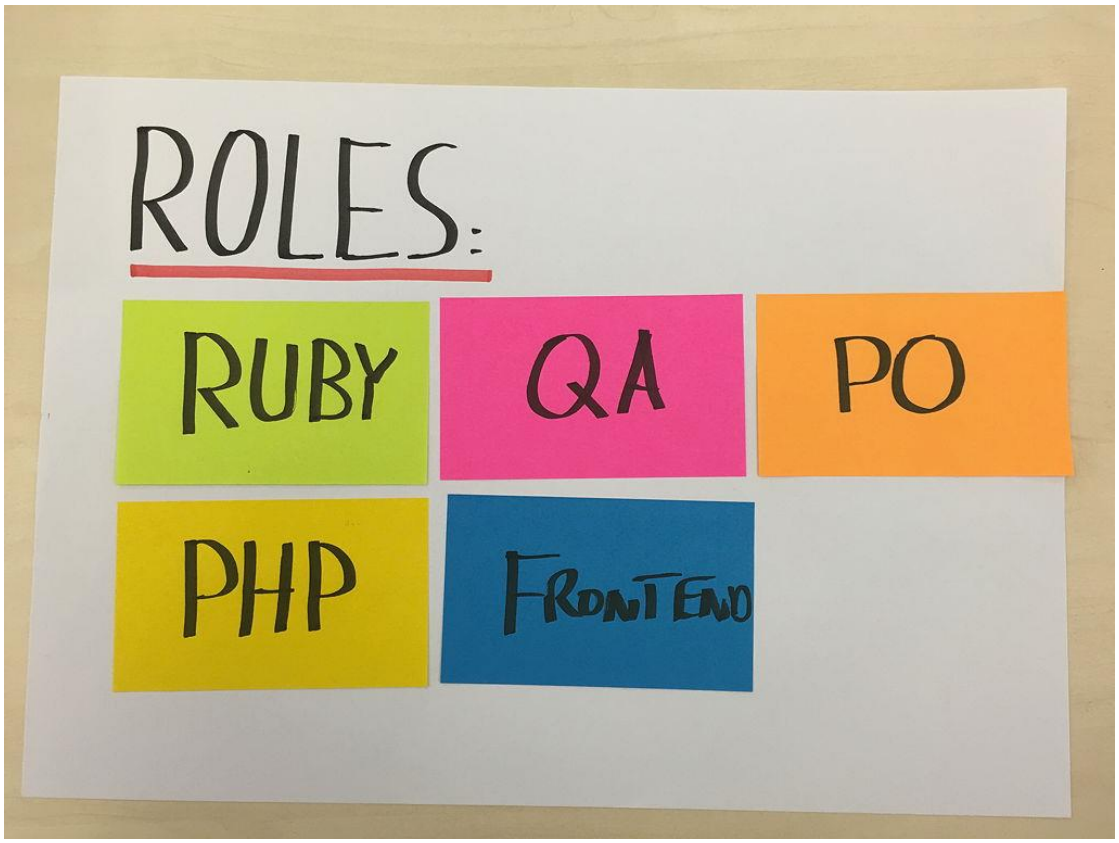
So, set the stage for the creation of Scrum team #1 by inviting everyone from the product delivery organization, as well as your sponsor—probably the CTO— to a kick-off meeting. This meeting's objective is to support the members of the engineering and product team to self-select a draft of Scrum team #1.

And here is the recipe that has been working for me several times by now:

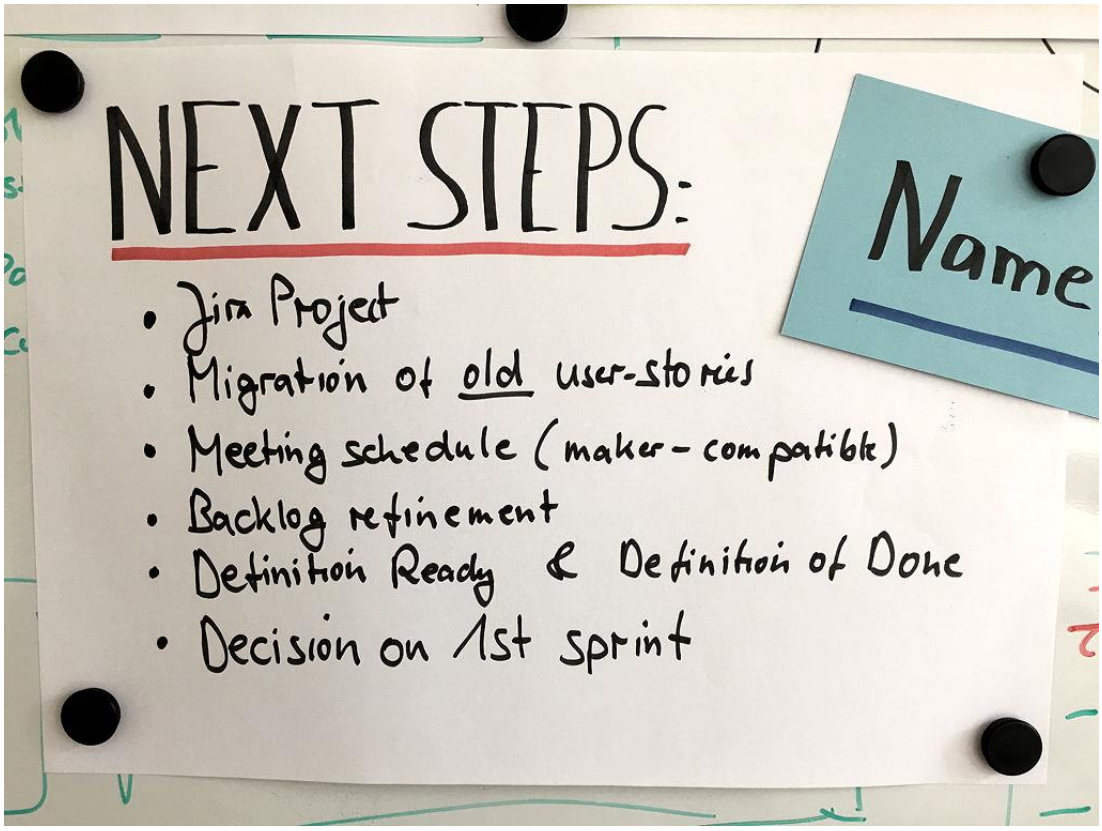
- Choose a meeting room that can accommodate the attendees easily and allows to move freely in front a whiteboard.
- Time-box the meeting. The meeting, I am referring to in this post, took 60 minutes and was attended by 15 people. Prolonging the meeting—as I tried before at another project—will most likely only provide room to discuss minor team variations without providing radically new insights worthwhile this investment.
- Ask your sponsor to define the mission of Scrum team #1, and why this goal is so important for the company. (5 minutes should be sufficient for this prelude.)
- Then deliver the context of the agile mindset, self-organization, and the team's autonomy on the one side, and the accountability of the team on the other side. (Don't expect that everyone is already familiar with agile/lean principles.) Also, don't be afraid to borrow from the DevOps "You build it, you ship it, you run it"-mantra to explain what they are getting into. The team needs to meet that standard in the future anyway. This sounds a bit like lecturing, but these 5-10 minutes are well invested to align everyone in the room with the "agile mindset":



- Introduce all roles, that the team will need to cover: engineers, DevOps, UX/UI, or product owner, for example. If you assign a specific color to each of these roles, the selection process of the volunteers will become much simpler



- Now ask all participants to write their names on one of the respectively colored Post-its and stick those to the whiteboard, if they want to volunteer for Scrum team #1. (In total, all of these preparations should consume no more than 15-20 minutes.)
- Now let the self-selection process start. Set the timer to 30 minutes and be surprised by the outcome. (Think George S. Patton: "Don't tell people how to do things, tell them what to do and let them surprise you with their results.")
- During the closing stage of the meeting, wrap-up the goal of the future team, its autonomy and also its responsibility for the outcome.
- Point at the next steps for the team—agreeing on a Scrum ceremony schedule, create the first backlog, start the backlog refinement process, find a team name, to name the most important ones—and ask all participants to sleep a night over the first team draft:



- Also, sketch a transition roadmap for those team members, who did not make it into Scrum team #1. (Or didn't want to be a part of it.)
- Check with your sponsor, whether the draft of Scrum team #1 is meeting her expectations.
- Last, but not least, if your sponsor supports the draft: Organize a follow-up meeting the next day and feel the pulse of the future team-members of Scrum team #1. During that meeting, revisit the next steps and agree on the further proceedings.
- If the draft of Scrum team #1 is not meeting your sponsor's expectations, start over with the self-selection meeting.

Conclusion

Some of you may wonder, whether a meeting of 60 minutes is actually adequate, given the task and its implications. To my experience, it is. Let me explains this: On a former project, to which I contributed as a product owner, the team set-up was organized as a 2-days-workshop. While workshop was really fun, we practically ended up with the same team set-up we all expected beforehand.

So, according to Mr. Tuckman, my Scrum team #1 passed the forming stage. The storming, norming, and hopefully performing will happen one way or another along the team's agile journey. More on this journey in the following chapters.

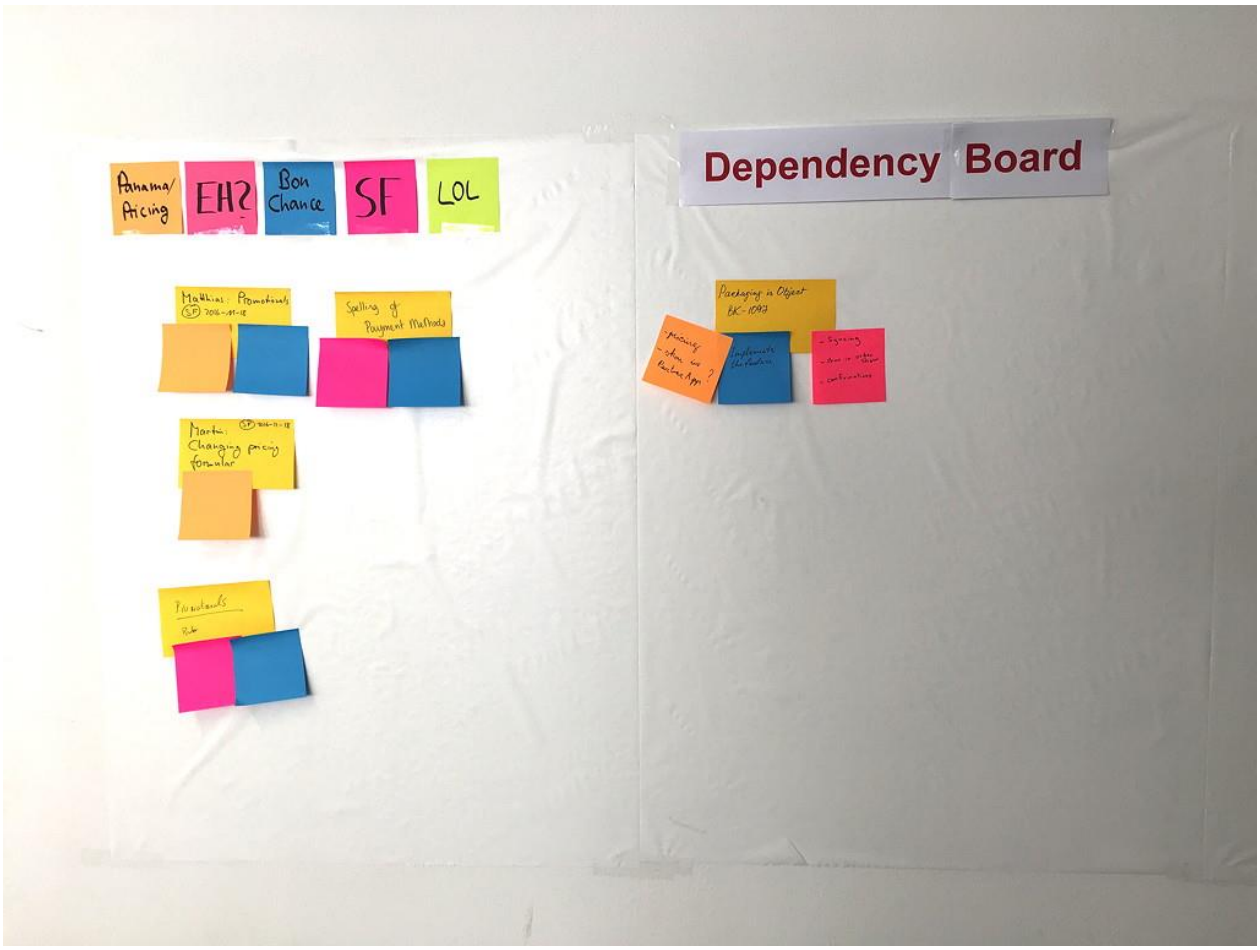
How to Align Scrum Teams

TL;DR: How to Align Scrum Teams

Do you remember the good old days when the organization started with its first Scrum team? And the new engineering kid on the block was “merely” supposed to deliver a potentially shippable product increment at the end of a sprint?

The first team was to sound the bell for the upcoming change towards a learning organization. Little did we know back then about the challenges along that route. When teams 2, 3 and 4 joined, shipping a product increment at the end of a sprint became first complicated, and then complex.

It turns out that becoming agile does not only required to create (Scrum) teams. Reaping the full benefits of becoming agile, of becoming a learning organization built around software also requires changing engineering practices. Nowadays, it is all about continuous value delivery.



Why it Is Beneficial to Align Scrum Teams

There are two levels of alignment that a product delivery organization needs to address when scaling Scrum:

Alignment at Product and Process Level:

Which team builds what? The alignment at product level is required to avoid delivery optimization at the individual team level. This form of local optimization is a familiar pattern with component teams, as they are specializing in a particular technical area. Working in this field of expertise, however, may not always deliver the most value to customers or the organization at a given moment. A task from another team that this team cannot attend to might be much more valuable by comparison.

Alignment at Technical Level:

This level is addressing engineering practices such as continuous integration as well as continuous delivery. It is also relevant concerning the software architecture. In most cases, it is the monolith v. micro-service architecture debate. (Read more: REST in Peace: Microservices vs monoliths in real-life examples.)

Signs That Your Efforts to Align Scrum Teams Are Failing

If the experiment with the first Scrum team proves successful, organizations will often start creating more Scrum teams.

No matter your engineering practices, you will quickly discover that the communication overhead will grow by comparison exponentially. This overhead growth is particularly visible to organizations that create component teams and that are working on a monolithic application.

Though the need for better communication among the Scrum teams is a low-brainer to everyone, it is interesting to observe the communication inertia between teams—even if located next to each other.

Some of the typically resulting alignment anti-patterns at the beginning are:

- 'They'—meaning another Scrum team—touched 'our code.' (Without approval from us: a perceived on even encouraged code ownership at the team level.)
- Deployments by individual Scrum teams happen without prior communication to the other teams, thus breaking things. ('Was working on our staging system' syndrome.)

- There is no regular sharing of knowledge among teams, for example, training colleagues on new code if not ordered by the management.
- Scrum of Scrums meetings are not attended. (Common excuses: a) “Scrum has so many meetings, when am I supposed to finish my work?” and b) “Our issues are not affecting any other Scrum team.”)

The Necessity to Improve Inter-Team Communication

The cause for the resulting communication overhead is dependencies between teams resulting in creating queues, waste and, thus, costs of delay.

To reduce the number of dependencies between Scrum teams, they need to be autonomous. However, autonomy without accountability equals anarchy. The process of aligning Scrum teams, therefore, needs to start at the organization’s cultural level.

Improving the culture cannot by ordered by edict from the C-level. It needs to be a journey that includes everyone affected. If Scrum teams are to move out of their comfort zones and accept accountability, failure must be a safe experience, and radical transparency needs to be embraced.

The cultural change which is required to align Scrum teams at the process level is ideally backed by a simultaneous transition to an anti-fragile or resilient system architecture. Technical failure—no matter the effort invested upfront—will be inevitable during this change process.

The effective way to deal with failure at a system level is, therefore, not to focus on preventing it from happening at all. The effective way is to create a fault-tolerant environment that excels at rolling back hazardous deploys. (More on high-performing teams in agile organizations from Jez Humble: GOTO 2015 • Why Scaling Agile Doesn’t Work.)

First Improvised Steps to Align Scrum Teams

Changing engineering processes as well as probably the architecture of an application will take time. There are some preliminary steps, though, to start the alignment process immediately:

- First of all, avoid individual sprint lengths, and start-dates, but align the sprints of all teams instead.
- Establish a ‘Scrum of Scrums’ ceremony, as well as a dependency board. (Both measures are temporary training wheels on the way to fully autonomous and aligned Scrum teams.)

- Apply the LGTM ('looks good to me') process before releases: This sounds like bureaucracy but is unavoidable if otherwise disaster is lurking around the corner. (Interesting question on the side: Will the Scrum teams address this issue themselves, or wait for management to 'solve' it?)
- Establish a peer teaching and coaching program. A simple pair programming board—who wants to learn what from whom—can be a good start.
- Track technical debt with a repository of issues that need to be fixed, shared and maintained by all Scrum teams.

Ultimately, Align Scrum Teams by Moving to Autonomous Feature Teams

In the end, at least this is my opinion, aligning Scrum teams will only work based on autonomous feature teams. Any other set-up will create too many dependencies, thus wasting money, brain, and time. It will also prevent the organization from becoming a learning one.

If you cannot move to features teams yet, here are some observations that might ease the transition:

- Continuous integration is a prerequisite for feature teams. This process needs to work flawlessly
- You build it, ship it, and run it: Everyone on a feature team needs to do "QA," avoid siloing quality assurance or delegating it to an individual team member
- If you continue using feature branches, make tasks as small as possible and try to merge all branches back to master in the evening. This practice will provide everyone with instant feedback, and fixes will be less expensive (and painful)
- Start measuring lead and cycle time to identify queues. (Read more: Agile Transition: Agile Metrics—the Good, the Bad, and the Ugly.)
- If you're working on a monolith, consider moving to a micro-service architecture if that is feasible.

(Legacy) Product Backlog Refinement

Where to start when kicking-off an agile transition?

Usually, tools and processes are smallest the common denominator among all participants, as they are at the core of the grand scheme of agile things.

It is a rare occasion that you start from scratch with a brand-new team without an existing product, probably even in a more or less nascent organization, for example a startup.

In most cases, an existing product delivery organization with available products, and services will go “agile“. In this case, turning attention to the available product backlog is a pragmatic first step. The following process describes what aspects need to be attended to to optimize the outcome.

Refinement of the Existing Product Backlog

Starting the agile transition with an extended product backlog refinement session has also proven to be an effective way of coaching a new team. Ceremonies, practices, and artefacts, as well as agile principles can be introduced “on the job”, so to speak, using break-out session whenever required.

Such an initial refinement workshop may require—depending on the maturity and size of the organization and its products—several hours or even several sessions.

As far as the kick-off in general is concerned, the product backlog per se is an interesting artefact as it provides a deep dive into the product delivery organization’s history. Almost like an archeologist digging through layers of an ancient civilization, you can identify organizational debt, process insufficiencies, problematic product decisions, and other anti patterns by sifting through the backlog.

Product Backlog Anti Patterns

Typical anti patterns that can be found in legacy product backlogs are, for example:

- The product backlog is structured is a waterfall manner: There are no team projects but solely projects spanning the whole organization or division to ease the reporting process. (The “70% finished” syndrome.)
- The product backlog is a long, unstructured list of ideas, features, bugs, minor technical tasks, change requests, and epics—and thus practically not in a state anyone could effectively work with

- The prospective value of tasks is not reflected by a ranking, as prioritization is due to the sheer size of the product backlog no longer an option
- There is no consistent quality in building user stories, e.g. based on Bill Wake's INVEST principle
- There is no "Definition of Ready" applied to user stories
- A significant proportion of user stories consist of barely more than a headline without any additional information on the "why" or "how"
- User stories are being kept in the product backlog for months without being improved over time, accomplished, or archive for lack of value
- Product backlog items are being created directly by everyone, e.g. stakeholders, and not just the product manager or Product owner
- Product managers already create (sub-)tasks and assigned them directly to developers.

How to Apply Structure to the Product Backlog

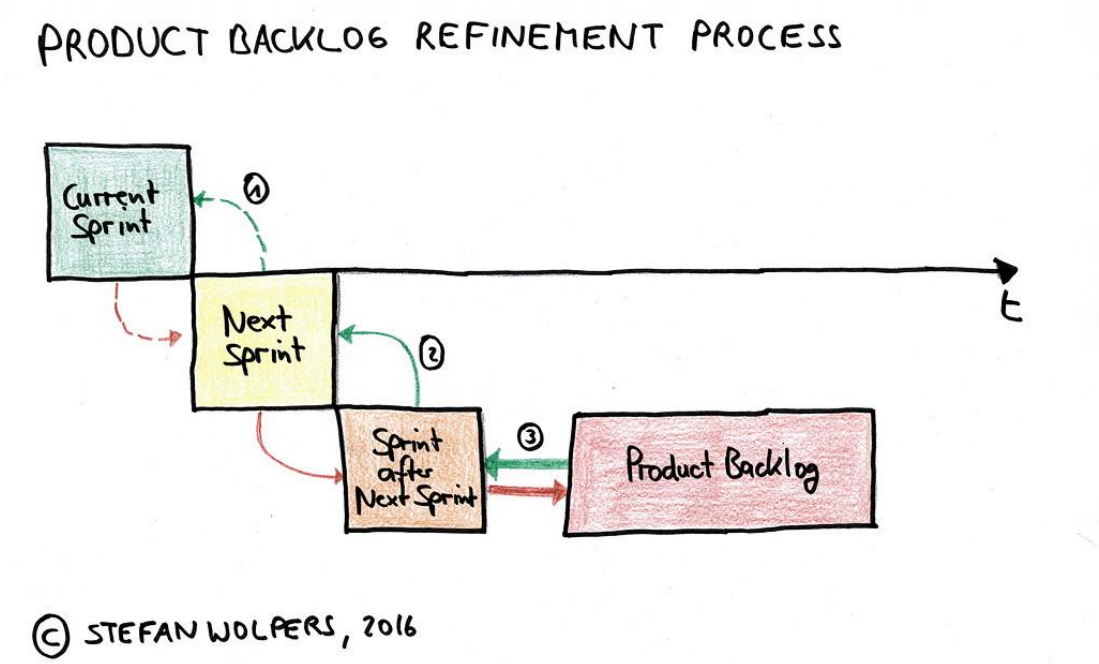
How to get from potentially hundreds of unstructured "user stories" hidden in the legacy product backlog to a valuable product backlog in an agile sense?

To my experience, there is only a single way to achieve this: Inspect each item in the legacy backlog and decide collaboratively with the whole team whether the issues is worth working on or not.

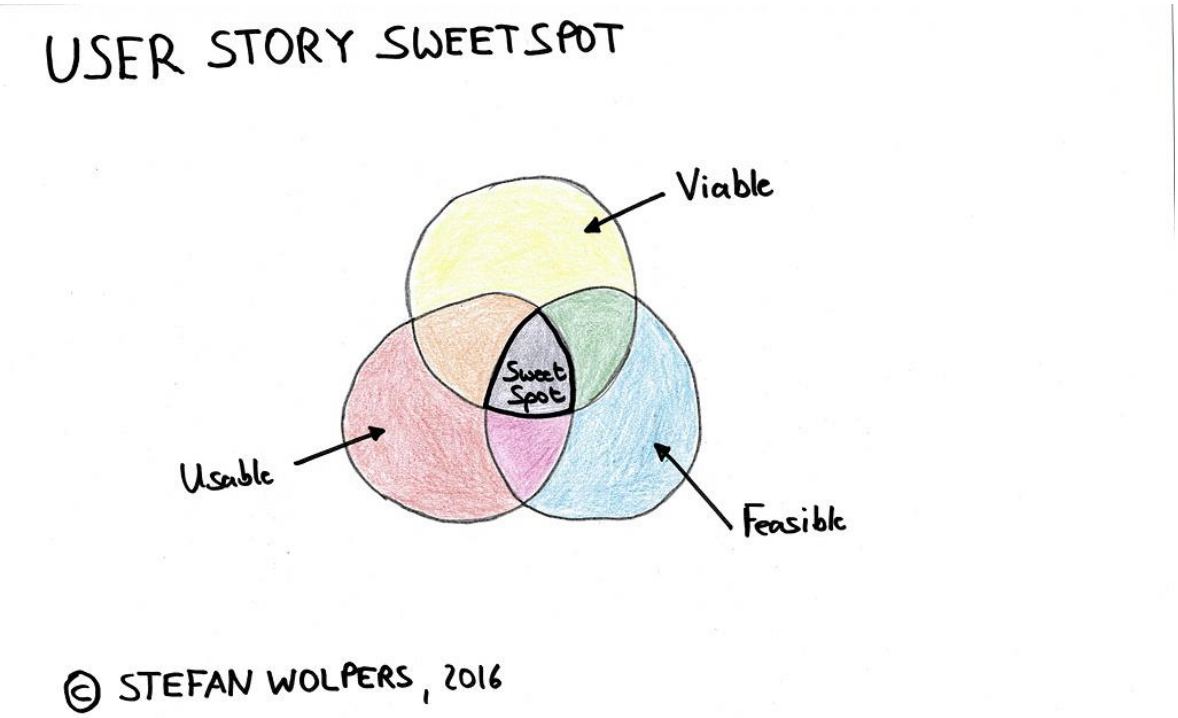
A process for this procedure may look like the following:

1. **Step 1:** Identify all relevant tasks for the newly created team—probably distributed across several projects—, and migrate those deemed valuable to the team's new project.
2. **Step 2:** Kick-off the refinement process with creating shared understanding on:
   - The purpose of the team's product backlog in general
   - That the product backlog isn't supposed to be a bucket for each and every idea
   - That a valuable product backlog covers rarely more than a period of maybe 6-8 weeks.
3. **Step 3:** Introduce the product backlog refinement process, why it is beneficial to divide the original Scrum sprint planning ceremony and create a separate backlog refinement ceremony:
   - To defuse the sprint planning meeting in the first place. (Who wants to spend eight hours on that?)
   - To be more flexible responding to change while working on user stories
   - To provide the Product owner with better options to answer questions of the Scrum team in a timely manner. (It's easier to do that once or twice a week than every 2 weeks, for example.)

- To create a smooth process including the graphic and UX designers. (Creating deliverables collaboratively becomes thus much simpler.)



4. **Step 4:** Introduce the Scrum team and the Product owner to user story best practice, for example Bill Wake's INVEST principle. Visualize the sweet spot of value, usability, and feasibility. The team needs to embrace the idea that a user story is primarily a token for discussion.



5. **Step 5:** Create with the Scrum team and the Product owner their version of a "Definition of Ready"
6. **Step 6:** Kick-off the estimation process by introducing the "Estimates are nothing, estimating is everything" idea:
   - That estimating is all about knowledge transfer, and creating a shared understanding of the "why" a user story is valuable, and the "how" it might be realized
   - That estimates are a metric for the Scrum team
   - That estimates are defined in relative sizes—t-shirt sizes, Fibonacci figures—not man-hours
   - That there might be a discussion ahead with the perception of velocity by stakeholders. (It might trigger attempts to micromanage the team, or determine delivery dates: Scrum: The Obsession with Commitment Matching Velocity.)
   - That there are useful agile metrics available. (Read more: Measurement For Scrum – What Are Appropriate Measures?)
   - Start using estimation poker on all required stories.
7. **Step 7:** Explain how dealing with technical debt fits into the game, so that the Scrum team ensures that a fair share of each sprint's resources are allocated to refactoring.

Tip: When there is (an urgent) release scheduled in the near future—say in 8 to 12 weeks—that needs to be met, and the Scrum team is already on a tight schedule, the initial backlog refinement will likely be not ideal from a Scrum Master's perspective.

However, insisting in such situations on a textbook approach—which inevitably will result in a delay of the release—is likely to cause serious communication problems. The management, and particularly those managers and stakeholders that might have opposed going agile in the first place, might consider their worst fears confirmed. Be pragmatic, and think twice before picking your battles.

Conclusion

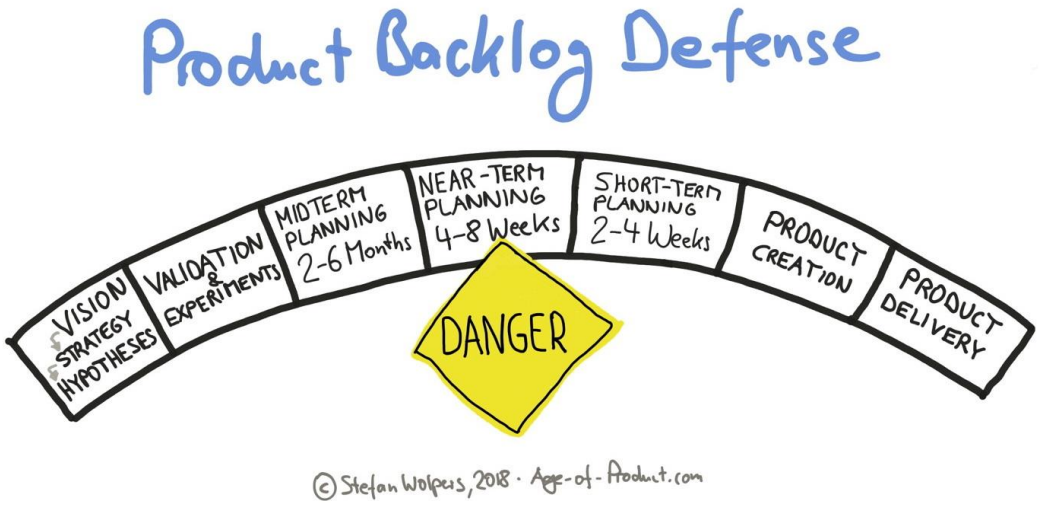
Kicking off the agile transition in an organization with an existing product backlog by a thorough refinement session is a proven technique. It is a good exercise to train the new Scrum team on the job in artefacts, ceremonies, and agile principles.

Be careful, though, to practice "Le Scrum pour le Scrum", particularly when an urgent release is imminent. This might cause unnecessary irritations with stakeholders.

Product Backlog Defense

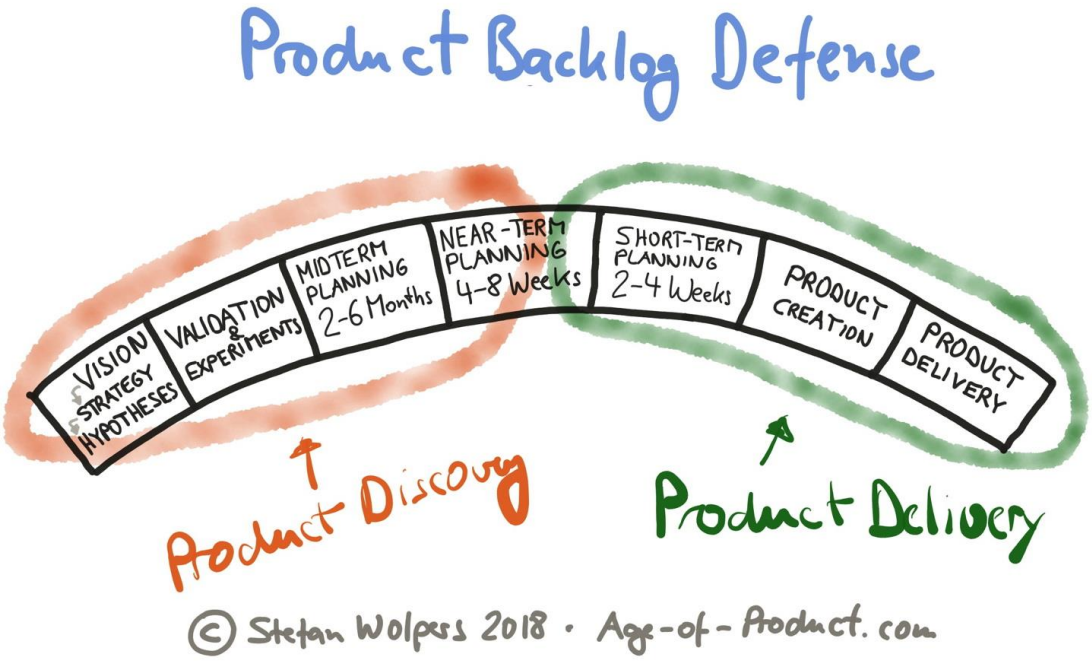
Why the Product Backlog Needs Defense

Product backlog defense — make no mistake: Your product backlog is the last line of defense preventing your team from becoming a feature factory. Figure out a process that creates value for your customers. Moreover, have the courage — and the discipline — to defend it at all costs.



The Product Backlog in the Context of the Product Creation Process

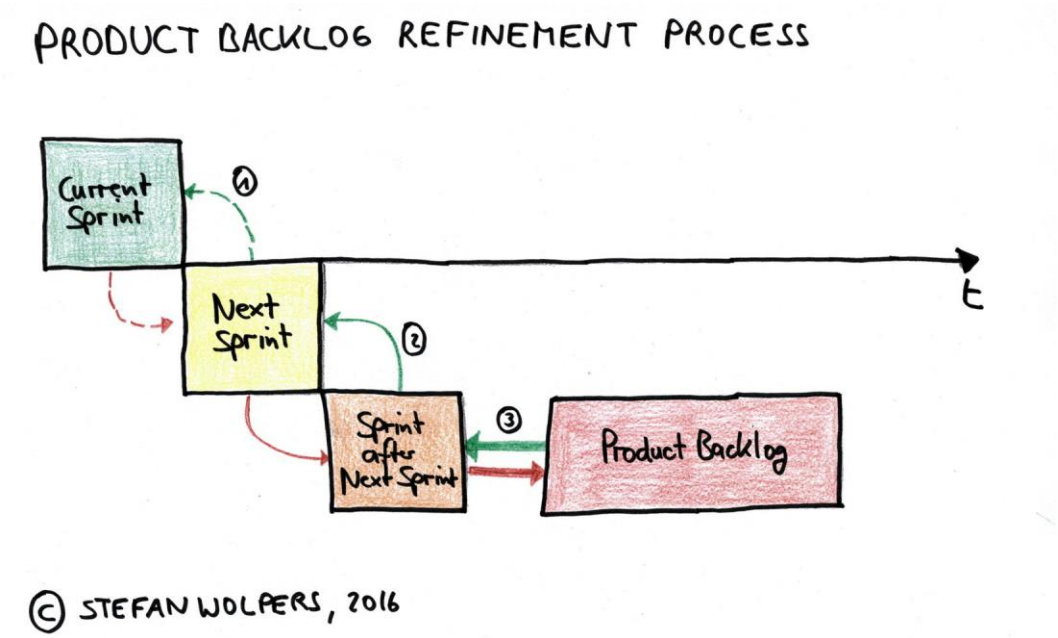
In my keep-it-simple and thus two-dimensional product world, the product backlog defines the near-term planning as the lynchpin of the product creation process:



The near-term planning horizon in my mental model covers about four to eight weeks. It is where the product discovery phase delivers validated ideas of valuable product increments to the product delivery phase. (I shy away from calling it a hand-over as this term has a negative connotation rightfully.)

At this moment, there should be no longer doubts why a product increment is valuable to your customer and your organization — the 'why' question has been answered at this stage. Probably, you will still discuss the scope of the idea for its first delivery, and the engineers will think about how this product increment will fit best into the application. However, the discussion among the team members should no longer revolve around basic questions from the 'valuable, feasible, usable' perspective. That has been accomplished further to the left side during the product discovery phase.

If you are practicing scrum, the near-term planning will cover something between two to four sprints:



This transfer from product discovery to product delivery is a serious moment from the investment perspective. Now, the team gets real and allocates resources to product delivery, for example, the continued collaborative refinement of the related user story. Or sketches are turned into designs, and probably some preliminary work is required on the tech stack side.

Now the team becomes accountable for spending money and producing a return on investment. If you fail to deliver this return because you accepted some work that bypassed our team's product creation process for whatever reason you will be rightfully accountable for this failure, too. And no one will be interested in reading the fine print why this happened. It is on you.

Expect Flanking Maneuvers to Bypass the Product Creation Process

Crossing the before-mentioned threshold is also the reason why the near-term planning or the product backlog part of the product creation process is so attractive to stakeholders who try to bypass the process and sneak in requirements or features.

Entering a competition of ideas and hoping that an idea will pass validation is a considerable effort and bears a significant risk of being rejected. The shortcut of targeting the near-term planning phase instead is hence less risky.

Typically, you can attribute a stakeholder's attempt to evade the competition of ideas part of the product discovery phase to his or her incentives provided by the organization. Or as Charlie Munger puts it:

"Never, ever, think about something else when you should be thinking about the power of incentives."

The Motivation Behind Flanking Maneuvers

There are various patterns of flanking maneuvers, depending on the nature, age, and the size of the organization. For example:

- The case of the sales manager who believes his or her bonus is at risk and hence wants to throw some features urgently at the wall to see what sticks (and generates revenue) is evident.

- The case of someone who is pursuing a specific (pet-) project to improve his or her CV for the next career step - probably with another organization — is more difficult to spot.

- Often you can observe local optimization efforts by stakeholders valuing the results of his or her department higher than

- Then there is the drive to determine what the "internal software agency works on because it is our budget and we know our needs best." Here, an organizational mindset — based on functional silos — of the industrial age clashes with the post-industrial value creation process.

Common Flanking Maneuvers

There are at least seven flanking maneuvers that make the product backlog defense mandatory for any product team:

1. **Pulling rank or HIPPO-ism**: There are different reasons for this kind of behavior. Some people believe that hierarchical status has privileges and rules do not apply to them. Others think that they know best, probably fueled by a paternalistic mindset. Often, it is a sign of lack of trust in the team's capability. No matter what makes an individual behave this way, it is a sign of poor leadership qualities.

2. **Brute-forcing it**: The bully at work, now probably plowing through a corporate hierarchy. What worked in the school-yard might still work today — differently packaged, though. Don't expect this behavior to leave a trail of artifacts or email threads.

3. **Have daddy fix it**: The magic of outsourcing the solution of 'issues' — here: the product team asks the stakeholder to accept the process like everyone else — to a higher hierarchy level. ('My boss will talk to your boss, and then we will see…')

4. **Bribery**: I scratch your back, you scratch mine.

5. **Direct ticket creation**: Why bother the product owner? He or she might be overworked anyway. Instead, your stakeholder shows initiative and creates the ticket for the required feature herself. (Not controlling access rights to the ticket system is a rookie mistake, just saying.)

6. **Dispatcher mode**: Why not assign a task directly to an engineer bypassing the product backlog altogether?

7. **Labeling feature requests as bugs**: By comparison to the other maneuvers sneaking features via bug reports is almost a thoughtful approach — someone tries to hack the system. Nice try. Try harder next time, though.

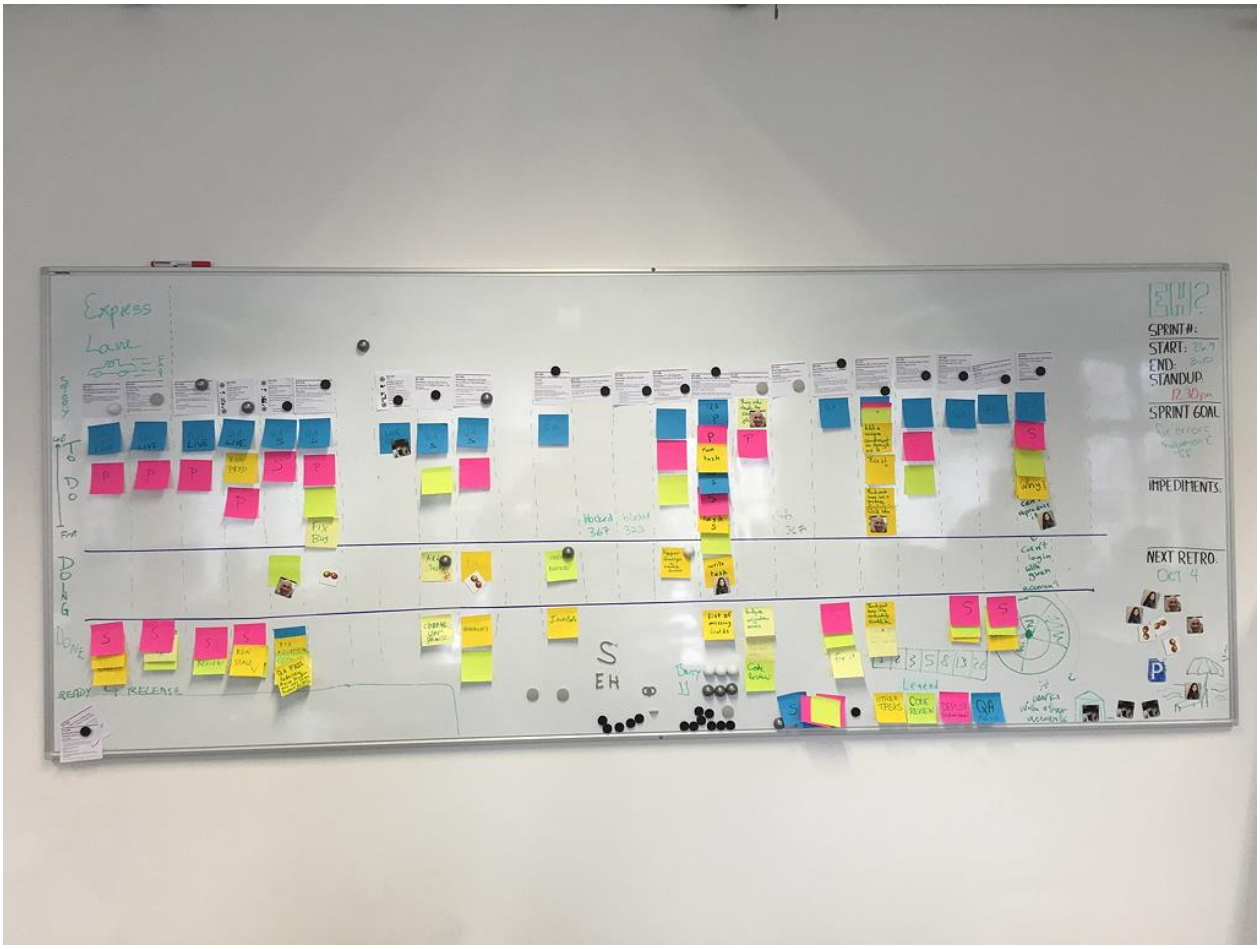
A note of caution: Do not outflank yourself during product backlog defense. The discipline to defend your product creation process from stakeholders' attempts to bypass it needs to be applied with the same rigor within your team. For example, do not use the product backlog as a repository of ideas and requirements that might be useful at a later stage. Apply the same rules indiscriminately to everyone.

Conclusion

If you want to be taken seriously as a product team and if you want the organization to accept your product team as the go-to team that solves critical customer problems and identifies opportunities for the organization then defend your process with tooth and claw. Making exceptions from this rule is a slippery slope leading to becoming a feature factory.

How to Build Offline Boards

Offline boards lift a team's level of collaboration significantly. They are great information radiators for stakeholders, and they massively benefit from the psychology of getting haptic. Learn the best practices of getting started with your own offline boards in this third post of our series on how to kick-off an agile transition.



Note: see the latest version of the board below.

The Benefits of Offline Boards

"When you put problem in a computer, box hide answer. Problem must be visible!"

(Hideshi Yokoi, former president of the Toyota Production System Support Center in Erlanger, Kentucky, USA)

Mr Yokoi's insight points directly at the heart of the problem: Tools like Jira look tempting to handle the administrative side of agile projects. However, utilizing these tools quickly turns into a slippery slope as they inherently favor the creation of "busy work" over valuable work on the product.

These tools tend to reduce transparency for the sake of administrating issues and tasks. Some team members thus may be less motivated to accept their part of the team's accountability to get things done—bookkeeping does not ship valuable product increments.

The solution is simple: Have an offline – or physical – board at the heart of your team's project, and don't reply solely on software solutions. The positive (psychological) effects of writing or sketching on a piece of paper, touching, and moving it cannot be overestimated.

Components of an Offline Board

Offline boards vary from team to team, effected by the culture of the organization, the maturity of the product, team size, policies, and various other factors. There are, however, some basic components that are beneficial for all teams:

- Display all policies affecting the work, e.g. both the "Definition of Ready" and "Definition of Done", or the working agreement.
- Have a schedule with all ceremonies: When is the team gathering for stand-ups, product backlog refinements and sprint planning sessions, sprint reviews, and the team retrospective?
- Provide all information on the sprint, including the sprint goal.
- Organize the sprint backlog: All issues in the backlog are prioritized. (Choose your style: From top to bottom, or from left to right—as long as it is systematic, and works for the team, you cannot be wrong.)
- Create issue lanes: Only one issue per lane. (**Tip**: Limit the space of the lanes to the size of your index cards or Post-it stickies.)
- Set up all workflow stages depending on the team's workflow.
- Consider adding an express lane for issues that require immediate attention of the team without having been picked for the current sprint. (**Tip**: Don't set up an express lane without a strict code defining access to it. Otherwise stakeholders will be tempted to push for all issues becoming "express lane eligible".)
- List the do-not-disturb hours.
- Show an availability chart: Who is available during the current sprint?
- Finally, add some board art: Why not start with the team logo?

Office Supplies for Offline boards

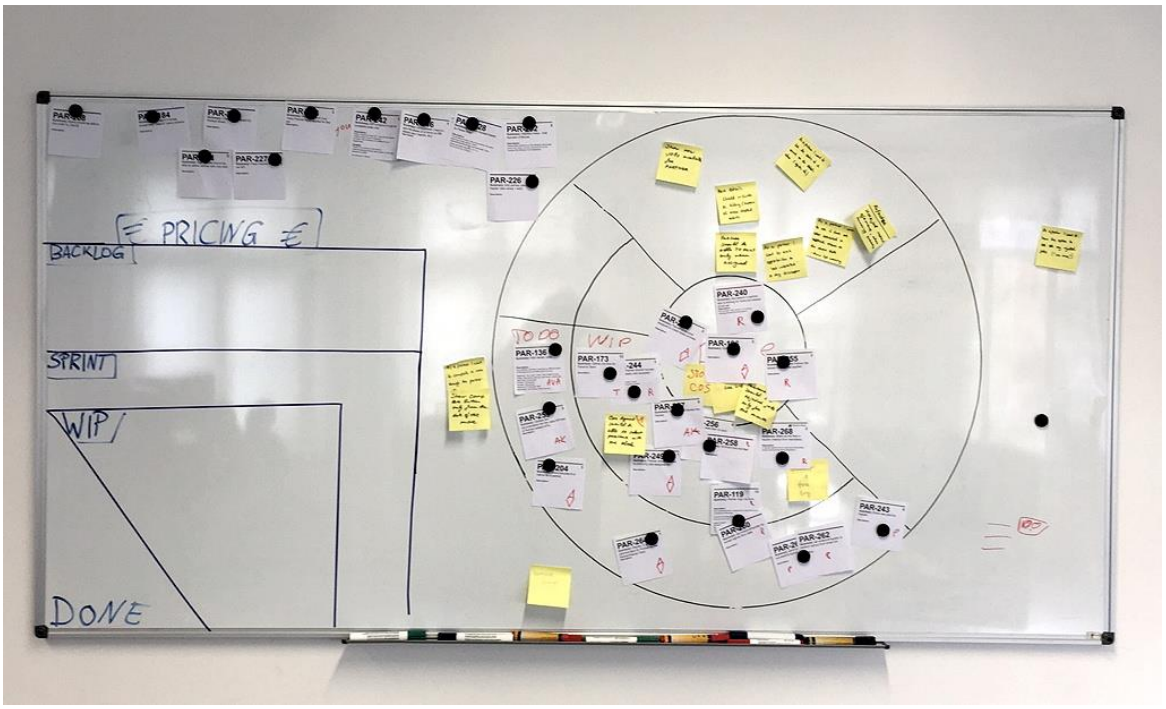
The basic office supplies (and other materials) that you will need besides a whiteboard and a variety of markers are:

- Have magnetic avatars of all team members made: They are both fun and highly effective to visualize who is working on what issues at any given moment. (In Germany, I source the magnetic avatars from magpics; they are less than a one Euro per piece.)
- You will need tape, 3mm wide. (You don't want to separate lanes on a whiteboard with normal markers, as they need to be redrawn all the time, and tend to soil clothes.)
- Use Post-its in various colors to support color coding. (**Tip**: Spend extra on the "Super sticky" version, as normal Post-its will fall off boards too quickly.)
- Alternatively, use colored index cards.
- Buy small magnets as even super sticky Post-its fall off when it's hot or humid, for example.
- Provide a sharpie to dot tasks or issues that are taking longer than a working day.
- Consider sourcing colored paper to print issues or tasks, and color code them if no color printer is available.

Best practices

The following practices have proven to be successful over the years. If your team is new to offline boards, don't start with all of them at once that would a) be overwhelming, and b) limit the team's interest in experimenting with the board design to figure out what is working for them. Remember: It is not your board, but the team's:

1. Building the board is a team exercise, not the Scrum Master's obligation: Setting the board up in the first place, as well as maintaining it over the sprints. (And each has team will have its own taste.)

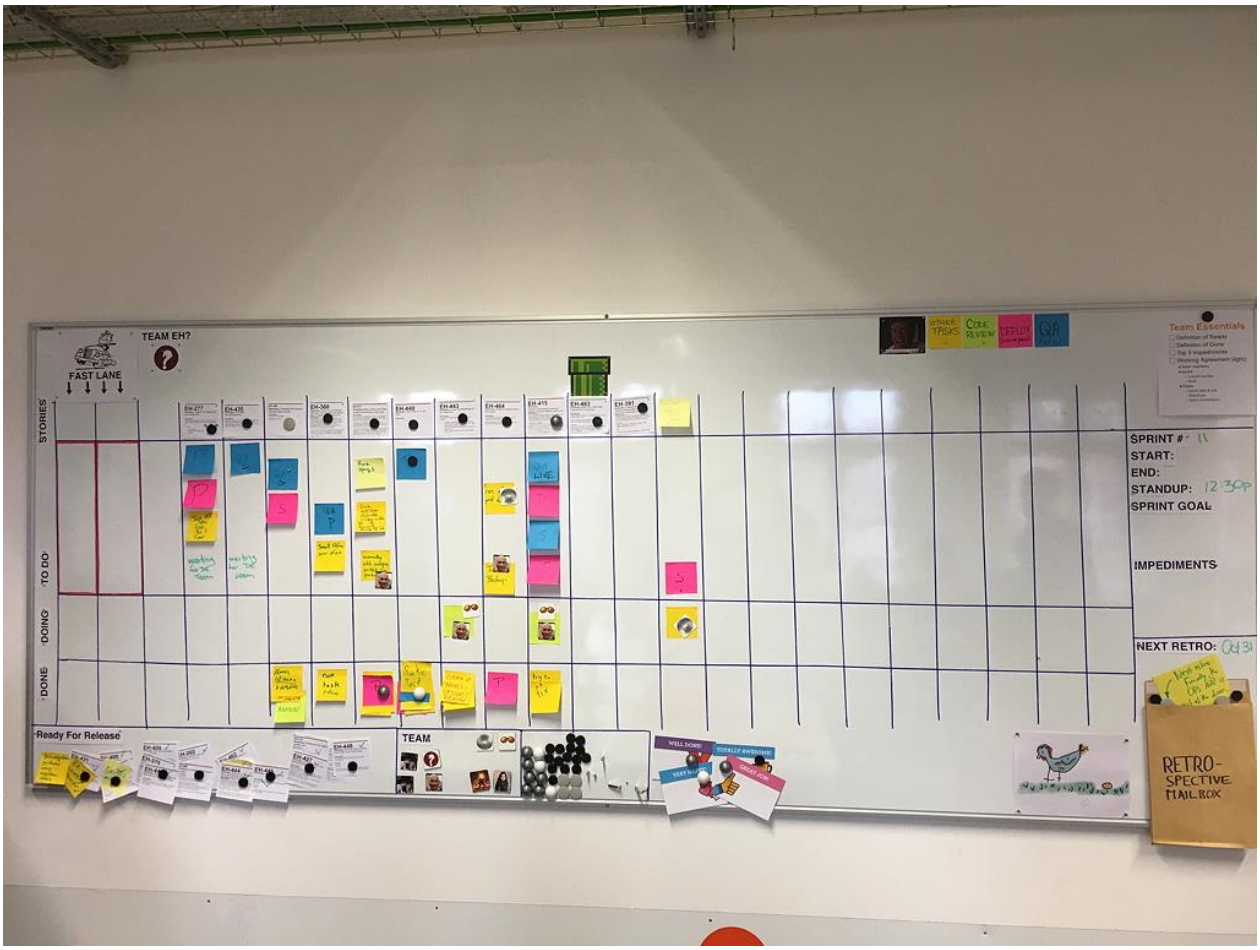


2. Color-coding the issues massively improves the user experience of the board: User stories, bugs, technical tasks, spikes. Don't forget to provide a legend that explains the color code to the observer.
3. Have your standups in front of the board.
4. If you are using in JIRA®, invest in Agile Cards.
5. Use tasks to break down issues into smaller work items. There are several reasons for doing that:
   - Tasks easily visualize progress, if they are moved individually through the stages until the issue itself is "done".
   - Create small tasks: Each task shouldn't take longer than a day. If that span is exceeded, it might indicate an impediment or the issue itself is not as well understood as previously thought.
   - If the team members initial each task, it becomes immediately transparent who is working on what. (Alternatively, you also can use magnetic avatars to achieve this as well.)
   - Some teams like to create tasks utilizing the online ticket system, e.g. Jira. Other teams create handwritten stickies during sprint planning II. Both approaches are working well.
6. Move tasks through the workflow stages during the standups, unless your team's tasks are so small that each team member accomplishes several of those during a day. In this case, updating the board independently of the standups will better visualize the state of the sprint at any given moment.

7. Mark all tasks that are taking more than one day to accomplish by adding a new dot every day to identify potential blockers, process or organizational problems. (**Note**: This practice is not meant to track a team member's productivity.)
8. Limit work in progress: To get things done, it does not help starting working on a new issue when the one you were previously working on is blocked. Hence, limit the number of tasks the team can work on in any workflow stage:
   - You could simply display the "WIP" limit above each category.
   - On top of that, you can use a grid structure: "Enforcing" the work in progress limit by limiting the available space for stickies.
   - Alternatively, if your team is using magnetic avatars, you can just limit the number of avatars available per team member instead, e.g. to three or four.
9. By the way, offline boards work for distributed teams, too. Just use a camera during the standups to update the remote team members, and move their issues on their behalf.
10. Last, but not least: Experiment. Boards—beloved by their teams—usually usually go through several iterations before the team settles on a design.

Update 2016-10-23

An ugly duckling turned swan: the final version of the offline board from above. The first draft proved to be a the good design for the team. We are still waiting for colored magnetic sheets to replace the stickies, though.



I particularly like the small board game the team integrated. It's fun to play with the little neodym magnets…

Conclusion

Offline boards improve a team's understanding of the situation of a sprint at any given time, and thus lift the collaboration among team members to a higher level. They are great information radiators for stakeholders, and benefit from the psychology of getting haptic.

As a Scrum Master, you should remember, though: It is not your board, but the team's. Experiment!

Frameworks & Agile Processes

Agile Metrics—The Good, the Bad, and the Ugly

Suitable agile metrics reflect either a team's progress in becoming agile or your organization's progress in becoming a learning organization.

At the team level, qualitative agile metrics typically work better than quantitative metrics. At the organizational level, this is reversed: quantitative agile metrics provide better insights than qualitative ones.

Good Agile Metrics

Generally speaking, metrics are used to understand the current situation better as well as to gain insight on change over time. Without metrics, assessing any effort or development will be open to gut feeling and bias based interpretation.

A metric should, therefore, be a leading indicator for a pattern change, providing an opportunity to analyze the cause in time.The following three general rules for agile metrics have proven to be useful:

1. The first rule of tracking meaningful metrics is only to track those that apply to the team. Ignore those that measure the individual.
2. The second rule of tracking metrics is not to measure parameters just because they are easy to follow. This practices often is a consequence of using various agile tools that offer out-of-the-box reports.
3. The third rule of tracking metrics is to record context as well. Data without context, for example, the number of the available team member, or the intensity of incidents during a sprint, maybe turn out to be nothing more than noise.

For example, if the (average) sentiment on the technical debt metric (see below) is slowly but steadily decreasing, it may indicate that the team:

- May have started sacrificing code quality to meet deadlines or
- May have deliberately built some temporary solution to speed up experimentation.

While the latter probably is a good thing, the first interpretation is worrying. (You would need to analyze this with the team during a retrospective.)

Good Qualitative Agile Metrics: Self-Assessment Tests

If you like to track a team's progress in adopting agile techniques and processes, self-assessment tests are well-suited for that purpose. For example, I like to use the Scrum Checklist by Henrik Kniberg.

All you have to do, is to run the questionnaire every four to six weeks during a retrospective, record the results, and aggregate them:

Team Metrics

Checklist Items	28.06.16	09.08.16
Core		
Clearly defined PO		
Team has a sprint backlog		
Daily Scrum happens		
Demo happens after every sprint		
Definition of Done available		
Retrospective happens after every sprint		
PO has a product backlog (PBL)		
Have sprint planning meetings		
Timeboxed iterations		
Team members sit together		
Recommended		
Team has all skills to bring backlog item to Done		
Team members not locked into specific roles		
Iterations doomed to fail are terminated early		
PO has product vision that is in synch with PBL		
PBL and product vision is highly visible		
Everyone on the team is participating in estimating		
Estimate relative size (points) rather than time		
PO is available when team is estimating		
Whole team knows the top 3 impediments		
Team has a Scrum master		
PBL items are broken into task within a sprint		
Velocity is measured		
Team has a sprint burndown chart		
Daily Scrum is every day, same time & place		

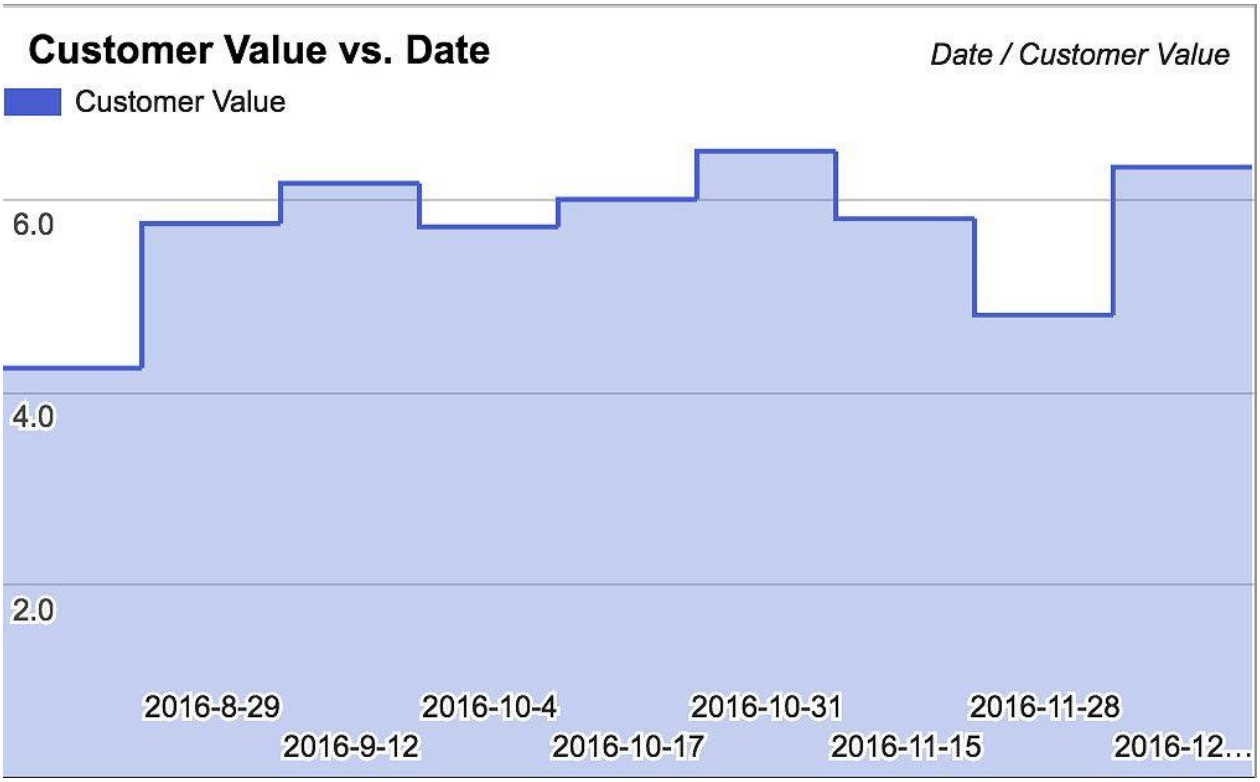
In this example, we were using a kind of estimation poker to answer each question with one of the three values green, orange, and red. The colors were coded as follows:

- *Green*: It worked for the team.
- *Orange*: It worked for the team but there was room for improvement.
- *Red*: It either didn't apply, for example the team wasn't using burn down charts, or the practice was still failing.

If the resulting Scrum practices map is getting greener over time, the team is on the right track. Otherwise, you have to dig deeper to understand the reasons why there is no continuous improvement, and adapt accordingly.

In addition to this exercise, I also like to run an **anonymous poll** at the end of every 2-week-sprint. The poll is comprising of three questions that are each answered on a scale from 1 to 10:

1. *What value did the team deliver last sprint?* (1: we didn't deliver any value, 10: we delivered the maximum value possible.)
2. *How has the level of technical debt developed during the last sprint?* (1: rewrite the application from scratch, 10: there is no technical debt.)
3. *Are you happy working with your teammates?* (1: I am looking for a new job, 10: I cannot wait to get back to the office on Monday mornings.)



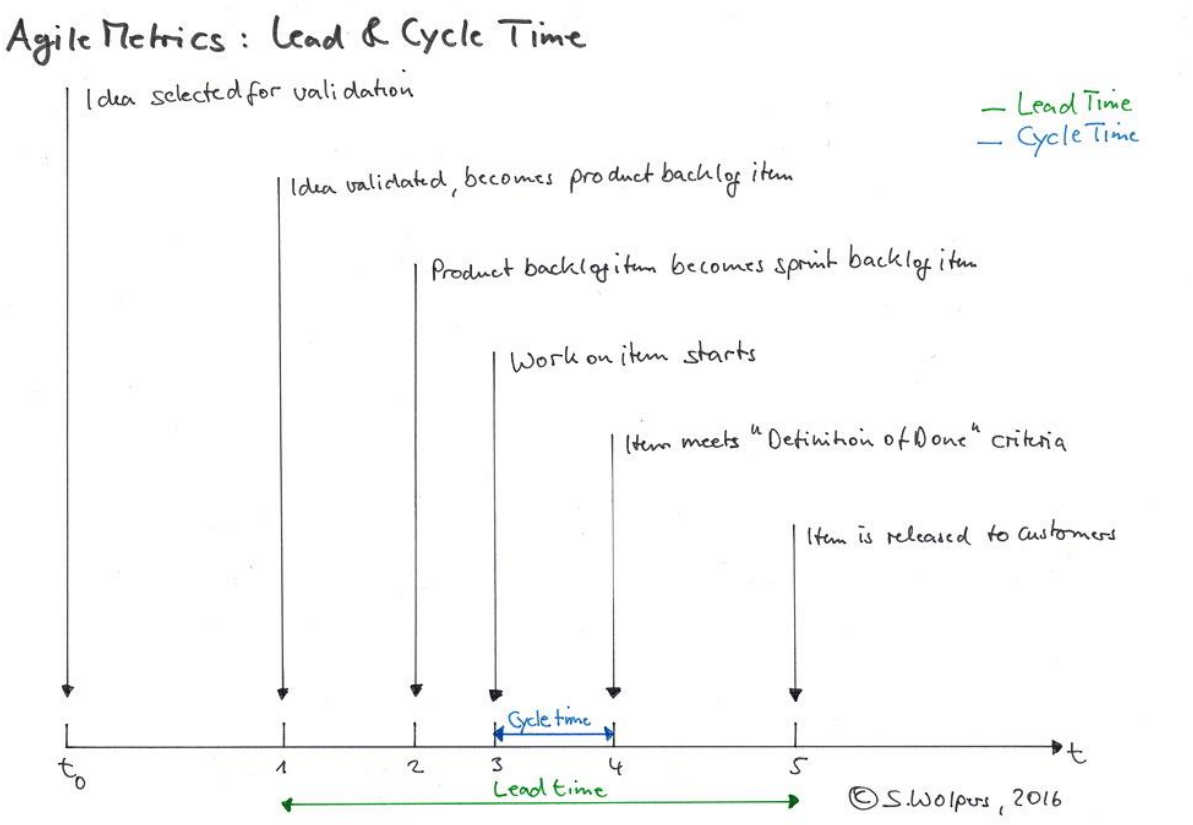
The poll takes less than 30 seconds of each team member's time, and the results are of course available to everyone. Again, tracking the development of the three qualitative metrics provides insight into trends that otherwise might go unnoticed.

Good Quantitative Agile Metrics: Lead Time and Cycle Time

Ultimately, the purpose of any agile transition is to become a learning organization, thus gaining a competitive advantage over the competition. The following metrics apply to the (software) product delivery process but can be adapted to various other processes accordingly.

In the long run, this will not only require to restructure the organization from functional silos to more or less cross-functional teams, where applicable. It will also require analyzing the system itself, for example, to figure out where queues impede value creation.
To identify the existing queues in the product delivery process, you start recording five dates:

1. The date when a previously validated idea, for example, a user story for a new feature, becomes a product backlog item.
2. The date when this product backlog item becomes a sprint backlog item.
3. The date when development starts on this sprint backlog item.
4. The date when the sprint backlog item meets the team's 'Definition of Done'.
5. The date when the sprint backlog item is released to customers.



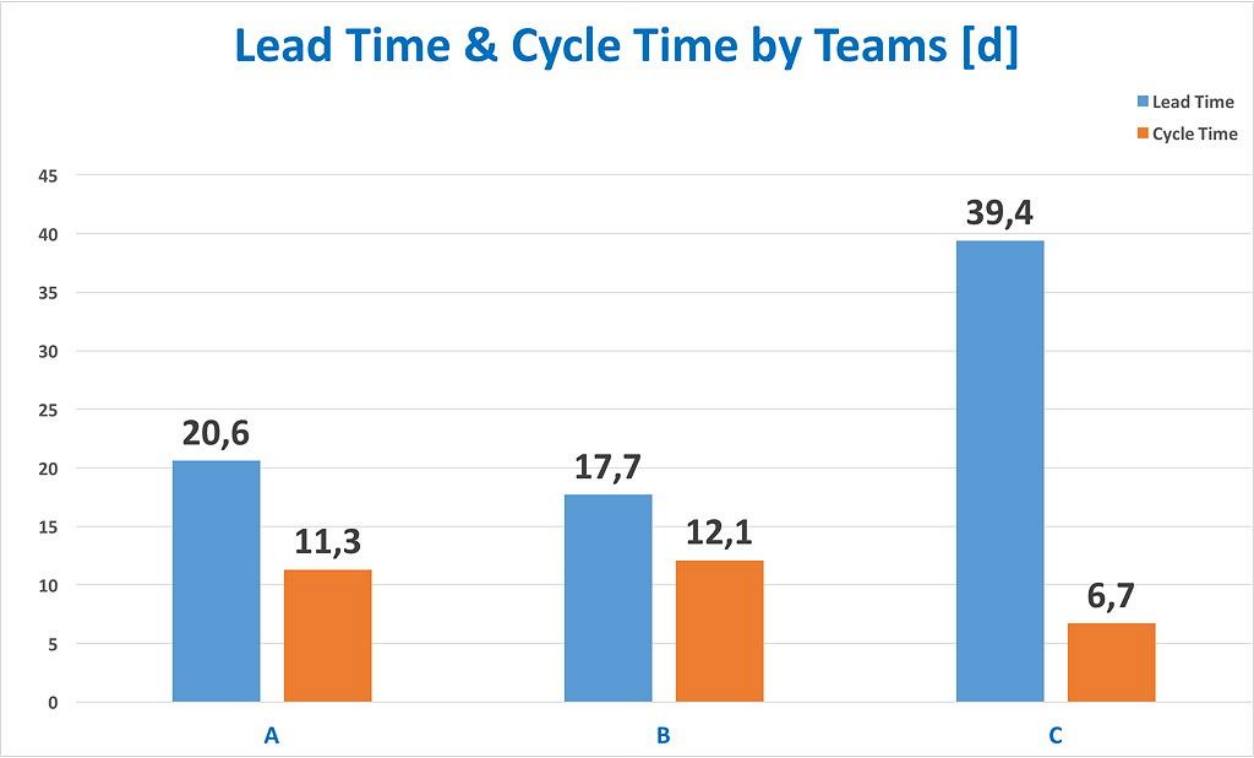
The **lead time** is the time elapsed between first and the fifth date, the **cycle time** the time elapsed between third and the fourth date.

The objective is to reduce both lead time and cycle time to improve the organization's capability to deliver value to customers. The purpose is accomplished by eliminating dependencies and hand-overs between teams within the product delivery process. Helpful practices in this respect are:

- Creating cross-functional and co-located teams
- Having feature teams instead of component teams
- Furthering a whole-product perspective, and systems thinking among all team members.

Measuring lead time and cycle time does not require a fancy agile tool or business intelligence software. A simple spreadsheet will do if all teams stick to a simple rule: note the date once you move a ticket. The method even works with index cards.

The following graphic compares median values of lead time and cycle time of three Scrum teams:



The values were derived from analyzing tickets—both user stories as well as bug tickets—from a period of three months. (It is planned to change that interval to two weeks in 2017.)

Other Good Agile Metrics

Esther Derby suggests in her article Metrics for Agile to also measure the ratio of fixing work to feature work, and the number of defects escaping to production.

Bad Agile Metrics

A bad, yet traditional agile metric is team velocity. Team velocity is a notoriously volatile metric, and hence actually only usable by the team itself.

Some of the many factors that make even intra-team sprint comparisons so difficult are:

- The team onboards new members,
- Veteran team members are leaving,
- Seniority levels of team members change,
- The team is working in unchartered territory,
- The team is working on legacy code,
- The team is running into unexpected technical debt,
- Holiday & sick leave reduce capacity during the sprint,
- The team had to deal with serious bugs.

Actually, you would need to normalize a team's performance each sprint to derive a value of at least some comparable value. (Which usually is not done.)

Additionally, velocity is a metric that can be easily manipulated. I usually include an exercise in how to cook the "agile books" when coaching new teams. And I have never worked with a team that did not manage to come up with suitable ideas how to make to sure that it would meet any reporting requirements based on its velocity. You should not be surprised by this—it is referred to as the Hawthorne effect:

The Hawthorne effect (also referred to as the observer effect) is a type of reactivity in which individuals modify or improve an aspect of their behavior in response to their awareness of being observed.

To make things worse, you cannot compare velocities between different teams since all of them are estimating differently. This practice is acceptable, of course, as estimates are usually not pursued merely for reporting purposes. Estimates are a no more than a side-effect of the attempt to create a shared understanding among team members on the why, how, and what of a user story.

So, don't use velocity as an agile metric. (Read more on velocity in the next chapter.)

Ugly Agile Metrics

The ugliest agile metric I have encountered so far is 'story points per developer per time interval'. This metric equals 'lines of code' or 'hours spent' from a traditional project reporting approach. The metric is completely useless, as it doesn't provide any context for interpretation or comparison.

Equally useless "agile metrics" are, for example, the number of certified team members, or the number of team members that accomplished workshops on agile practices.

Conclusion

If you can only record a few data points, go with start and end dates to measure lead time and cycle time. If you have just started your agile journey, you may consider also tracking the adoption rate of an individual team by measuring qualitative signals, for example, based on self-assessment tests like the 'Scrum test' by Henrik Kniberg.

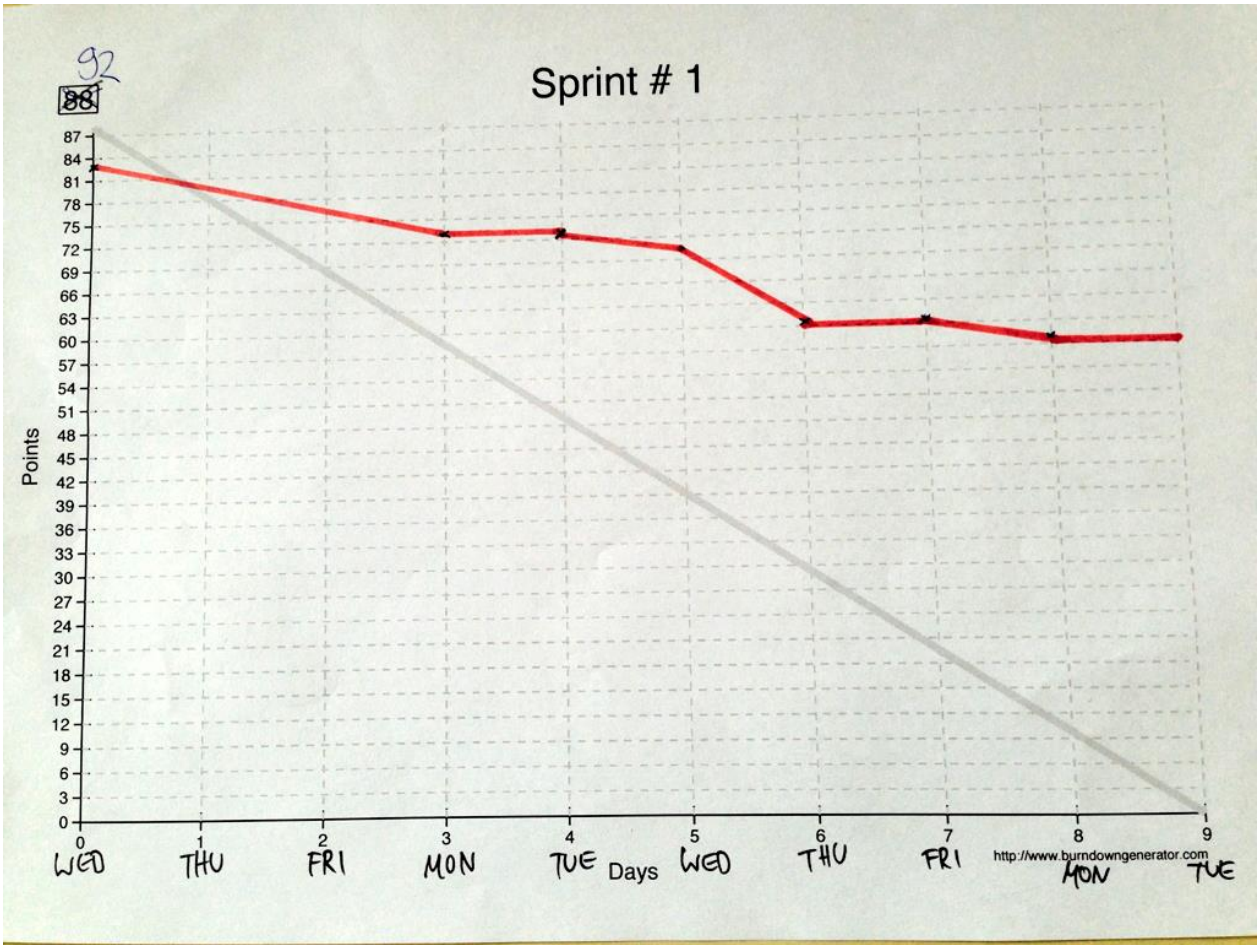
Scrum: The Obsession with Commitment Matching Velocity

The Fine Line Between Risk Mitigation and Falling Back into Covering Your Butt

"The team hasn't met its commitments once. Not once."

The atmosphere was becoming thicker by the minute. The management was displeased with the progress of the project and was looking for answers, starring at a bunch of Jira charts, I prepared earlier. "How can we claim that we are working in Scrum mode, if the team is not sticking with the rules?"

Throughout the majority of projects I have been working on I could observe an obsession with burn-down charts and other Scrum metrics, mainly team commitments. And as a consequence, a side product of backlog grooming, estimation and sprint planning is elevated to the most important management indicator that "Agile" works: The team's commitment is matching or outperforming its average velocity.



While I do see its value when it comes to risk mitigating—when can X probably be delivered to the customers—, the emphasis on its reporting value escapes me. It's all playing safe again, corporate style, when it should be about delivering great software that is valuable, usable and feasible.

Why A Team's Velocity Is Volatile Per Se

There are so many factors that make any team's velocity volatile per se:

1. New team members being onboard,
2. Other team members leaving,
3. Seniority levels,
4. Working in unchartered territory,
5. Working on legacy code, probably undocumented,
6. Running into unexpected technical debt,
7. Holiday & sick leave,
8. Executive intervention,

— you get the idea. Actually, you would need to normalize a team's performance each sprint to derive a value of at least some comparable value. (Which usually is not done.)

The Primary Purpose of Estimation and Grooming

And then there is grooming and estimation. Many of by-the-books Scrum practitioners tend to locate the value of these meetings in delivering an estimate on each user story. To my experience, that's just a side effect of creating a shared understanding among the team members why they will be building a certain feature. (If you like to brush up your understanding of the Agile Manifesto, now would be a good time.)

In other words: A side figure is correlated to a volatile-by-nature figure of a minor inherent value to determine whether the team's performance is on track and Agile is working.

Cooking the Agile Books

If that was case, the rational response of a team to this metrics driven "management style" would be to regularly under-commit and over-sell the effort at the same time. Which would leave sufficient buffer to handle the unexpected. It means de facto cooking the (agile) books to provide the middle management with a (perceived) feeling of being in control.

Does it sound waterfall-ish to you? It absolutely is, just with some Scrum sugar coating in top of it. (As I mentioned earlier, stripping Scrum off some unnecessary features—e.g. turning the product owner into a project manager—creates a viable Waterfall 2.0 framework.)

From my perspective, the obsession with “commitment matching velocity” is one of many reasons that lead to the “Peak Scrum” effect and contradicts a lot of what Agile stands for. It’s like introducing socialism through the backdoor: All plans are fulfilled and yet (probably) little or no value is created in the process.

Faux Metrics and a Way Out of this Dead-End

Instead of focusing on faux metrics, learn to appreciate real KPI that indicate that value is being created for customers and the company. And there are plenty of those available, just look for something supporting the successful delivery of valuable, feasible and usable software.

What is your take on “commitment matching velocity”—is it that the predominant metric in your organization? Please share with us in the comments.

Scrum Master Anti Patterns: Beware of Becoming a Scrum Mom (or Scrum Pop)

Disclaimer: Of course, this post is in no way intended to be gender-specific. In my experience, there is no difference between the Scrum pop and the Scrum mom. This post is all about the emerging trend of Scrum helicopter parenting.

Beware of the Scrum Mom

Trying to be supportive and do good, is most of the time a honorable thing. This is particularly true in your capacity as a Scrum Master. However, doing too much good can quickly have the opposite effect. It's a known Scrum anti pattern, often referred to as the Scrum mom syndrome.

Read on to learn more about its manifestations, and the damage to your team caused by being overly protective.

Scrum Master Anti Patterns

By common understanding, Scrum's host of ceremonies—the Scrum Master—is a leadership role. Wielding no real authority, she can inspire, coach, mentor, and lead by example to further her team's progress to become fully self-organized.

However, if you observe how this role is lived in practice, you can often identify two anti patterns: The Scrum manager and the Scrum mom. While the Scrum manager is easily identifiable by the teacher-like, patronizing attitude, the Scrum mom is tricky to spot. What is (still) beneficial for the team, and what is just overly protective?

Scrum Mom's Characteristics

The Scrum mom is generally shielding the team from the cold and cruel world, creating a safe & happy agile bubble. Some manifestations are:

1. Scrum mom deals with all impediments personally, although practically any other team member could act, too.
2. Scrum mom filters feedback from stakeholders, particularly any negative feedback. Often, she does so by not merely restricting access to the team, but basically shutting it off.
3. Scrum mom is pampering the team, for example by running errands, or being the team secretary, sometimes bordering on the helper syndrome.
4. Scrum mom is also preventing the team from failure whenever possible. This even applies, if failing would be easily fixable and wouldn't be really damaging. (Remember: If you're not failing, you're not pushing hard enough…)
5. Scrum mom is not really challenging the team. She seems to be content, once a certain level of proficiency is achieved.
6. Scrum mom maybe setting boundaries, but is rarely enforcing them. She tends to tolerate damaging behavior from team member in the (futile) hope, the culprit will be insightful and improve over time.
7. Scrum mom likes all her team members, but there will be often a favorite among.
8. Scrum mom's motto: "Right or wrong, my team!"

If the topic resonates with you, you may be interested in the following links, too:

- **Ben Linders**: Why Scrum Masters shouldn't be the one solving all impediments
- **Mia Horrigan**: Confessions of a Scrum Mum

Conclusion

A Scrum Master's good intentions can become an impediment for the Scrum team's progress. This is particularly true in the case of the Scrum mom, when her shielding of the team prevents its members from learning by failing.

In the end, failure is not the opposite of success, but a necessary stepping stone towards success. Like parents, I believe, Scrum Masters need to let go. They need to challenge and support their teams when those are venturing out into the big world on their own. How is a Scrum team otherwise supposed to be become self-organized?

Cargo Cult Agile: The 'State of Agile' Checklist for Your Organization

You want to know the state of agility in your organization? Here we go: Download the checklist, distribute it generously among your colleagues and run a quick poll. It will only take 5 minutes of their time–and then run an analysis on their feedback. If the average number of checkboxes marked is higher than nine, then you are probably practicing cargo cult agile. Consider changing it. Or abandon your agile experiment all together. But don't refer to it as "agile" any longer.

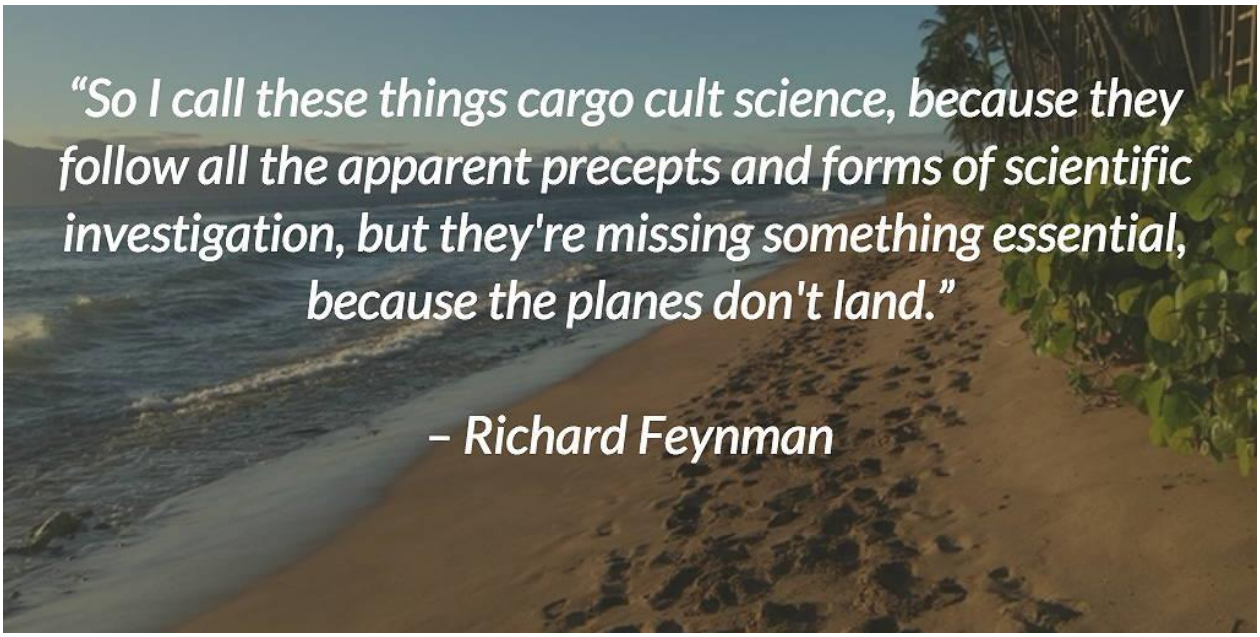
Everyday Failures in Applying Agile

Agile methodologies, like Scrum, have been on the rise across organizations of all kind and sizes for some years by now. Many consultants responded to the increasing demand for agile practitioners, particularly from corporate organizations, with rebranding themselves.

I do not mind professionals pursuing new career opportunities. But pretending to be able to practice "agile" by taking a shortcut, for example reading some books, has increasingly resulted in collateral damage to the agile community. And the damage is tainting the reputation of a great way to build software. A significant part of the developer community meanwhile seems to despise Scrum, for example.

In my experience, "agile" — e.g. Scrum — isn't learned from books or from attending workshops, but in the trenches when confronted with real problems and the urge to ship product. Hence, it is not too difficult to figure out, when the introduction of an agile methodology or framework to an organization didn't work out as planned.

If some of the following observations are common in your organization, you might be experiencing a phenomenon often referred to as "cargo cult".



A cargo cult generally describes a movement that applies a set of rules to the letter without understanding for what reason they should be practiced.

One of the well-know examples with regard to technology was described by Richard Feynman:

"In the South Seas there is a cargo cult of people. During the war they saw airplanes with lots of good materials, and they want the same thing to happen now. So they've arranged to make things like runways, to put fires along the sides of the runways, to make a wooden hut for a man to sit in, with two wooden pieces on his head for headphones and bars of bamboo sticking out like antennas—he's the controller—and they wait for the airplanes to land. They're doing everything right. The form is perfect. It looks exactly the way it looked before. But it doesn't work. No airplanes land. So I call these things cargo cult science, because they follow all the apparent precepts and forms of scientific investigation, but they're missing something essential, because the planes don't land."

Source: "Surely You're Joking, Mr. Feynman!", page 340.

It turns out, that Scrum seems to be particularly well suited for cargo cult agile. So, best practices become rules overnight. And rules need to be enforced. And if the people don't live up to them, if metrics are not met, then you need–of course–more structure, more rules "to fix agile".

What Is the Purpose of the Poll?

The purpose of the whole exercise is to start a conversation about what part of your agile transition is going well and where action needs to be taken. The poll supports this approach in several ways:

1. It is anonymous—no one will hold back, contrary to retrospectives as a competing format to gather feedback.
2. The poll delivers data, it's no longer a gut feeling, and hence a good basis to start talking to the management.
3. It is very affordable.

Download the Cargo Cult Agile Checklist as a PDF

You can download the following list of 25 issues as a PDF, print and use them for yourself.

If you do so, I would appreciate your feedback on how this worked out for you and what manifestations you would add to the list.

The Cargo Cult Agile Checklist

Now let's have a look at some typical manifestations of cargo cult agile within an organization. The checklist assumes that you are using Scrum, but can be applied generally to other agile practices accordingly. (A final note: The list may be become less applicable with an increase in size of the organization in question.):

1. (Product) vision and strategy are not communicated
2. Roadmaps with fixed release dates are provided for a year ahead by the CTO
3. No one from the organization is talking to customers
4. CTO and stakeholders insist on every change to be approved by them in advance
5. Offline boards are banned for confidentiality reasons
6. Product owners are bypassed by stakeholders talking directly to the CTO
7. Stakeholders decide on shipping product increments, not the product owner
8. Projects are shipped only when completed, but not incrementally
9. Stakeholders are prevented from talking to the Scrum teams
10. The product backlog is defined by a product council
11. Features of doubtful value are being pushed through, e.g. to secure bonuses
12. Sales is promising non-existent features to close deals w/o including the product owner
13. Deadlines or fixed schedules are still in use for noncritical issues
14. Product management isn't granted access to business intelligence to make informed, data-driven decisions
15. Stakeholders communicate to product and engineering in the form of requirement documents

16. Product owners spend time mostly on the creation and administration of user stories
17. Sprint backlogs are changed on short notice after the sprint started
18. There is a dedicated Scrum team even for bugs and minor change requests
19. Scrum ceremonies are never attended by any stakeholders
20. Velocity matching commitment is the main metric to measure the success of Scrum
21. Developers are not participating in user story creation
22. Scrum teams are changing in size and composition, depending on the number of simultaneously running side projects and task forces
23. During stand-ups, the team members are reporting to the Scrum Master
24. Retrospectives are held regularly, but no changes follow
25. Scrum teams are not cross-functional and therefore depending on other teams or departments.

The Cargo Cult Agile Litmus Test For Your Organization

Now, here is a fun game for everyone involved in agile processes in your organization to check the health status:

Print out this post–or download the printable PDF version instead–, and distribute it generously among your colleagues. Ask them to go through it and check all boxes that apply to your organization–it will only take 5 minutes of their time. Then run an analysis on their feedback and assess your situation:

- **0 to 2 boxes**: I would like to talk to you how you managed to do that. Care for a Skype-call? Or would you like to contribute a guest-post to my blog?
- **to 5 boxes**: Well done! You're on a good way.
- **6 to 8 boxes**: There is room for improvement. Lot's of it.
- **9 to 14 boxes**: If you haven't very recently embarked on your agile voyage, then it is time to change your approach.
- **15 to 20 boxes**: Okay, start over with agile–it is not working in the current set-up within your organization.
- **21 to 25 boxes**: You either haven't started going agile yet. Or you are sugar-coating command-and-control structures to look "agile". It won't work, by the way.

Conclusion

There is no "agile by the book" that automatically works well within your organization, if you only stick to the letters. You will have to identify your version of "agile" by yourself. Start doing so by testing things that have been successfully used by other organizations in the past.

If those work for you, too, great–stick with them. Otherwise move on. It is fine to adapt or even drop standard agile rituals in the process, if it helps figuring out, what is working within your organization. And don't hesitate to change or adapt any best practices as you see fit to make them work within your organization.

17 Stand-up Anti-Patterns

The Stand-up

The daily stand-up is the ceremony with the highest anti-pattern density among all scrum ceremonies. Learn more about the stand-up anti-patterns that threaten to derail your agile transition.

Stand-up Anti-Patterns – From Dysfunctional Scrum Teams to Organizational Failures

Typically, a good scrum team needs about five to ten minutes for a stand-up. Given this short period, it is interesting to observe that the daily stand-up is the scrum ceremony with the highest potential anti-pattern density. The anti-patterns range from behaviors driven by dysfunctional teams to apparent failures at an organizational level.

My favorite stand-up anti-patterns are as follows:

1. **No routine:** The stand-up does not happen at the same time and the same place every day. (While routine has the potential to ruin every retrospective, it is helpful in the context of stand-ups. Think of it like a spontaneous drill: don't put too much thought into the stand-up, just do it. Skipping stand-ups can turn out to be a slippery slope. And skipping may only be acceptable the day after the sprint planning. However, please keep in mind that every team member can veto skipping the stand-up.))

2. **Status report:** The stand-up is a status report meeting, and team members are waiting in line to "report" progress to the scrum master, the product owner, or maybe even a stakeholder.

3. **Ticket numbers only:** Updates are generic with little or no value to others. ("Yesterday, I worked on X-123. Today, I will work on X-129.")

4. **Problem solving:** Discussions are triggered to solve problems, instead of parking those so they can be addressed after the stand-up.

5. **Planning meeting:** The team hijacks the stand-up to discuss new requirements, to refine user stories, or to have a sort of (sprint) planning meeting.

6. **No red dots:** A team member experiences difficulties in accomplishing an issue over several consecutive days, and nobody is offering help. (This a sign that that people

either do not trust each other, or that the utilization of the team is maximized.)

7. **Monologs:** Team members violate the time-boxing, starting monologues. (60 to 90 seconds per team member should be more than enough time on air.)

8. **Statler and Waldorf:** A few team members are commenting every issue. (Usually, this is not just a waste of time, but also patronizing as well as annoying.)

9. **Disrespect I:** Other team members are talking while someone is sharing his or her progress with the team. (Similarly irritating is the need to use speak tokens among adults to avoid this behavior.)

10. **Assignments:** The product owner – or scrum master – assigns tasks directly to team members.

11. **Cluelessness:** Team members are not prepared for the stand-up. ("I was doing some stuff but I cannot remember what. Was important, though.")

12. **Let's start the shift:** The stand-up acts as a kind of artificial factory siren to start the next shift. (This is a common Taylorism artifact where trust in the team is missing.)

13. **Disrespect II:** Team members are late to the stand-up. (**Note**: if the time for the stand-up was not chosen by the team it otherwise indicates distrust on the management side.)

14. **Excessive feedback:** Team members criticize other team members right away sparking a discussion instead of taking their critique outside the stand-up.

15. **Overcrowded:** Stand-ups are ineffective due to the large number of active participants.

16. **Talkative chickens:** "Chickens" actively participate in the stand-up. (I think it is generally acceptable if stakeholder ask a question during the stand-up. However, they are otherwise supposed to merely listen in.)

17. **Anti-agile:** Line managers are attending stands-up to gather "performance data" on individual team members. (This behavior is defying the very purpose of self-organizing teams.)

Depending on the context, it could also be an anti-pattern if the product owner – or even another stakeholder – is introducing new tickets to the current sprint during the stand-up. This behavior may be acceptable for priority one bugs. (Although the team should be aware of those before the stand-up.) However, it is an unacceptable behavior – and thus an anti-pattern – for changing priorities on the fly in the middle of a sprint.

Lastly, some teams like to have stand-ups in Slack, particularly those that are not co-located. Again, depending on the context, this does not need to manifest an anti-pattern per se. I was even working with a co-located team that used Slack as their preferred way of having a stand-up. It worked.

Conclusion

A lot of agile practitioners tend to consider stand-ups to be a candidate for waste. However, from a scrum master or agile coach perspective stand-ups offer the highest yield of anti-patterns – given the effort is so small by comparison to other ceremonies.

28 Product Backlog and Refinement Anti-Patterns

The Product Backlog

Scrum is a practical framework to build products, provided you identified in advance what to build. But even after a successful product discovery phase, you may struggle to make the right thing in the right way if your product backlog is not up to the job.

The following article points at the most common product backlog anti-patterns – including the product backlog refinement process – that limit your team's success.

The Product Backlog Refinement According to the Scrum Guide

Frist of all, let's have a look at the current issue of the Scrum Guide on the product backlog refinement:

"Product Backlog refinement is the act of adding detail, estimates, and order to items in the Product Backlog. This is an ongoing process in which the Product Owner and the Development Team collaborate on the details of Product Backlog items. During Product Backlog refinement, items are reviewed and revised. The Scrum Team decides how and when refinement is done. Refinement usually consumes no more than 10% of the capacity of the Development Team. However, Product Backlog items can be updated at any time by the Product Owner or at the Product Owner's discretion.

Higher ordered Product Backlog items are usually clearer and more detailed than lower ordered ones. More precise estimates are made based on the greater clarity and increased detail; the lower the order, the less detail. Product Backlog items that will occupy the Development Team for the upcoming Sprint are refined so that any one item can reasonably be "Done" within the Sprint time-box. Product Backlog items that can be "Done" by the Development Team within one Sprint are deemed "Ready" for selection in a Sprint Planning. Product Backlog items usually acquire this degree of transparency through the above-described refining activities.

The Development Team is responsible for all estimates. The Product Owner may influence the Development Team by helping it understand and select trade-offs, but the people who will perform the work make the final estimate."

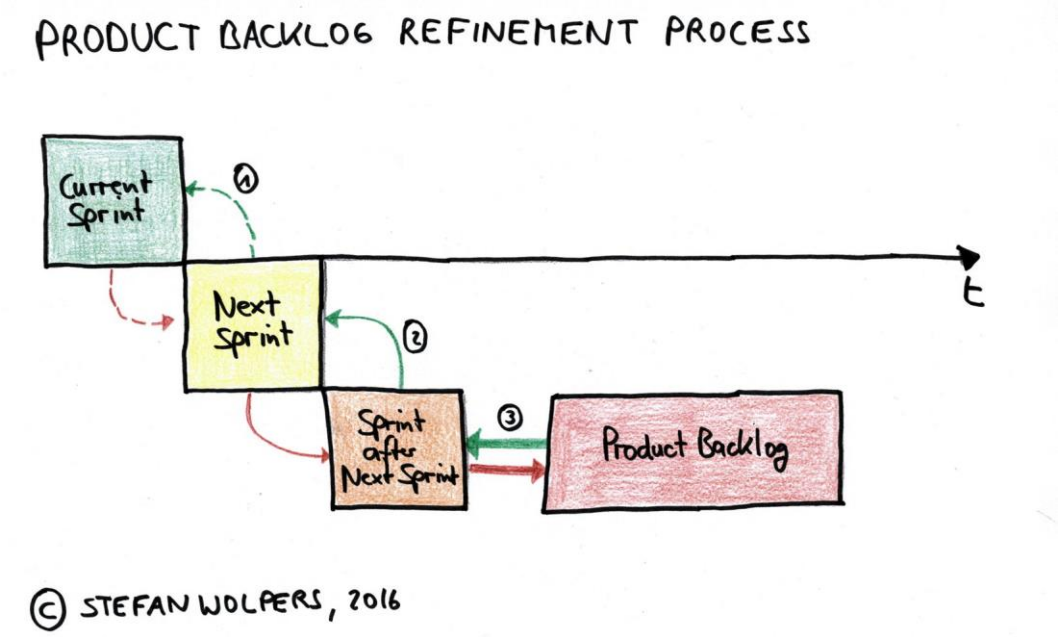
Source & Copyright: *©2016 Scrum.Org and ScrumInc*[1].

A Typical Product Backlog Refinement Process

Based on the Scrum Guide, a typical process looks as follows:

[1] *Offered for license under the Attribution Share-Alike license of Creative Commons, accessible at http://creativecommons.org/licenses/by-sa/4.0/legalcode and also described in summary form at http://creativecommons.org/licenses/by-sa/4.0/.*

1. The product owner prioritizes the backlog in advance, so it reflects the best possible use of the development team's resources:
   a. The product owner creates and pre-populates the two upcoming sprints with user stories, using the team's project management software (or a spreadsheet or any other organizational tool the team applies).
   b. The product owner maintains this pattern continuously.
   c. The product owner also adds new user stories that he or she may have identified since the previous refinement session.



2. The product owner and the development team are jointly working on user stories:
   a. The product owner provides the answer to the 'why' question (business purpose),
   b. The team answers the 'how' question (technical implementation),
   c. And both collaborate on the 'what' question: what scope is necessary to achieve the desired purpose?
3. The whole team agrees to time-box discussions. A typical time-box per user story would be around five minutes on average per cycle.
4. The product owner provides the acceptance criteria to user stories.
5. The development team defines what is required to consider a user story to be ready for becoming a sprint backlog item.

6. The product owner clarifies questions of the team or invites subject matter experts to refinement sessions who can answer the team's questions.
7. Consecutive refinement cycles last until each user story meets the definition of ready, or is no longer pursued.
8. Lastly, the team may estimate the available user stories that meet the "definition of ready" criteria. The product owner can now choose from those user stories to become part of an upcoming sprint.

Common Product Backlog (Refinement) Anti-Patterns

Despite being a relatively straightforward, the process of creating and refining a product backlog often suffers from various anti-patterns. I have identified four different categories:

General Product Backlog Anti-Patterns

- **Prioritization by proxy:** A single stakeholder or a committee of stakeholder prioritizes the product backlog. (The strength of Scrum is building on the strong position of the product owner. The product owner is the only persons to decide what tasks become product backlog items. Hence, the product owner also decides on the priority. Take away that empowerment, and Scrum turns into a pretty robust waterfall 2.0 process.)

- **100% in advance:** The scrum team creates a product backlog covering the complete project or product upfront because the scope of the release is limited. (Question: how can you be sure to know today what to deliver in six months from now?)

- **Over-sized:** The product backlog contains more items than the scrum team can deliver within three to four sprints. (This way the product owner creates waste by hoarding issues that might never materialize.)

- **Outdated issues:** The product backlog contains items that haven't been touched for six to eight weeks or more. (That is typically the length of two to four sprints. If the product owner is hoarding backlog items, the risk emerges that older items become outdated, thus rendering previously invested work of the scrum team obsolete.)

- **Everything is estimated:** All user stories of the product backlog are detailed and estimated. (That is too much upfront work and bears the risk of misallocating the scrum team's time.)

- **Component-based items:** The product backlog items are sliced horizontally based on components instead of vertically based on end-to-end features. (This

may be either caused by your organizational structure. Then move to cross-functional teams to improve the team's ability to deliver. Otherwise, the team – and the product owner – need a workshop on writing user stories.)

- **Missing acceptance criteria:** There are user stories in the product backlog without acceptance criteria. (It is not necessary to have acceptance criteria at the beginning the refinement cycle although they would make the task much easier. In the end, however, all user stories need to meet the definition of ready standard, and acceptance criteria are a part of that definition.)

- **No more than a title:** The product backlog contains user stories that comprise of little more than a title. (See above.)

- **Issues too detailed:** There are user stories with an extensive list of acceptance criteria. (This is the other extreme: the product owner covers each edge case without negotiating with the team. Typically, three to five acceptance criteria are more than sufficient.)

- **Neither themes nor epics:** The product backlog is not structured by themes or epics. (This makes it hard to align individual items with the "big picture" of the organization. The product backlog is not supposed to be an assortment of isolated tasks or a large to-do-list.)

- **No research:** The product backlog contains few to no spikes. (This often correlates with a team that is spending too much time on discussing prospective problems, instead of researching them with a spike as a part of an iterative user story creation process.)

Product Backlog Anti-Patterns at Portfolio and Product Roadmap Level

- **Roadmap?** The product backlog is not reflecting the roadmap. (The product backlog is supposed to be detailed only for the first two or three sprints. Beyond that point, the product backlog should rather focus on themes and epics from the product roadmap. If those are not available, the product backlog is likely to granular.)

- **Annual roadmaps:** The organization's portfolio plan, as well as the release plan or product roadmap, are created once a year in advance. (If the product backlog stays aligned to these plans, it introduces waterfall planning through the backdoor. Agile planning is always "continuous". At the portfolio level, the plan needs to be revised be least every three months.)

- **Roadmaps kept secret:** The portfolio planning and the release plan or product roadmap are not visible to everybody. (If you do not know where you are going any road will get you there. This information is crucial for any scrum team and needs to be available to everybody at any time.)

- **China in your hands:** The portfolio planning and the release plan or the product roadmap are not considered achievable and believable. (If this is reflected in the product backlog, working on user stories will probably be a waste.)

Product Backlog Anti-Patterns of the Product Owner

- **Storage for ideas:** The product owner is using the product backlog as a repository of ideas and requirements. (This practice is clogging the product backlog, may lead to a cognitive overload and makes alignment with the 'big picture' at portfolio management and roadmap planning level very tough.)

- **Part-time PO:** The product owner is not working daily on the product backlog. (The product backlog needs to represent at any given time the best use of the development team's resources. Updating it once a week before the next refinement session does not suffice to meet this requirement.)

- **Copy & paste PO:** The product owner creates user stories by breaking down requirement documents received from stakeholders into smaller chunks. (That scenario helped to coin the nickname "ticket monkey" for the product owner. Remember: user story creation is a team exercise.)

- **Dominant PO:** The product owner creates user stories by providing not just the 'why' but also the 'how', and the 'what'. (The team answers the 'how' question – the technical implementation –, and both the team and the PO collaborate on the 'what' question: what scope is necessary to achieve the desired purpose.)

- **INVEST?** The product owner is not applying the INVEST principle by Bill Wake to user stories.

- **Issues too detailed:** The product owner invests too much time upfront in user stories making them too detailed. (If a user story looks complete, the team members might not see the necessity to get involved in a further refinement. This way a "fat" user story reduces the engagement level of the team, compromising the creation of a shared understanding. By the way, this didn't happen back in the days when we used index cards given their physical limitation.)

- **What team?** The product owner is not involving the entire scrum team in the refinement process and instead is relying on just the "lead engineer" (or any

other member of the team independently of the others).

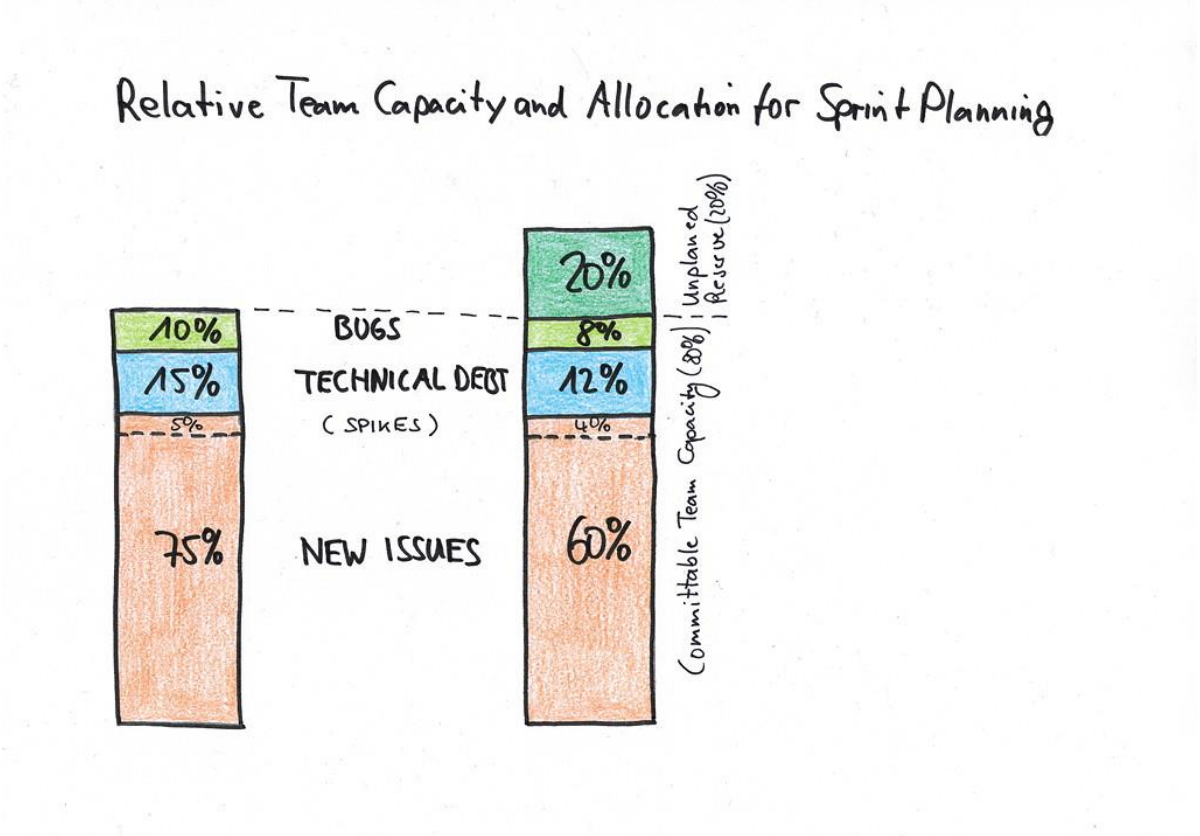
- **'I know it all' PO:** The product owner does not involve stakeholders or subject matter experts in the refinement process. (A product owner who believes to be either omniscient or a communication gateway is a risk to the Scrum team's success.)

Product Backlog Anti-Patterns of the Development Team

- **Submissive team:** The development team submissively follows the demands of the product owner. (Challenging the product owner whether his or her selection of issues is the best use of the development team's time is the noblest obligation of every team member: why shall we do this?)

- **What technical debt?** The development team is not demanding adequate resources to tackle technical debt and bugs. (The rule of thumb is that 25% of resources are allocated every sprint to fixings bugs and refactor the code base.)

- **No slack:** The development team is not demanding 20% slack time from the product owner. (This is overlapping with the sprint planning and the team's commitment. However, it cannot be addressed early enough. If a team's capacity is always utilized at 100 %, its performance will decrease over time. Everyone will focus on getting his or her tasks done. There will be less time to support teammates or to pair. Small issues will no longer be addressed immediately. And ultimately, the 'I am busy' attitude will reduce the generation of a shared understanding among all team members why they do what they are doing.)



Product Backlog Anti-Patterns of the Scrum Team

- **No time for refinement:** The team does not have enough refinement sessions, resulting in a low-quality backlog. (The Scrum Guide advises spending up to 10% of the Scrum team's time on the product backlog refinement. Which is a sound business decision: Nothing is more expensive than a feature that is not delivering any value.)

- **Too much refinement:** The team has too many refinement sessions, resulting in a too detailed backlog. (Too much refinement isn't healthy either.)

- **No DoR:** The scrum team has not created a 'definition of ready' that product backlog items need to match before becoming selectable for a sprint. (A simple checklist like the 'definition of ready' can significantly improve the scrum team's work. It will increase the quality of both the resulting user stories as well as the general way of working as a team.)

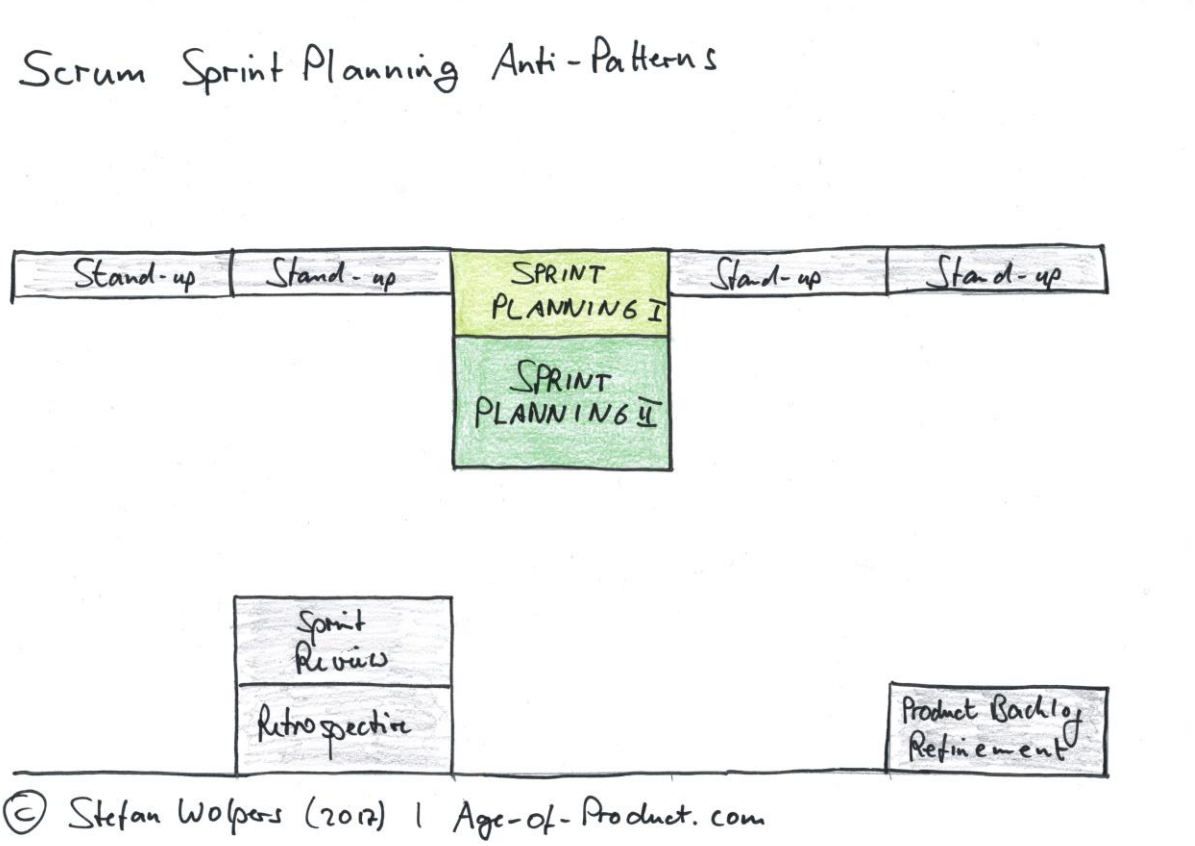
Conclusion:

Even in the case, you have successfully identified what to build next, your product backlog, as well as its refinement process, will likely provide room for improvement. Just take it to the team.

19 Sprint Planning Anti-Patterns

The Sprint Planning

Scrum's sprint planning is a simple ceremony. Invest upfront during the product backlog refinement, and you will keep it productive. Avoiding the following 19 sprint planning anti-patterns will help, too.



The Purpose of the Sprint Planning

The purpose of Scrum's sprint planning is to align the development team and the product owner. Both need to agree on the shippable product increment of the next sprint. The idea is that the development team's commitment reflects the product owner's sprint goal. Also, the team needs to come up with a plan on how to accomplish its commitment. (Or forecast if you prefer that term.)

If the scrum team has been successfully using product backlog refinements in the past the sprint planning part 1 will be short. The development team and the product owner will

adjust the discussed scope of the upcoming sprint to the available capacity. Maybe, someone from development team will not be available next sprint. So, one or two tasks will have to go back to the product backlog.

Or a valuable new task appeared overnight, and the product owner wants this task to become a part of the next sprint backlog. Consequently, some other user story needs to go back to the product backlog. A good team can handle that in five to ten minutes before moving on to sprint planning part 2. During sprint planning II the team breaks down the first set of sprint backlog items into subtasks.

Sprint Planning Anti-Patterns

There are three categories of sprint planning anti-patterns. They concern the development team, the product owner, and the scrum team.

Sprint Planning Anti-Patterns of the Development Team

- **Any absentees?** The team members do not determine their availability at the beginning of the sprint planning. (Good luck with making a commitment in this situation.)

- **Capacity?** The development team overestimates its capacity and takes on too many tasks. (The development team should instead take everything into account that might affect its ability to deliver. The list of those issues is long: public holidays, new team members, and those on vacation leave, team members quitting, team members on sick leave, corporate overhead, scrum ceremonies and other meetings to name a few.)

- **Ignoring technical debt:** The development team is not demanding adequate capacity to tackle technical debt and bugs during the sprint. (The rule of thumb is that 25% of resources are allocated every sprint to fix bugs and refactor the code base. If the product owner ignores this practice, and the development team accepts this violation the scrum team will find itself in a downward spiral. Its future product delivery capability will decrease.)

- **No slack:** The development team is not demanding 20% slack time from the product owner. (If a team's capacity is always over-utilized, its performance will decrease over time. This will particularly happen in an organization with a volatile daily business. As a consequence, everyone will focus on getting his or her tasks done. There will be less time to support teammates or to pair. The team will no longer address smaller or urgent issues promptly. Individual team members will become bottlenecks, which might seriously impede the flow within the team. Lastly, the 'I am busy' attitude will reduce the generation of a shared understanding among all team

members. Overutilization will always push the individual team member to focus on his or her output. On the other side, slack time will allow the scrum team to act collaboratively and focus on the outcome.)

- **Planning too detailed:** During sprint planning II, the development team plans every single subtask of the upcoming sprint in advance. (Don't become too granular. Two-thirds of the sub-tasks are more than sufficient, the rest will follow naturally during the sprint. Doing too much planning upfront might result in waste.)

- **Too much estimating:** The development team estimates sub-tasks. (That looks like accounting for the sake of accounting to me. Don't waste your time on that.)

- **Too little planning:** The development team is skipping the sprint planning II altogether. (Skipping the sprint planning II is unfortunate, as it is also a good situation to talk about how to spread knowledge within the development team. For example, the team should think about who will be pairing with whom on what task. The sprint planning II is also a well-suited to consider how to reduce technical debt.)

- **Team leads?** The development team does not come up with a plan to deliver on its commitment collaboratively. Instead, a 'team lead' assigns tasks to individual team members. (I know that senior developers do not like the idea, but there is no 'team lead' in a scrum team. **Read More**: Why Engineers Despise Agile).

Sprint Planning Anti-Patterns of the Product Owner

- **What are we fighting for?** The product owner cannot provide a sprint goal, or the chosen sprint goal is flawed. (An original sprint goal answers the "What are we fighting for?" question. It is a negotiation between the product owner and the development team. It is focused and measurable, as sprint goal and team commitment go hand in hand. Lastly, the sprint goal is a useful calibration for the upcoming sprint.)

- **Calling Kanban 'Scrum':** The sprint backlog resembles a random assortment of tasks, and no sprint goal is defined. (If this is the natural way of finishing your sprint planning I, you probably have outlived the usefulness of Scrum as a product development framework. Depending on the maturity of your product, Kanban may prove to be a better solution. Otherwise, the randomness may signal a weak product owner who listens too much to stakeholders instead of prioritizing the product backlog appropriately.)

- **Unfinished business:** Unfinished user stories and other tasks from the last sprint spill over into the new sprint without any discussion. (There might be good reasons for that, for example, a task's value has not changed. It should not be an automatism,

though, remember the sunk cost fallacy.)

- **Last minute changes:** The product owner tries to squeeze in some last-minute user stories that do not meet the definition of ready. (Principally, it is the prerogative of the product owner to make such kind of changes to ensure that the development team is working only on the most valuable user stories at any given time. However, if the scrum team is otherwise practicing product backlog refinement sessions regularly, these occurrences should be a rare exception. If those happen frequently, it indicates that the product owner needs help with prioritization and team communication. Or the product owner needs support to say 'no' more often to stakeholders.)

- **Output focus:** The product owner pushes the development team to take on more tasks than it could realistically handle. Probably, the product owner is referring to former team metrics such as velocity to support his or her desire. (This would be an opportunity for the candidate to step in and address the issue.)

Sprint Planning Anti-Patterns of the Scrum Team

- **Irregular sprint lengths:** The scrum team has variable sprint cadences. For example, tasks are not sized to fit into the regular sprint length. Instead, the sprint length is adapted to the size of the tasks. (It is quite common to extend the sprint length at the end of the year when most of the team member are on holiday. However, there is no reason to deviate from the regular cadence during the rest of the year. Instead of changing the sprint length, the scrum team should invest more effort into sizing epics and user stories in the right way.)

- **Over-commitment:** The scrum team regularly takes on way too many tasks and moves unfinished work simply to the next sprint. (If two or three items spill over to the next sprint, so be it. If regularly 30-40 percent of the original commitment is not delivered during the sprint the scrum team may have created a kind of 'time-boxed Kanban.' Maybe, this is the right moment to ask the scrum team whether moving to Kanban might be an alternative.)

- **Stage-gate by DoR:** The definition of ready is handled in a dogmatic way thus creating a stage-gate-like approval process. (That is an interesting topic for a discussion among the team members. For example, should a valuable user story be postponed to another sprint just because the front end designs will not be available for another two working days? My suggestion: take it to the team. If they agree with the circumstances and accept the user story into the sprint — that is fine. **Read More:** The Dangers of a Definition of Ready.)

- **Ignoring the DoR:** The development team is not rejecting user stories that do not meet the definition of ready. (This is the opposite side of being dogmatic about the

application of DoR: not-ready user stories that will cause unnecessary disruptions during the sprint are allowed into it. Laissez-faire does not help either.)

- **Forecast imposed:** The sprint commitment is not a team-based decision. Or it is not free from outside influence. (There are several anti-patterns here. For example, an assertive product owner dominates the development team by defining its scope of the commitment. Or a stakeholder points at the team's previous velocity demanding to take on more user stories. ("We need to fill our free capacity.") Or the 'tech lead' of the development team is making a commitment on behalf of the team. Whatever the reason is, the candidate should address the underlying issues.)

- **Planning ignored:** The development team is not participating collectively in the sprint planning. Instead, two team members, for example, the tech and UX leads, represent the team. (As far as the idea of one or two 'leading' teammates in a scrum team is concerned, there are none, see above. And unless you are using LeSS – no pun intended – where teams are represented in the overall sprint planning, the whole scrum team needs to participate. It is a team effort, and everyone voice hence needs to be heard.)

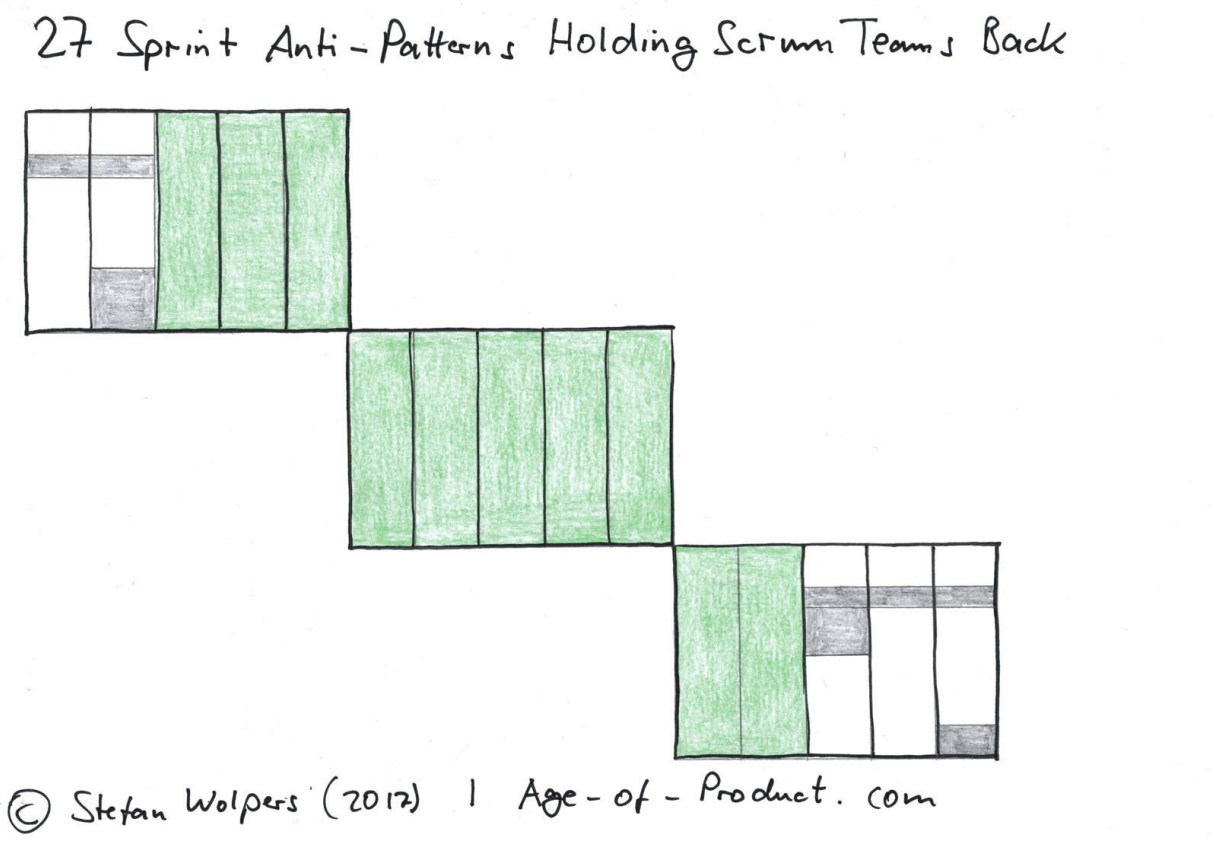
Conclusion

Scrum's sprint planning is a simple ceremony. Invest upfront during the product backlog refinement, and you will keep it productive. Most of the beforementioned sprint planning anti-patterns are simple to fix. Just take it to the team.

27 Sprint Anti-Patterns

The Sprint

The sprint is neither an official scrum ceremony nor an artifact. Obviously, it is not a role either. It is merely a time-box. Still, there are plenty of sprint anti-patterns to make your life as a scrum team harder than necessary.



Sprint Anti-Patterns

This list of notorious sprint anti-patterns applies to the development team, the product owner, the scrum master, the scrum team, as well as stakeholders and the IT management:

Sprint Anti-patterns of the Product Owner

- **Absent PO:** The product owner is absent most of the sprint and is not available to answer questions of the development team. (That creates a micro-waterfall

approach for the duration of the sprint.)

- **PO clinging to tasks:** The product owner cannot let go product backlog items once they become sprint backlog items. For example, the product owner increases the scope of a user story. Or, he or she changes acceptance criteria once the team accepted the issue into the sprint backlog. (There is a clear line: before a product backlog item turns into a sprint backlog item the product owner is responsible. However, once it moves from one backlog to the other, the development team becomes responsible. If changes become acute during the sprint the team will collaboratively decide on how to handle them.)

- **Inflexible PO:** The product owner is not flexible to adjust acceptance criteria. (If the work on a task reveals that the agreed upon acceptance criteria are no longer achievable or waste, the scrum team needs to adapt to the new reality. Blindly following the original plan violates a core scrum principle.)

- **Delaying PO:** The product owner does not accept sprint backlog items once those are finished. Instead, he or she waits until the end of the sprint. (In the spirit of continuous integration, the product owner should immediately check tasks that meet the acceptance criteria. Otherwise, the product owner will create an artificial queue which will increase the cycle-time. This habit puts also reaching the sprint goal at risk.)

- **Misuse of sprint cancellation:** The product owner cancels sprints to impose his or her will onto the team. (It is the prerogative of the product owner to cancel sprints. However, the product owner should not do this lightly without a serious cause. The product owner should also never abort a sprint without consulting the development team first. Probably, the team has an idea how to save the sprint. Lastly, misusing the cancelation privilege also indicates a serious team collaboration issue.)

- **No sprint cancellation:** The product owner does not cancel a sprint whose sprint goal can no longer be achieved. (If the product owner identified a unifying sprint goal, for example, integrating a new payment method, and the management then abandons that payment method mid-sprint, continuing working on the sprint goal would be waste. In this case, the product owner should cancel the sprint.)

Sprint Anti-patterns of the Development Team

- **No WiP limit: T**here is no work in progress limit. (The purpose of the sprint is to deliver a potentially shippable product increment that provides value to either the customers or the organization. This goal requires getting something done by the end of the sprint. The flow theory suggests that the productivity of a team improves with a work-in-progress (WiP) limit. The WiP limit defines the largest number of tasks a

team can work on at the same time. Exceeding this WiP number results in creating extra queues that reduce the throughput of the team. The cycle time which is the period between starting and finishing a ticket measures this effect.)

- **Cherry-picking:** The team cherry-picks work. (This effect often overlays with the missing WiP issue. Human beings are motivated by short-term gratifications. It just feels good to solve yet another puzzle from the board, here: coding a new task. By comparison to this dopamine fix, checking how someone else solved another problem during code review is less rewarding. Hence you often notice tickets queueing in the code-review-column, for example.)

- **Board out-of-date:** The team does not update tickets on the board in time to reflect the current statuses. (The board, no matter if it is a physical or digital board, is not only vital for coordinating a team's work. It is also an integral part of the communication of the scrum team with its stakeholders. A board that is not up-to-date will impact the trust the stakeholders have in the scrum team. Deteriorating trust may then cause counter-measures on the side of the stakeholders. The (management) pendulum may swing back toward traditional methods as a consequence. The road back to PRINCE II is paved with abandoned boards.)

- **Side-gigs:** The team is working on issues that are not visible on the board. (While sloppiness is excusable, siphoning off resources, and by-passing the product owner – who is accountable of the scrum team's return on investment – is inacceptable. This behavior also signals a massive conflict within the "team." Given this display of distrust—why didn't the engineers address this seemingly important issue during the sprint planning or before—the team is probably rather a group anyway.)

- **Gold-plating:** The team increases the scope of the sprint by adding unnecessary work to sprint backlog items. (This effect is often referred to as scope-stretching or gold-plating. The development team ignores the original scope agreement with the product owner. For whatever reason, the team enlarges the task without prior consulting of the product owner. This ignorance may result in a questionable allocation of resources. However, there is a simple solution: the developers and the product owner need to talk more often with each other. If the product owner is not yet co-located with the development team now would be a good moment to reconsider.)

Sprint Anti-patterns of the Scrum Master

- **Flow disruption:** The scrum master allows stakeholders to disrupt the flow of the development team during the sprint. (There are several possibilities how stakeholders can interrupt the flow of the team during a sprint. For example:

    - The scrum master has a laissez-faire policy as far as access to the development team is concerned.
    - The scrum master does not object that the management invites engineers to random meetings as subject matter experts.
    - Lastly, the scrum master allows that either stakeholders or managers turn the daily scrum into a reporting session.
      Any of these behaviors will impede the team's productivity. It is the scrum master's obligation to prevent them from manifesting themselves.)

- **Lack of support:** The scrum master does not support team members that need help with a task. (Often, development teams create tasks an engineer can finish within a day. However, if someone struggles with such a task for more than two days without voicing that he or she needs support, the scrum master should address the issue. By the way, this is also the reason for marking tasks on a physical board with red dots each day if they do not move to the next column.)

- **Micro-management:** The scrum master does not prevent the product owner—or anyone else—from assigning tasks to engineers. (The development team organizes itself without external intervention. And the scrum master is the shield of the team in that respect.)

- **#NoRetro:** The scrum master does not gather data during the sprint that supports the team in the upcoming retrospective. (This is self-explanatory.)

Note: I do not believe that it is the task of the scrum master to move tickets on a board. The team members should do this during the stand-up. It is also not the responsibility of the scrum master to update an online board so that it reflects the statuses of a corresponding physical board. Lastly, if the team considers a burn-down chart helpful, the team members should also update the chart after the stand-up.

Sprint Anti-patterns of the Scrum Team

- **The maverick & the sprint backlog:** Someone adds an item to the sprint backlog without consulting the team first. (The fixed scope of a sprint backlog—in the sense of workload—is at the core of enabling the team to make a commitment or forecast. The scope is hence per se untouchable during the sprint. Changes of the composition of sprint backlog are possible, for example, when a critical bug pops up after a sprint's start. However, adding such an issue to the sprint backlog requires compensation. Another task of a similar size needs to go back to the product backlog. All these exceptions have in common that the scrum team decides collectively on them. No single teammate can add or remove an item to or from the sprint backlog.)

- **Hardening sprint:** The scrum team decides to have a hardening or clean-up sprint. (That is a simple one: there is no such thing as a hardening sprint in scrum. The goal of the sprint is the delivery of a valuable potentially shippable product increment. Often, scrum teams agree in advance on a standard of what "done" means – also known as DoD or definition of done –. Declaring buggy tasks "done" thus violates core principles of the team's way of collaboration. Hardening sprints are commonly a sign of a low grade of adoption of agile principles by the team or the organization. This is probably because the team is not yet cross-functional. Or quality assurance is still handled by a functional, non-agile silo with the product delivery organization.)

- **Delivering Y instead of X:** The product owner believes in getting X. The development team is working on Y. (This is not merely a result of an inferior product backlog refinement. This anti-pattern indicates that the team failed to create a shared understanding. There are plenty of reasons for this to happen, for example:
    - The product owner and the development team members are not talking enough during the sprint. (The product owner is too busy to answer questions from the team or attend the daily scrum. Or, the team is not co-located, etc.)
    - No development team member has ever participated in user tests. There is a lack of understanding the users' problems among the engineers. (This is the reason why engineers should also interview users regularly.)
    - The product owner presented a too granular user story, and no one from the development team cared enough to have a thorough look. The user story seemed ready.
    - Probably, the user story was missing acceptance criteria altogether, or existing acceptance criteria missed the problem. No matter the reason, the team should address the issue during the next retrospective.)

- **No sense of urgency:** There is no potentially shippable product increment at the end of the sprint. There was no reason to cancel the sprint either. It was just an ordinary sprint. (This is a sign that the scrum team lacks the sense of urgency to deliver at the end of the sprint. If it is acceptable to fail on delivering value at the end of the sprint, the whole idea behind scrum is questioned. Remember, a scrum team trades a commitment or forecast for inclusion in decision-making, autonomy, and self-organization. Creating a low-grade time-boxed Kanban and calling it "scrum" will not honor this deal. Therefore, it is in the best interest of the scrum team to make each sprint's outcome releasable even if the release will not materialize.)

- **New kid on the block:** The scrum team welcomed a new team member during the sprint. They also forgot to address the issue during sprint planning thus ending up overcommitted. (While it is acceptable to welcome new teammates during a sprint, the team needs to account for the resulting onboarding effort during the sprint planning and adjust its capacity. The new team member should not be a surprise. However, if the newbie turns out to be a surprise it is rather an organizational anti-

pattern.)

- **Variable sprint length:** The scrum team extends the sprint length by a few days to meet the sprint goal. (This is just another way of cooking the agile books to match a goal or a metric. This is not agile; it is just inconsequential. Stop lying to yourself, and address the underlying issues why the team outcome does not meet the sprint goal. **Note**: I would not consider a deviating sprint length during the holiday season at the end of the year to be an anti-pattern.)

Sprint Review Anti-patterns of the IT Management

- **All hands to the pumps w/o scrum:** The management temporarily abandons scrum in an all-hands-to-deck situation. (This is a classic manifestation of disbelief in agile practices, fed by command & control thinking. Most likely, canceling sprints and gathering the scrum teams would also solve the issue at hands.)

- **Reassigning team members:** The management regularly assigns team members of one scrum team to another team. (Scrum can only live up to its potential if the team members can build trust among each other. The longevity of teams is hence essential. Moving people between teams, on the contrary, reflects a project-minded idea of management. It also ignores the preferred team building practice that scrum teams should find themselves. All members need to be voluntarily on a team. Scrum does rarely work if team members are pressed into service.

  Note: It is not an anti-pattern, though, if two teams decide to exchange teammates temporarily. It is an established practice that specialists spread knowledge this way or mentor other colleagues.)

- **Special forces:** A manager assigns tasks directly to engineers, thus bypassing the product owner. (This behavior does not only violate core scrum principles. It also indicates that the manager cannot let go command and control practices. He or she continues to micromanage subordinates although a scrum team could accomplish the task in a self-organized manner. This behavior demonstrates a level of ignorance that may require support from a higher management level to deal with.)

Sprint Review Anti-patterns of Stakeholders

- **Pitching developers:** The stakeholders try to sneak in small tasks by pitching them directly to developers. (Nice try #1.)

- **Everything's a bug:** The stakeholders try to speed up delivery by relabeling their tasks are 'serious bugs'. (Nice try #2. A special case is an "express lane" for bug fixes and other urgent issues. Every stakeholder will try and make his or her tasks eligible

for that express lane.)

- **Distrupting the flow:** The stakeholders disrupt the flow of the scrum team. (See above, scrum master section.)

Conclusion

Although the sprint itself is just a time-box there are plenty of sprint anti-patterns to observe. A lot of them are easy to fix by the scrum team. Other sprint anti-patterns, however, point at organizational issues that probably will require more than a retrospective to change.

14 Sprint Review Anti-Patterns

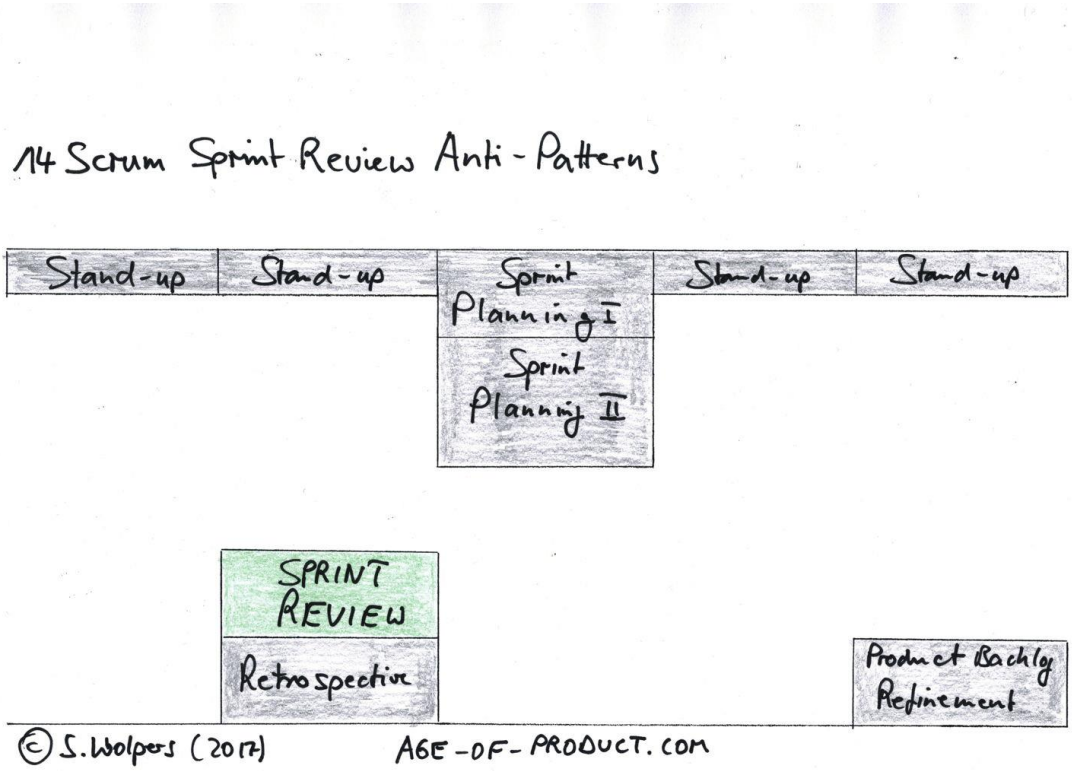
The Sprint Review

Are we still on the right track? Answering this question in a collaborative effort of the scrum team as well as internal (and external) stakeholders is the purpose of the sprint review. Given its importance, it is worthwhile to tackle the most common sprint review anti-patterns.

The Purpose of Scrum's Sprint Review

The sprint review is about Scrum's core principle: inspect & adapt. The development team, the product owner, and the stakeholders need to figure out whether they are still on track delivering value to customers. It is the best moment to create or reaffirm the shared understanding among all participants whether the product backlog is still reflecting the best use of the scrum team's resources. It is also because of this context that calling the sprint review a "sprint demo" does not match its importance for the effectiveness of the scrum team.

The sprint review is thus an excellent opportunity to talk about the general progress of the product. The sprint review's importance is also the reason to address sprint review anti-patterns as soon as possible.



Sprint Review Anti-Patterns

Typically, you can observe the following sprint review anti-patterns.

Sprint Review Anti-patterns of the Product Owner

- **Selfish PO**: The product owner presents "his or her" accomplishments to the stakeholders. (Remember the old saying: There is no "I" in "team"?)
- **Delayed sprint acceptance**: The product owner uses the sprint review to accept user stories. (This should be decoupled from the sprint review. The product owner should accept user stories the moment they meet the acceptance criteria.)
- **Unapproachable PO**: The product owner is not accepting feedback from the stakeholders. (Such a behavior violates the prime purpose of the sprint review ceremony.)

Sprint Review Anti-patterns of the Development Team

- **Death by PowerPoint**: Participants are bored to death by PowerPoint. (The foundation of a successful sprint review is "show, don't tell," or even better: let the stakeholders drive the ceremony.)

- **Same faces again**: It is always the same representatives from the development team who participate. (Unless the organization works with several teams based on LeSS (large scale Scrum), this is not a good sign. The challenge is that you cannot enforce the team's participation either, though. Instead, make it interesting enough that everyone wants to participate. **Note**: If the team does not attend religiously in full strength in each sprint review it is not an anti-pattern per se. However, there should be some rotation among participating team members.)

- **Side gigs**: The development team was working on issues outside the sprint scope. The product owner learns about those for the first time during the sprint review.

- **Cheating**: The development team demos items that are still buggy. (There is a good reason to show unfinished work on some occasions. Buggy work on the other side violates the DoD at an unacceptable level.)

Sprint Review Anti-patterns of the Scrum Team

- **Following a plan:** The scrum team does not use the sprint review to discuss the current state of the product or project with the stakeholders. (Again, getting feedback is the purpose of the exercise. A we-know-what-to-build attitude is bordering on hubris. **Read More**: Sprint Review, a Feedback Gathering Event: 17 Questions and 8 Techniques.)

- **Sprint accounting:** Every task accomplished is demoed, and stakeholders do not take it enthusiastically. (Tell a compelling story at the beginning of the review to engage the stakeholders. Leave out those user stories that are not relevant to the story. Do not bore stakeholders by including everything that was accomplished. We are not accountants.)

Sprint Review Anti-patterns of the Stakeholders

- **Scrum à la stage-gate:** The sprint review is a kind of stage-gate approval process where stakeholders sign off features. (This anti-pattern is typical for organizations that use an agile-waterfall hybrid. Otherwise, it is the prerogative or the product owner to decide what to ship when.)

- **No stakeholders**: Stakeholders do not attend the sprint review. (There are several reasons why stakeholders do not go to the sprint review: they do not see any value in the ceremony. It is conflicting with another important meeting. They do not understand the importance of the sprint review event. No sponsor is participating in the sprint review, for example, from the C-level. To my experience, you need to "sell" the ceremony within the organization.)

- **No customers**: External stakeholders – also known as customers – do not attend the sprint review. (Break out of your organization's filter bubble, and invite some paying users of your product.)

- **Starting over again**: There is no continuity in the attendance of stakeholders. (Longevity is not just a team issue, but also applies to stakeholders. If they change too often, for example, because of a rotation scheme, how can they provide in-depth feedback? If this pattern appears the team needs to improve how stakeholders understand the sprint review.)

- **Passive stakeholders**: The stakeholders are passive and unengaged. (That is simple to fix. Let the stakeholders drive the sprint review and put them at the helm. Or organize the sprint review as a science fair with several booths.)

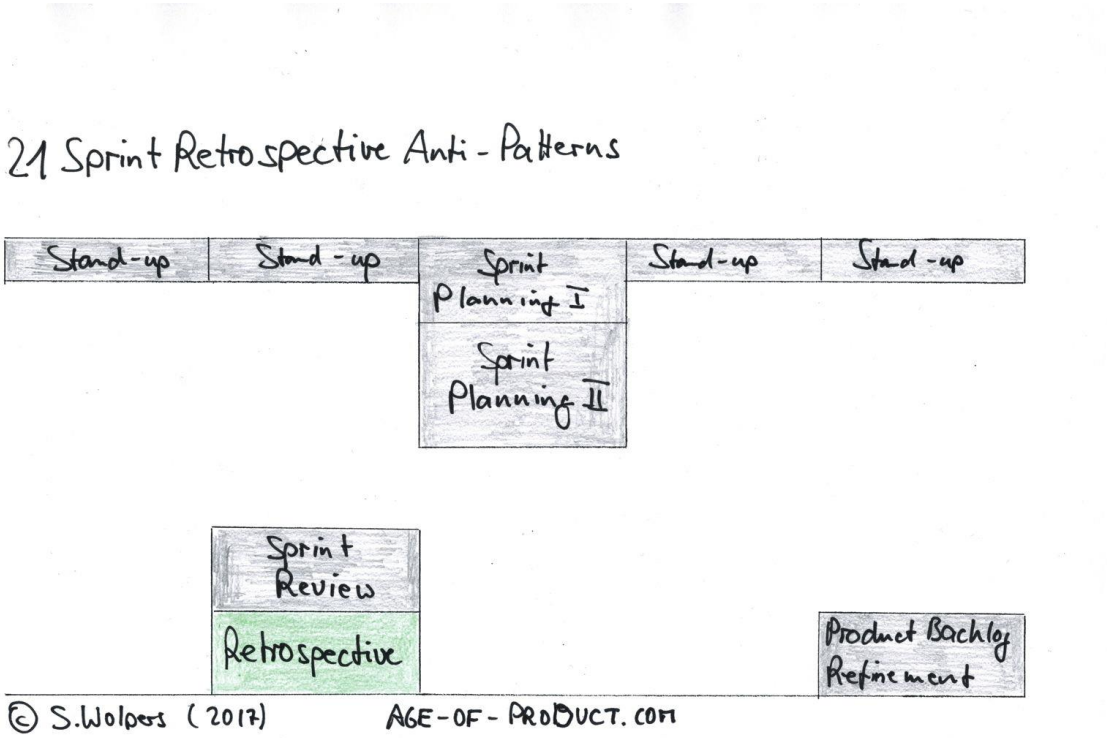
Conclusion

Scrum's sprint review is a simple yet meaningful ceremony. It answers the question whether the scrum team is still on track delivering value to the customers and the organization. Avoiding the sprint review's anti-patterns can significantly improve a team's effectiveness.

21 Sprint Retrospective Anti-Patterns Impeding Scrum Teams

Introduction

What ceremony could better embody scrum's 'inspect and adapt' mantra than the sprint retrospective? I assume all agile peers agree that even the simplest retrospective—if only held regularly—is far more useful than having a fancy one once in a while, or in the worst case having none at all. And there is always room for improvement. Learn more about 21 common sprint retrospective anti-patterns.



Sprint Retrospective Anti-Patterns

No matter the frequency of your retrospectives you should always watch out for the following sprint retrospective anti-patterns from the scrum team, the development team, the scrum master, as well as the organization:

Sprint Retrospective Anti-Patterns of the Scrum Team

- **#NoRetro**: There is no retrospective as the team believes there is nothing to improve. (There is no such thing as an agile Nirwana where everything is just perfect. As people say: becoming agile is a journey, not a destination, and there is always something to improve.)

- **Dispensable buffer**: The team cancels retrospectives if more time is needed to accomplish the sprint commitment/forecast. (The retrospective as a sprint emergency reserve is a common sign of cargo cult agile. I believe, it is even a worse anti-pattern than not having a retrospective because there is presumably nothing to improve. That is just an all too human fallacy bordering on hubris. However, randomly canceling the retrospective to achieve a sprint goal is a clear sign that the team does not understand basic agile principles, such as continuous improvement. If the scrum team repeatedly does not meet a sprint goal, it should inspect what is going on here. Guess which scrum ceremony is designed for that purpose?)

- **Rushed retrospective**: The team is in a hurry and allocates much less than the necessary 60 to 90 minutes for a retrospective. (That is a slippery slope and will probably end up with a ritualized ceremony of little value. Most team members will likely regard it as a waste sooner or later. Do it right by allocating whatever time is needed or consider stop having retrospectives. And while you are at it, why don't you abandon scrum altogether?)

- **Someone sings**: Someone from the participants provides information on the retrospective to an outsider. (For retrospectives, the Vegas rule applies: what is said in the room stays in the room. There is no exception from this rule.)

- **Extensive whining**: The team uses the retrospective primarily to complain about the situation and assumes the victim's role. (Change requires reflection, and occasionally it is a good exercise to let off steam. However, not moving on once you have identified critical issues and trying to change them defies the purpose of the retrospective. Limiting the number of stickies to 2-3 per participant may help to change this attitude. You may also consider balancing good and negative feedback by handing out an equal number of green and red stickies. Looks to be a bit too enforcing for my taste, though.)

- **UNSMART**: The team chooses to tackle UNSMART actions. (Bill Wake created the SMART acronym for reasonable action items: S – Specific, M – Measurable, A – Achievable, R – Relevant, T – Time-boxed. If the team picks UNSMART action items, though, it sets itself up for failure and may thus contribute to a bias that "agile" is not working. **Read More**: INVEST in Good Stories, and SMART Tasks.)

- **#NoAccountability**: Action items were accepted before. However, no one was chosen to be responsible for the delivery. (If the "team" is supposed to fix X, probably everyone will rely on his or her teammates to handle it. Make someone

accountable instead.)

- **What action?** The team does not check the status of the action items from the previous retrospectives. (The sibling of autonomy is accountability. If you are not following up on what you wanted to improve before why care about picking action items in the first place?)

Sprint Retrospective Anti-Patterns of the Development Team

Product owner non grata: The product owner is not welcome to the retrospective. (Some purists still believe that only the development team and the scrum master shall attend the team's retrospective. However, the Scrum Guide refers to the scrum team, including the product owner. It does so for a good reason: the team wins together, and the team loses together. How is that supposed to work without the product owner?)

Sprint Retrospective Anti-Patterns of the Scrum Master

- **Waste of time**: The team does not collectively value the retrospective. (If some team members consider the retrospective to be of little or no value it is most often the retrospective itself that sucks. Is it the same procedure every time, ritualized, and boring? Have a meta-retrospective on the retrospective itself. Change the venue. Have a beer- or wine-driven retrospective. There are so many things a scrum master can do to make retrospectives great again and reduce the absence rate. And yes, to my experience introverts like to take part in retrospectives, too.)

- **Prisoners**: Some team members only participate because they are forced to join. (Don't pressure anyone to take part in a retrospective. Instead, make it worth their time. The drive to continuously improve as a team needs to be fueled by intrinsic motivation, neither by fear nor by order. Tip: Retromat's "Why are you here?" exercise is a good opener for a retrospective from time to time.)

- **Groundhog day**: The retrospective never changes in composition, venue, or length. (There is a tendency in this case that the team will revisit the same issues over and over again – it's groundhog day without the happy ending, though.)

- **Let's have it next sprint**: The team postpones the retrospective into the next sprint. (Beyond the 'inspect & adapt' task, the retrospective shall also serve as a moment of closure that resets everybody's mind so that the team can focus on the new sprint goal. That is the reason why we have the retrospective before the planning of the follow-up sprint. Postponing it into the next sprint may interrupt the flow of the team. It also delays tackling possible improvements by up to a sprint.)

- **#NoDocumentation**: No one is taking minutes for later use. (A retrospective is a substantial investment and should be taken seriously. Taking notes and photos supports this process.)

- **No psychological safety**: The retrospective is an endless cycle of blame and finger pointing. (The team wins together, the team loses together. The blame game documents both the failure of the scrum master as the facilitator of the retrospective as well as the team's lack of maturity and communication skills.)

- **Bullying**: One or two team members are dominating the retrospective. (This communication behavior is often a sign of either a weak or uninterested scrum master. The retrospective needs to be a safe place where everyone–introverts included–can address issues and provide his or her feedback free from third party influence. If some of the team members are dominating the conversation, and probably even bullying or intimidating other teammates, the retrospective will fail to provide such a safe place. This failure will result in participants dropping out of the retrospective and render the results obsolete. It is the main responsibility of the scrum master to ensure that everyone will be heard and has an opportunity to voice his or her thoughts. By the way, equally distributed speaking time is according to Google also a sign of a high-performing team. **Read More**: What Google Learned From Its Quest to Build the Perfect Team.)

- **Stakeholder alert**: Stakeholders participate in the retrospective. (There are plenty of scrum ceremonies that address the communication needs of stakeholder: the sprint review, the product backlog refinement, the daily scrums, not to mention opportunities of having a conversation at water coolers, over coffee, or during lunchtime. If that spectrum of possibilities still is not sufficient, feel free to have additional meetings. However, the retrospective is off-limits to stakeholders.)

- **Passivity**: The team members are present but are not participating. (There are plenty of reasons for such a behavior: they regard the retrospective a waste of time, it is an unsafe place, or the participants are bored to death by its predictiveness. Probably, the team members fear negative repercussions in the case of their absence, or you managed to hire a homogenous group of East-European introverts. In other words: there is no quick fix, and the scrum master needs to figure out what kind of retrospective works in his or her organization's context.)

Sprint Retrospective Anti-Patterns of the Organization

- **No suitable venue**: There is no adequate place available to run the retrospective. (The least appropriate place to have a retrospective is a meeting room with a rectangular table surrounded by chairs. And yet it is the most common venue to have a retrospective. Becoming agile requires space. If this space is not available, you

should become creative and go somewhere else. If the weather is fine, grab your stickies and go outside. Or rent a suitable space somewhere else. If that is not working, for example, due to budget issues, remove at least the table so you can sit/stand in a circle. Just be creative. **Read More**: Agile Workspace: The Undervalued Success Factor.)

- **Line managers present**: Line managers participate in retrospectives. (This is the worst anti-pattern I can think off. It turns the retrospective into an unsafe place. And who would expect that an unsafe place triggers an open discussion among the team members? Any line manager who insists on such a proceeding signals his or her lack of understanding of basic agile practices. Note: If you are small product delivery team at a start-up and your part-time scrum master (or product owner) also serves in a management function, retrospectives might be challenging. In this case, consider hiring an external scrum master to facilitate meaningful retrospectives.)

- **Let us see your minutes**: Someone from the organization—outside the team—requires access to the retrospective minutes. (This is almost as bad as line managers who want to participate in a retrospective. Of course, the access must be denied.)

The Conclusion

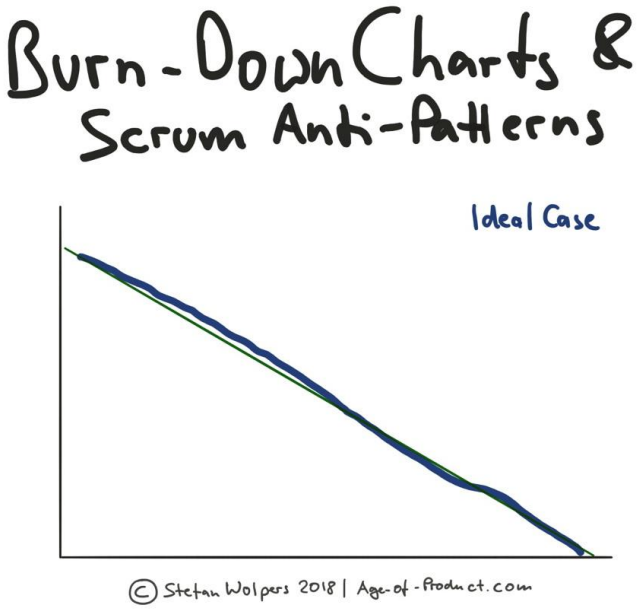
There are many ways in which a retrospective can be a failure even if it looks favorable at first glance. The top three sprint retrospective anti-patterns from my perspective are: not making the retrospective a safe place, unequally distributed speaking time, and a ritualized format that never changes.

Use Burn-Down Charts to Discover Scrum Anti-Patterns

Introduction

A burn-down chart tracks the progress of a team toward a goal by visualizing the remaining work in comparison to the available time. So far, so good. More interesting than reporting a status, however, is the fact that burn-down charts also visualize scrum anti-patterns of a team or its organization.

Learn more about discovering these anti-patterns that can range from systemic issues like queues outside a team's sphere of influence and other organizational debt to a team's fluency in agile practices.



Scrum Anti-Patterns Visualized by Burn-Down Charts

Burn-down charts have become popular to provide team members as well as stakeholders with an easy to understand status whether a sprint goal will be accomplished. (Critics of the

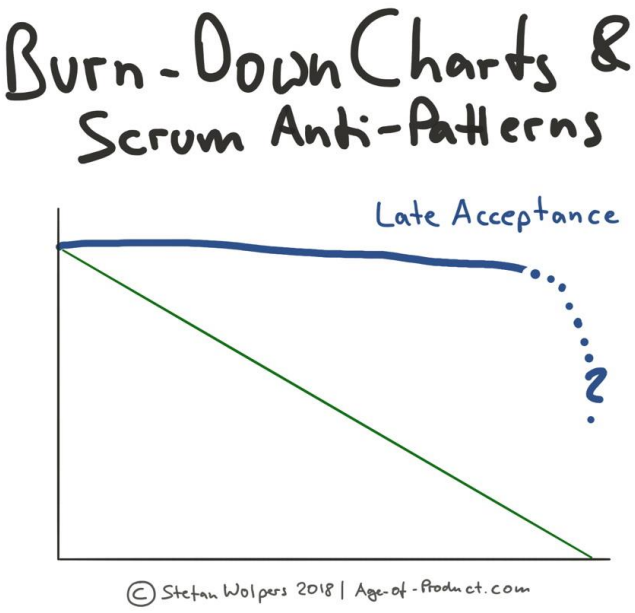
burn-down chart may note, though, that a scrum team should have a gut feeling anyway whether the sprint goal is achievable.)

Hence, this post is focusing on another useful aspect of burn-down charts: they are equally well suited to provide additional insights into all kind of impediments, both at a team level and at an organizational level.

The following graphs visualize four of the typical anti-patterns that can be easily detected with burn-down charts:

1. Late Acceptance

The product owner accepts or rejects tasks only late in the sprint:



This behavior may be rooted in various issues, for example:

- **The absent product owner**: The product owner is rarely available for the team to clarify matters and accept work. This is creating an artificial queue that has a

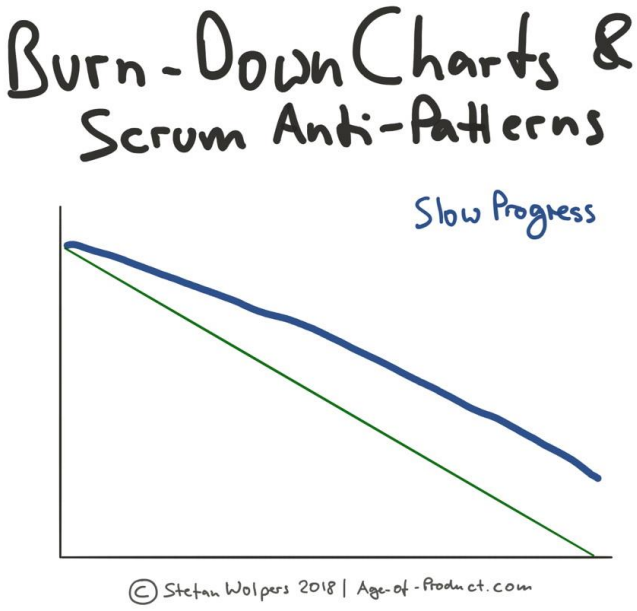
diminishing effect on the team's ability to deliver value by delaying the necessary clarification of tasks or the shipment of tasks themselves. (Note: LeSS susceptible for this effect when the product owner when not willing to delegate responsibility.)

- **The proxy product owner**: The team is working in a remote setup and the product owner is not onsite with the rest of the team. (Note: A proxy product owner is usually not a solution as he or she will just increase the time for feedback and add to the communication problems.)

- **Consequences**: There will likely be a spill-over to the next sprint as the feedback loop does not provide enough time to fix issues during the sprint. The team will probably not meet the sprint goal. If this not an isolated incident but a persistent pattern, action needs to be taken.

2. Slow Progress

In this case, the graph is located above the line of the expected progress for the complete sprint length:



There are several reasons why this might be the case:

- **The ambitious team**: The sprint goal is too ambitious and the team realizes only during the sprint that it will not deliver the sprint goal. (Note: It is okay to aim high and fail, however, it should not be the regular pattern as it is negatively influencing the trust of the organization in the team.)

- **The submissive team**: The sprint goal is too ambitious from an engineering perspective. However, instead of speaking up, the team tries to make it happen thus failing at the end of the sprint.

- **Capacity issues**: The capacity of the team changes after the sprint starts, for example, team members get sick, or they give notice and leave the team. (Note: Admittedly, this is hardly plannable anyway.)

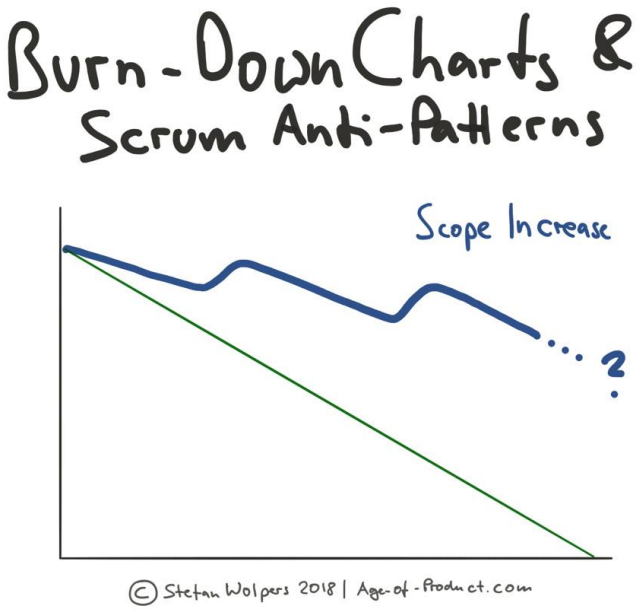
- **Change of priorities**: The team needs to address a critical issue—probably a bug—which leaves less capacity to accomplish the original sprint goal. (Note: Depending on the magnitude of the disturbance it might be useful to consider canceling the

spirit. At least, the team needs to reduce original sprint scope—which may require a mid-sprint re-planning to determine whether a reduced sprint backlog will still deliver the original sprint goal.)

- **Outside dependencies**: The team faces dependencies outside its sphere of influence not foreseeable during sprint planning. (Note: A classic systemic dysfunction.)

3. Scope Increase

The scope of work increases over the course of the sprint:



Most of the time, this pattern can be attributed to inadequate preparation:

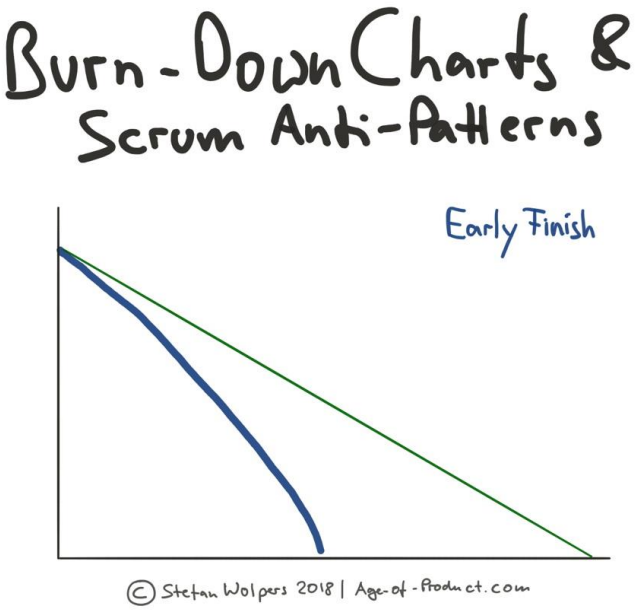
- **Refinement failure**: The scrum team fails to refine tasks accurately only to discover that the effort to create a valuable product increment is higher than originally expected. (Note: If this happens multiple time during the sprint then the team accepted stories into the sprint the team has not fully understood. This points at

serious issues with the product backlog refinement process or the collaboration with the product owner in general.)

- **Dynamic sprint backlog**: Urgent tasks are pressed or find their way into the sprint without compensation. (Note: Depending on the magnitude of the tasks, canceling the current sprint and focusing on the apparently more valuable new issues might be the better alternative. Unless, of course, those new issues are hacking the scrum process of sprint planning. There are several examples of this behavior: A manager pulls strings to get his or her task into a sprint or tasks are disguised as critical bugs that need to be fixed immediately.)

4. Early Finish

The team accomplishes the sprint goal way earlier than expected:



Of course, an early finish is the anti-anti-pattern if the team figured out how to deliver a task with much less effort than expected. Or the sprint goal could be achieved with fewer tasks that planned.

However, the positive news might also hint at some problems. Again, the reasons for this phenomenon are multi-faceted. My two top-candidates are:

- **The overly cautious team**: The team probably overestimated the effort to be on the safe side with its prediction. (Note: This could indicate that the management tracks, for example, velocity as an important metric for the contribution of the team members despite its limited usefulness. Or the organization is output oriented and does not accept 'failure' as an option. In these cases, the organization is setting the wrong incentives. See also the Hawthorne effect.)

- **A hack for slack time**: The team included buffer time to be able to address technical debt, its need for pairing or other issues that do not regularly receive attention and hence managed to finish early. (Note: This might indicate that the current allocation of resources is neglecting the long-term health of the team as well as the code base. Also, watch out for the feature factory syndrome where team utilization and output matter more than the long-term outcome.)

Note: These anti-patterns are only recognizable if the team provides the necessary transparency.

The Conclusion

It is a good idea to use burn-down chart patterns for the next retrospective as they easily identify team problems or systemic dysfunctions. And utilizing burn-down charts in that capacity does not even require switching to story points per se—equally sized stories can just be counted to create a dimension for the y-axis.

Enhancing burn-down charts with additional data, for example, context and occurrences, as well as lead time and cycle time values, will increase the benefit of burn-down charts even more.

Speaking of which: At the team level, I would suggest creating a rotating scheme of team members to update the burn-down chart daily. It is a team exercise and not the job of the scrum master.

Lastly, no matter what purpose you are using burn-down charts for, avoid falling into a common trap: Start counting subtasks. This accounting will quickly lead you on the track of abandoning your definition of done. Instead, you will start marking tasks as 90 % complete. Welcome to cargo cult agile—how would that differ from the waterfall approach?

Continuous Improvement: Gathering Feedback with Retrospectives

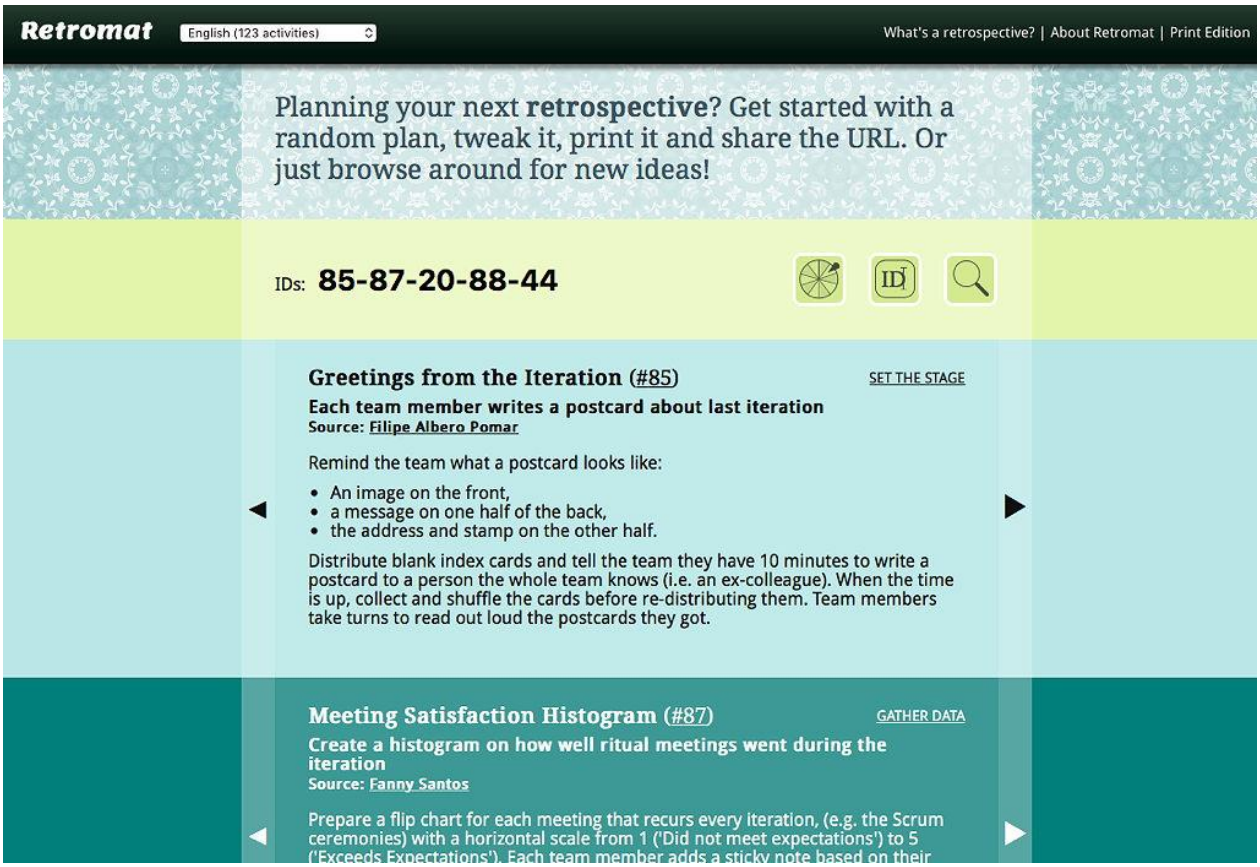
How to Curate Retrospectives with Retromat

Admittedly, up to now my retrospective style has been a sort of opportunistic, depending on the sprint in question.

However, facing three retrospectives in the next week with recently created Scrum teams, I decided to change this approach. I spent two days on going through all (English) modules of the Retromat, and decided to curate retrospectives in advance, one for each team. In the first step, I picked my favorite exercises from each of the five stages:

1. Set the stage
2. Gather data
3. Generate insights
4. Decide what to do
5. Close the retrospective,

and then decided to ignore the rest. (Although, the exercises are supposed to be interchangeable, I believe that some of the 7,084,005 combinations actually do not work that well together. Some combinations just break a cohesive story, making the retrospective experience a bit random.)



Then I created a sort of storyline for each of the three retrospectives, by starting with me first favorite of the “set the stage” category, connecting it to a suitable favorite from the “gather data” stage. Once the two were identified, I started looking for the one from the “generate insights” stage, and so on until all five stages were set, telling a cohesive story.

Theme-wise, I picked the UEFA Euro 2016 cup for decision making, the upcoming Berlin regional election to generate insight (here the election manifesto for change), as well as the “The Worst We Could Do” exercise—I still cannot quite believe that #Brexit is now a reality. If you like to try my curated retrospectives yourself, here we go:

1. UEFA Euro 2016
2. (Berlin) election manifesto for change
3. Brexit

What are your favorite Retromat retrospectives? Please share with me in the comments. Maybe, we manage to build a pool of tried and tested retrospectives that way.

Thanks to Corinna Baldauf for creating and maintaining this amazing repository of agile practices.

Retrospective Exercises Repository

Summary: The Retrospective Exercises Repository

How to prevent retrospective boredom? One way to achieve that is never to repeat the same combination of retrospective exercises twice.

Avoiding repetitions might sound like much work for a single team. However, if your product delivery organization comprises of more than one Scrum team, I can highly recommend creating a retrospective exercises repository as it improves the quality of the retrospectives and saves much time if you share the retrospective exercises with your fellow scrum masters.

Curating Retrospectives with Retromat

Retromat aggregates a lot of suitable exercises for retrospectives, covering the five stages of retrospectives as suggested by Esther Derby and Diana Larson in their excellent book "Agile Retrospectives: Making Good Teams Great."

Esther and Diane distinguish five stages:

- Set the stage
- Gather data
- Generate insights
- Decide what to do
- Close the retrospective.

Retromat provides a lot of exercises for each of those stages. As a first step, I went through all the exercises listed for the different stages and picked my choices from them. To do so in a structured way, I used an Excel sheet:

Retrospectives Planning

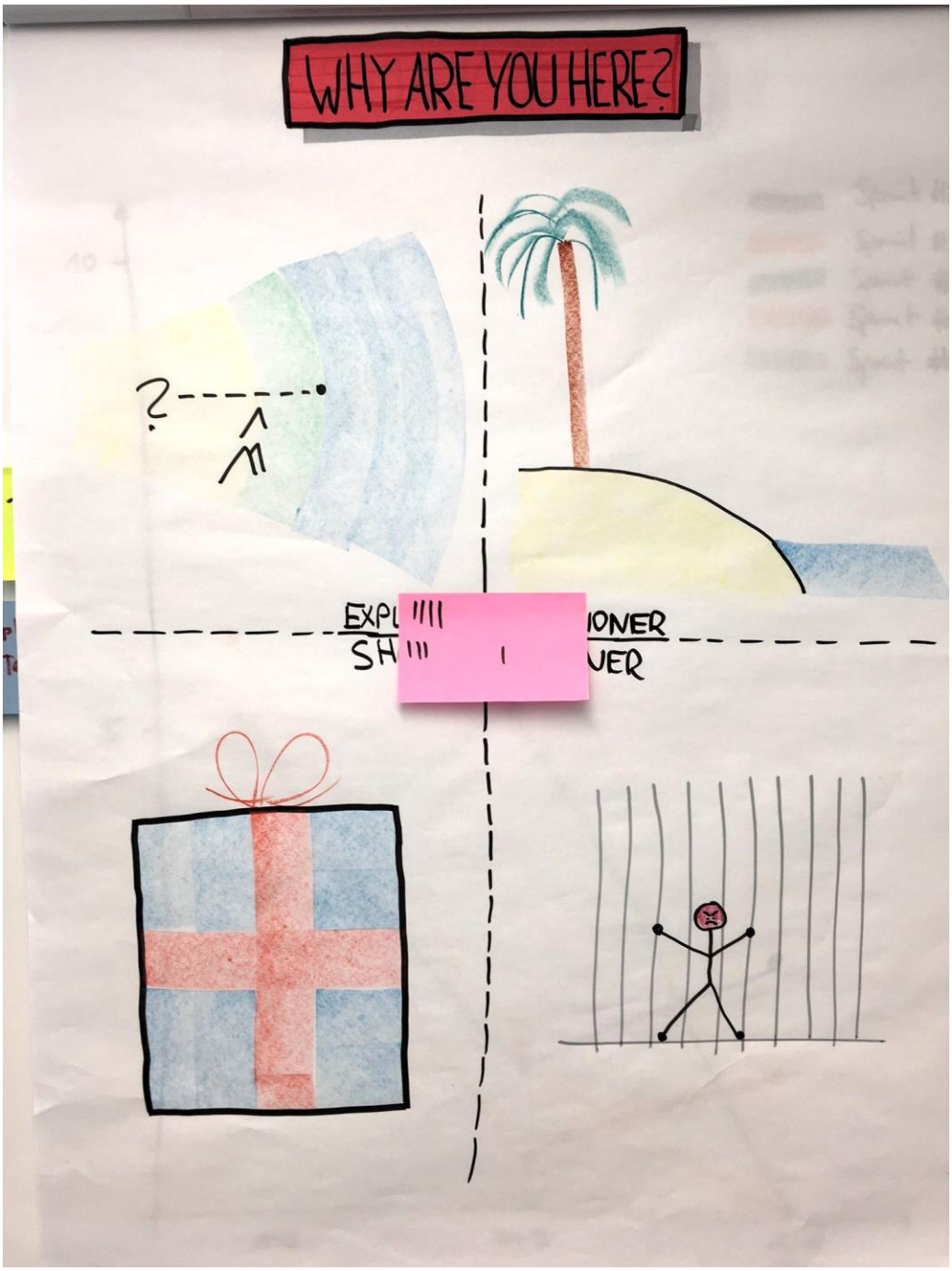
I. Repository of Exercises:

Activity	Stage	Title	URL	Available?	Team 1	Team 2
1	1 Set the stage	ESVP	https://plans-for-retrospectives.com/en/?id=1	Y	6	
2	1 Set the stage	Weather report		Y	7	1
3	1 Set the stage	Check in: One word (e.g. emotional state: glad, sad, mad, scared?				
18	1 Set the stage	Amazon Review		Y	x	
31	1 Set the stage	Draw the iteration				
32	1 Set the stage					
43	1 Set the stage	Take a stand : Opening; refers to the last iteration/sprint				
46	1 Set the stage	Why Do We Do Retrospectives?				
70	1 Set the stage	Satisfaction with Iteration Results				
85	1 Set the stage	Postcard: Greetings from the iteration/sprint				
90	1 Set the stage	Agile Values Cheer Up				
122	1 Set the stage					
N1	2 Gather data	Top 3 impediments		Y	x	
N3	2 Gather data	Interruption bucket				
4	2 Gather data	Timeline		Y	10	1
N15	2 Gather data	Oth'r Issue?		Y		
7	2 Gather data					
19	2 Gather data	Sailing boat: Propulsion & Inertia		Y	7	
35	2 Gather data	Agile self-assessment				
47	2 Gather data	Empty the mail box				2
51	2 Gather data					
54	2 Gather data					
78	2 Gather data					
87	2 Gather data	Meeting satisfaction histogram				
89	2 Gather data					
97	2 Gather data					
98	2 Gather data					
116	2 Gather data					
123	2 Gather data					

Identifying the exercises for future retrospectives is one thing. (The Retromat number is listed in the first column 'activity'.) Producing the exercises in advance is something different. Doing this upfront for your complete selection is bordering on waste in my eyes. So, I create new exercises on flip chart paper only shortly before a retrospective. (Which is also a good way to familiarize yourself with the exercise itself.) Once an exercise becomes available, I mark it in the Excel sheet with a green background.

For each scrum team, I also add a column to the retrospective exercises repository indicating in which retrospective an exercise was used. (Normally, I use the number of the individual sprint here.) This allows for a quick overview when a certain retrospective exercise was utilized for the last time for a team retrospective. (Of course, I also save all exercises for every retrospective of each team.)

To my experience, if you keep at least three to four retrospectives between the usages of an individual exercise you are on the safe side. Also, try to avoid reusing more than two exercises in a retrospective again. It is interesting to observe that even my clumsy graphics have a considerable potential to be remembered. (Probably, it is also because of their clumsiness.) Anyway, there are more than enough permutations of exercises through all five stages of a retrospective available that you can work around a repetition of activities.



Storing the Retrospective Exercises

When it comes to storing the flip chart sheets, I recommend a small filing cabinet from Bankers Box. They are outrageously expensive — almost 30 Euros in Germany — but very well designed for this purpose. A filing cabinet can hold up to 20 flip chart sheets. Two of those cabinets will suffice for any need to organize retrospectives.

The easiest way to order the exercises is by consecutive numbers, not by retrospective stages. This way is particularly helpful when you branch out beyond Retromat for activities.

Speaking of branching out: there are other sources for respective exercises, for example, Fun Retrospectives or Ben Linders' retrospective exercises.

I add those exercises with a different number range to the Excel sheet, typically starting with an N. Lastly, I also add dedicated games to the respective exercises repository, for example, the candy game.

Conclusion

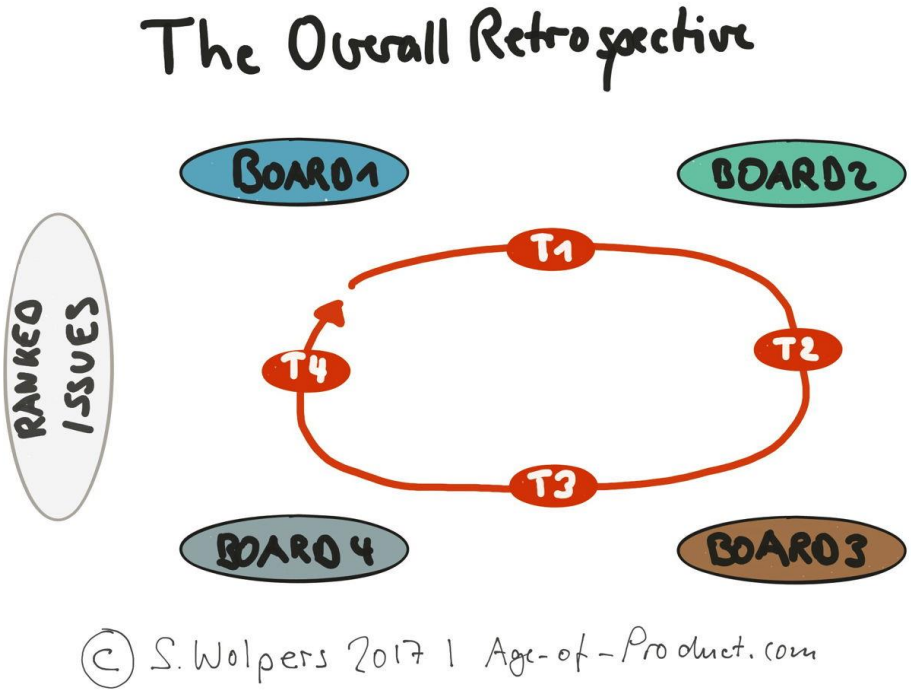
Creating a retrospective exercises repository is an easy way to make retrospectives more engaging as well as save efforts when your product delivery organization comprises of more than one scrum team. Remember: sharing is caring.

The Overall Retrospective for Team and Stakeholders

Introduction

After rebuilding an existing application on a new tech stack within time and under budget our team had an overall retrospective with stakeholders this week to identify systemic issues. We found more than 20 problems in total and derived eight detailed recommendation the organization will need to address when moving forward to the next level of agile product creation.

Read on and learn how we achieved this result in under two hours with an overall retrospective attended by 16 people.



The Origin and Purpose of the Overall Retrospective

The concept of an overall retrospective is a ceremony borrowed from Large Scale Scrum:

"The Overall Retrospective is a new meeting in LeSS. Its purpose is to discuss cross-team, organizational and systemic problems within the organization."

Source: Large Scale Scrum.

We decided to adopt two changes by comparison: a) we ran the team's overall retrospective after achieving a significant result: the application launched successfully four weeks earlier, and b) all team members participated not just representatives from the team. (Instead of having two teams, we worked as a large, distributed team anyway.)

The purpose of this overall retrospective was to provide recommendations to the organization on how to improve working in an agile manner for future teams and products.

Preparing for the Overall Retrospective

To run an overall retrospective, you need to have:

- A large room that provides enough space to host up to 20 people working in front of several boards and walking between them
- Large stickies in different colors—each team shall have a different color
- Sharpies
- Sticky dots for dot-voting
- Board labels that address the topics of the overall retrospective.

For the intended discovery and discussion, I picked the following four topics:

- Working with the organization
- Working with customers
- Agile & Scrum
- Technical excellence.

Kicking off the Overall Retrospective

The overall retrospective started with creating four teams — one for each topic. Given that sixteen people attended the ceremony we ended up with four teams of four people each working in parallel on four issues.

The overall retrospective had three different phases:

The Ideation Phase

In the ideation phase, all teams were working at the same time for seven minutes on a topic, each creating their three most important issues in an open discussion. At the end of the time-box, the team moved the resulting three stickies to the backside of the board to prevent creating bias in the following group. Then each team moved on to the neighboring board. This way, all teams were cycling through all four topics within about 30 minutes.

The Clustering Phase

At the end of the fourth cycle, each team was asked not just to move their three issues to the backside but also have a look at the already available nine other stickies. Their task was to cluster all twelve stickies at the board into similar topics to prepare for the third phase of the overall retrospective: the ranking of the issues by dot-voting.

The clustering phase took about ten minutes as we discovered that we listed similar topics on different boards. The team hence decided to have a brief check on all four boards to identify related issues and put them on the same board.

The Ranking and Discussion Phase

The third phase of the overall retrospective started with a dot-voting on the identified issues across all boards. In total, each participant had six dots, and everyone was free to assign them. (There was no limit on the number of dots that could be attached to a single issue.) This part took about ten minutes.

The three highest rankings issues were:

- Autonomy in technical questions (17 out of 96 possible votes)
- Longevity of teams (11 out of 96 possible votes)
- Clash of cultures: agile vs. waterfall. (7 out of 96 possible votes.)
-

These top three issues attracted almost 35 percent of all available votes.

The final part of the overall retrospective was a lean coffee to create recommendations to the organization based on the top three ranking issues. This part required about 60 minutes and delivered eight suggestions in total, for example:

- Empower teams to make appropriate technical decisions for their product/project
- Involve the overall team in the initial definition phase of the product/project
- Don't move people on and off the team
- Empower the product owner in the sense of the Scrum Guide
- Align project governance with agile practices, for example, in procurement, legal, HR.

If you are now under the impression that this organization is still in the early stages of its agile journey you are right. The good news is, though, that the team's success also triggered that a promising development: the team will become next year a product team as the application is no longer considered a project.

Conclusion

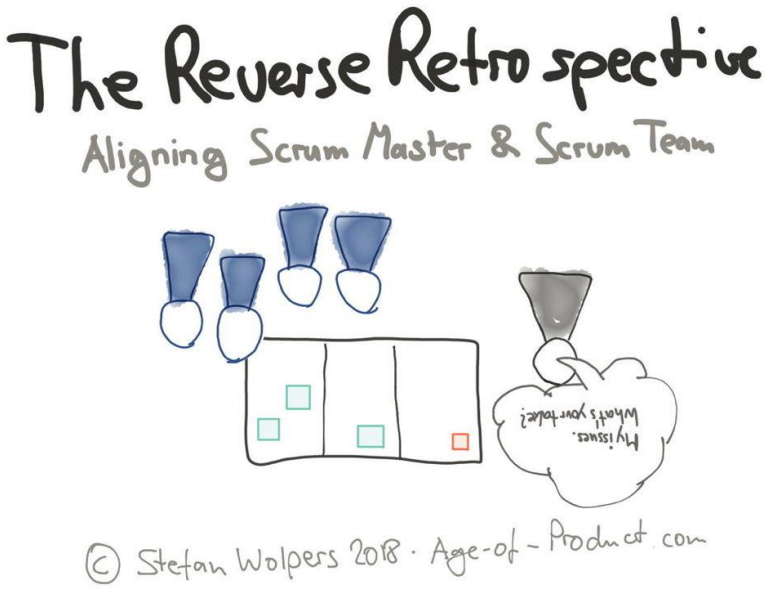
The overall retrospective with the team and its stakeholders proved to be a success. Everyone was positively surprised by the compact nature of the retrospective and its outcome. Of course, the overall retrospective cannot guarantee in the end that the recommendations will be accepted. At least, the people involved in building the new application lived up to their part of taking responsibility for continuous improvement.

The Reverse Retrospective — Aligning Scrum Team and Scrum Master

Introduction

Are you—as scrum master or agile coach—experiencing more communication kerfuffles with "your" team? Is its speed of improvement stalling? Are you under the impression that the team is slipping back into old habits and patterns? Maybe, it is time to run a reverse retrospective where your share your observations with the team.

Learn how to run a reverse retrospective to realign with your scrum team.



Reverse Retrospective: Framing the Problem—Groundhog Day

As a scrum master or agile coach, you may ask yourself sometimes why it takes your team so long to change direction even with issues under control of the team. This might be a simple technique such as 'put the Jira task number in red into the upper right corner of its index card, so we have a chance to read it from a distance.' Or it could be the understanding of fundamental agile practices. For example, that insufficient product backlog refinements will likely cause spill-overs at the end of a sprint, thus endangering meeting the sprint goal.

Such moments of falling behind or not living up to the potential are particularly unfortunate when they repeat themselves. Nothing is more frustrating than a team managing to pin down problems during retrospectives and deciding to take action only to not deliver on

those action items. Or, initially changing the situation for the better just to fall back into old patterns after a while slowly. It is groundhog day, the agile edition.

Self-Organizing Scrum Teams

One of the problems from a coaching perspective is that self-organizing teams do not form when the scrum master is instructing team members regularly what to do and how to do things, probably even enforcing those "suggestions" at a later stage.

Self-organization means that the team takes care of its affairs as autonomy without accountability equals anarchy. (It is what Patty McCord—formerly Netflix—refers to as treating everyone "like fully formed adults.")

One way out of this dilemma might be a retrospective exercise that aims at realigning the scrum team with the scrum master or agile coach—the reverse retrospective.

How to Run a Reverse Retrospective

A reverse retrospective combines several techniques from the scrum master toolbox:

- The scrum master creates stickies with issues
- The circles of influence-model by Diana Larsen allows the clustering of issues by team authority
- There is a dot-voting step to identify issues where the scrum team and the scrum master align and—more importantly—do not align
- There is a lean coffee-style discussion based on the dot voting results.

The desired outcome of the reverse retrospective is to confirm that team and coach have the same perspective on the team situation. If the retrospective reveals, that is not the case the ensuing discussion shall overcome these differences and realign the scrum team with the scrum master.

Preparing the Reverse Retrospective as the Scrum Master

The most important part of preparing the reverse retrospective as the scrum master is the identification of the issues you want to address. I start doing so in parallel with the daily work anyway by collecting lists of observations—a kind of problem backlog. I also revisit the protocols of past retrospectives and check the action items the team agreed on tackling.

I usually end up with 10 to 40 issues of various levels of severity during this step depending on the fluency of the scrum team. Some might be mission-critical while others are merely nice-to-have. Next, I rank the issues: to what extent are these impeding the team from reaching its goal, and what can the team do about them? (This ranking allows me to focus

the reserve retrospective on the top 10-15 issues at hand. Otherwise, the team might get lost in the noise.)

I also like to label the issues. A simple model will do, and my model covers:

- Organization
- Team
- Technology
- Process.

Then I transfer these issues to stickies or index cards. (I use cards of a single color.)

The room or space of the retrospective needs to be prepared with a matrix based on circles of influence-model by Diana Larsen. (See above.) I find it difficult to run the reverse retrospective on a single flipchart paper with the original circle design. A smooth wall, a large whiteboard or a window is much better suited for that purpose. Also, you need to ensure that the team has enough room in front of the wall for the dot-voting.

I also use a matrix design instead of original concentric circles as a matrix makes the exercise less complicated to follow visually: there is less overlap of stickies, it is easier to dot-vote and more straightforward to come up with action items.

The Kick-off of the Reverse Retrospective

I kick off the reserve retrospective by introducing the team to each issue explaining my thoughts or observations behind it. This should take no more than 45 to 60 seconds per point, so the introduction phase fits into a 10-minutes time-box.

During this introduction, I also place the issues into one of the three categories:

1. Team controls
2. Team influences
3. “The Soup.”

(If you are not familiar with the terms, please check Diana Larsen’s original post.)

The Dot-Voting

The following dot-voting by the team members is done with two differently colored dots: one color signals agreement on the issue at hand. The other color signals disagreement and the need for discussion. (Choose contrasting colors for this purpose. Or example, red and green dots will not work if someone among the team members is red-green color blind.) The team members assign dots by relevance from their perspective.

You will end up with issues where the alignment between scrum team and scrum master is apparent. Other questions might be controversial among team members which an approximately equal number of dots of both colors. The most interesting issues are those where the team and the scrum master are not aligned.

Ranking Issues for Discussion

My preferred ranking of issues for the following discussion is as follows:

1. Issues without alignment
2. Controversial issues. (Roughly an equal number of both dots.)
3. Issues with a high level of alignment.
4.

The purpose of the discussion is solving issues. In the case of misalignments, this might be as simple as fixing a communication kerfuffle. In other cases, the solution might require creating a task or action-item. According to Diana Larsen, there are three categories for that:

1. Direct action. (The team controls the issue.)
2. Recommending action. (The team influences the issue.)
3. Respond as a team accordingly. (The team has no authority over the issue.)

This categorization helps particularly to plan the next steps of improvement when the most pressing issues are of a systemic nature. Those have proven in the past to be of a more frustrating quality.

No matter the outcome, though, the reverse retrospective will likely improve the future collaboration of both scrum team as well as scrum master.

The Conclusion

Running a reverse retrospective is a helpful feedback loop between scrum team and scrum master or agile coach to assure alignment between them on the issues that impede the team's progress.

Consider running a reverse retrospective whenever the speed of improvement seems to slow down, communication kerfuffles become more frequent, or systemic issues hold back the team.

Indicators of Failure

Agile Failure Patterns in Organizations

Why Agile Is Simple, But Complex at the Same Time

Who wouldn't agree that the four core principles of the Agile Manifesto –

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan.

– aren't derived from applying common sense to a serious problem?

That the application of those principles might be suited to fix numerous organizational dysfunctions and reduce an error-prone and complex social setting to maybe just a complicated one?

There are various good reasons to adopt Agile, for example:

1. Low productivity,
2. Low morale,
3. Problems hiring senior people in an increasing competitive war for talent,
4. Senior people quitting,
5. Budget constrains (no more funds to waste on waterfall projects), or
6. The competition drives innovation at a high pace and the traditional methodology cannot keep up with them.

But fact is also, that the scope of an Agile organizational transformation is often completely underestimated. That Agile is not the quick fix for everything that's going wrong. Each organization has it own set of dysfunctions and hence solutions dealing with them need to be tailored specifically to that organization.

Agile Failure Patterns

Analyzing my past projects, I identified the following cross-organizational patterns that are making Agile transitions to much harder, less effective and significantly more expensive:

Agile Failure at Organizational Level:

- Not having a (product) vision in the first place: If you don't know where you are going, any road will get you there.
- The fallacy of "We know what we need to build". There is no need for product discovery or hypotheses testing, the senior management can define what is relevant for the product backlog.
- A perceived loss of control at management level leads to micro-management.
- The organization is not transparent with regard to vision and strategy hence the teams are hindered to become self-organizing.

- There is no culture of failure: Teams therefore do not move out of their comfort zones, but instead play safe. (Mathgeek on HN suggests to replace "culture of failure" with "culture of learning to get back up again after falling.")
- The organization is not optimized for a rapid build-test-learn culture and thus departments are moving at different speed levels. The resulting friction caused is likely to equalize previous Agile gains.
- Senior management is not participating in Agile processes, e.g. sprint demos, despite being a role model. But they do expect a different form of (push) reporting.
- Not making organizational flaws visible: The good thing about Agile is that it will identify all organizational problems sooner or later. „When you put problem in a computer, box hide answer. Problem must be visible!" Hideshi Yokoi, former President of the Toyota Production System Support Center in Erlanger, Kentucky, USA
- Product management is not perceived as the "problem solver and domain expert" within the organization, but as the guys who turn requirements into deliverables, aka "Jira monkeys".
- Other departments fail to involve product management from the start. A typical behavior in larger organizations is a kind of silo thinking, featured by local optimization efforts without regard to the overall company strategy, often driven by individual incentives, e.g. bonuses. (Personal agendas are not always aligned with the company strategy.)
- Core responsibilities of product management are covered by other departments, e.g. tracking, thus leaving product dependent on others for data-driven decisions.
- Product managers without a dedicated team can be problem, if the product management team is oversized by comparison to the size of the engineering team.

Agile Failure At Team Level:

- There are too many junior engineers on an engineering team. They tend to appreciate micro-management as part of their training. Usually, they have no or little experience with Agile methodologies, hence they hardly can live up to processes, particularly they fail to say "No".
- Engineers with an open source coding mentality: Tasks are discussed on pull requests once they're finished, but not in advance during grooming or sprint planning sessions. (They tend to operate in distributed team mentality.)
- Teams are too small and hence not cross-functional. Example: A team is only working on frontend issues and lacks backend competence. That team will always rely on an other team delivering functionality to build upon.
- Teams are not adequately staffed, e.g. Scrum Master positions are not filled and product owners have to serve two roles at the same time.
- Team members reject Agile methodologies openly, as they do not want to be pushed out of their comfort zones.

- Teams are not self-organizing. That would require to accept responsibility for the team's performance and a sense of urgency for delivery and value creation. A "team" in this mode is actually more acting like a group—e.g. people waiting at a bus-stop—: They are at the same time in the same place for the same purpose, but that does not result in forming a team. (The norming, storming, forming, performing cycle seems to be missing.)
- Even worse, team members abandon Agile quietly, believing it is a management fad that will go away sooner or later.
- **Cargo Cult Agile**: Teams follow the "Agile rules" mechanically without understanding why those are defined in the first place. This level of adoption often results often in a phenomenon called "Peak Scrum": There is no improvement over the previous process, despite all Agile rules are being followed to the letter. Or even worse: morale and productivity go down, as the enthusiasm after the initial agile trainings wears off quickly in the trenches.
- Moving people among teams upon short notice. Even if required for technical reasons, this has a negative impact on a team's performance & morale.

Agile Failure At Process Level:

- Agile processes are either bent or ignored whenever it seems appropriate. There is a lack of process discipline.
- Agile processes are tempered with, e.g. the Scrum Product Owner role is reduced to a project manager role. Often this is done so by assigning the task of backlog ownership to a different entity at management level. ("Scrum" without a Product Owner actually makes a great Waterfall 2.0 process.)
- Stakeholders are bypassing product management to get things done and get away with it in the eyes of the senior management, as they would show initiative.
- The organization is not spending enough time on team communication and workshops to create a shared understanding on what is to be built.

Agile Failure At Facility Level:

- A team is not co-located, not working in the same room, but scattered across different floors, or worse, different locations.
- The work environment is lacking spots for formal and – more important – informal communication: cafeterias, tea kitchens, sofas etc.
- The work environment is lacking whiteboards. Actually, the absence of whiteboards on each and every available wall within the office should be questionable, not having them.
- Agile requires suitable offices to further collaboration: spacy, with plenty of air and light. But they should not be a mere open space, which tends to get too noisy, particularly when several Scrum team are having stand-ups at the same time.

Why Engineers Despise Agile

The agile consulting industry repackages an originally human-centered, technology-driven philosophy into a standardized, all-weather project-risk mitigating methodology.

Sold to command & control organizations, their middle managers turn “Agile” into a 21. century adoption of Taylorism for knowledge workers. Beyond this meta-level, the reasons, why engineers despise Agile, fall into five categories: Control, manipulation, monitoring, technology and teamwork.

The Issues Why Engineers Despise Agile

When I wrote Agile Failure Patterns In Organizations, summarizing typical anti patterns of agile adoptions in organizations, I was surprised to see the traction it received on HN.
So, I started digging more into the topic. Up to then, I had been well aware of the problems that product people are facing concerning "Agile". You're dreaming of a Spotify-like place to work for, creating products customers love, while actually you're being stuck in a sort of Jira-monkey position, churning out user-stories.

From that background, engineers have always appeared to be natural allies for a good cause to me. Why wouldn't you want to be empowered, doing meaningful work, and having a purpose in your professional life?

Unsurprisingly, there are quite some outspoken critics among engineers. Don't get me wrong: not all engineers despise Agile. But there are serious concerns being voiced that fall into five categories:

I. Control

The middle management is not willing to give up control to empower teams, thus contributing to the creation of a learning organization. Hence, the agile ideal of transparency and radiating information within the organization ultimately leads to an increase in surveillance.

Read more on the agile micromanagement aspect: Agile Micromanagement in the Era of Autonomy, Mastery and Purpose.

II. Manipulation

The pursuit of personal agendas also drives the employment of external consultants to enforce the middle management's perspective of the right agile process by sticking to the "rules"—aka: cargo cult Agile.

This approach does not completely ignore the Shu-Ha-Ri principle of learning a new technique:

"The fundamental idea here is that when teaching a concept, you have to tailor the style of teaching to where the learner is in their understanding and that progression follows a common pattern. Early stages of learning focus on concrete steps to imitate, the focus then shifts to understanding principles and finally into self-directed innovation.".

It rather starts with applying the “rules” by the book in phase one and then conveniently sticks to them, ignoring phases two and three. And by doing so, the skeleton Agile is turned into yet another management style that advocates following a plan—just in shorter intervals, called sprints.

III. Monitoring

Agile mechanics are adopted whenever beneficial, but the culture itself is not changed. In the case of transparency and metrics, the new level of information leads to monitoring, and ultimately to micro-management. This way, transparency backfires, creating vulnerability instead of opportunity.

The all important metrics of Agile are story points and velocity, and Jira acts as the manifestation of the resulting bureaucratic overhead: have a ticket for each and everything to make every engineer’s performance visible.

By making the “tech” visible for non-technical people, it enables those to gain a sort of managerial control over territory that they could not exercise before.

Built on top of this are forced commitments without having the authority to actually make them happen. Principles like team empowerment, leading by OKRs, and service leadership thus become lip-services, while micro-management rules.

IV. Technology

Agile fails to deliver–as promised by the Agile Manifesto–an engineering driven development. Decisions are still business-driven, made by people without an understanding of technology. That includes in most cases the product owner as well as the middle management, or business analysts.

Agile also makes technical debt inevitable, as teams need to deliver each sprint, preferable in a way that commitment matches velocity to make planning and risk mitigation easier for the management.

V. Teamwork

There is no room for the individual in Agile. It does not respect seniority and personal growth of the individual engineer, as there are no longer tech leads.

Instead of “individuals & interactions over processes & tools”, Agile turns individual developers again into cogs of the machinery, making the disposable clones within a more or less anonymous process. Which is also the rationale behind shuffling team-members around upon short notice.

All this contributes to a loss of ownership: a project is just a list of tasks provided by business-people, split among consecutive time-boxes, aka sprints or iterations. Which is the reason why projects become hard to be passionate about.

Despite all loss of ownership, team members are nevertheless expected to participate actively in ceremonies: from time-consuming standups and backlog refinements to retrospectives, a sprint-based "self-improvement" ritual.

What Are Your Thoughts?

Can you teach an old (management) dog, socialized in a command & control environment, new agile tricks?

To begin with, I believe there is Catch 22: a "good manager" by traditional standards is defined by knowing what to do and how to solve a problem.

Now, what if a new idea requires precisely the opposite by admitting she doesn't know? What if it is about embracing learning, experimentation and failure and empowering teams to figure out the solution, but not to deliver it yourself?

So, are we stuck in cargo cult agile for all eternity? Or will it pass all other like other management fads, too? Or shall will we be able to turn the ship around?

Personally, I still believe in George S. Patton's "Don't tell people how to do things, tell them what to do and let them surprise you with their results." approach.

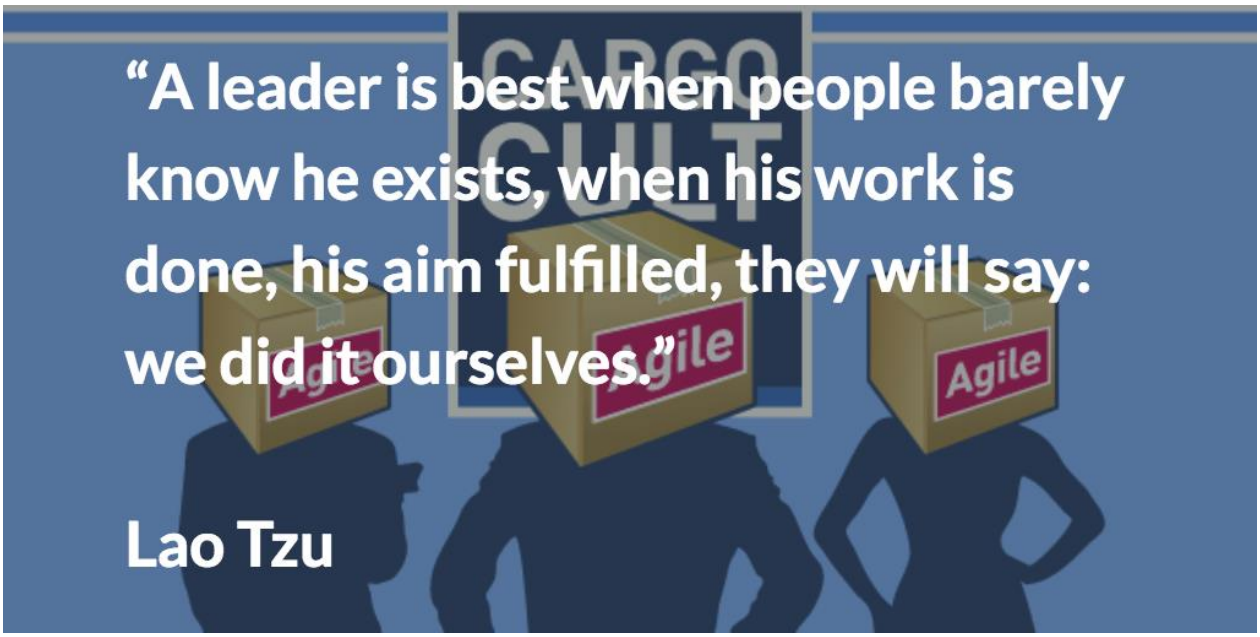
Further Reading

If you like to dig deeper into the issues why engineers despise Agile, I recommend the following posts and videos as a starting point:

1. softwarebypaul: Run Away From Agile
2. Andy Hunt: The Failure of Agile
3. Michael O. Church: Why "Agile" and especially Scrum are terrible
4. ayasin: Agile Is The New Waterfall
5. Harry Harrold, Rupert Redington: Agile Apocrypha and an Ad-hoc Manifesto

Why Agile Turns into Micromanagement

Agile turns into micromanagement as a result of the middle management's resistance to change. Despite better knowledge, changing an organization into a learning one, that embraces experimentation and failure is not in the best interest of everybody. Self-organizing, empowered teams often conflict with the middle management's drive to execute on personals agendas.



Agile and Its Proximity to Micromanagement

On the one side, we are getting increasingly better at understanding what management styles are actually creating value in the era of the knowledge economy. Service leadership, and self-organization as in holacracy, for example.

And we do have an idea, backed by one of the most robust findings in social science, what drives the individual knowledge worker. According to Daniel Pink, lasting motivation in the 21. century is fueled intrinsically by autonomy, mastery and purpose.

On the other side, one of the most promising approaches—agile in its various flavors–has meanwhile gained sort of a tainted reputation. It has become regularly associated less with empowerment and self-organization, but micromanagement.

The idea that agile is bordering on micromanagement is not a new one. Mike Cohn himself suggests that it's the team, that is micromanaging themselves for their own benefit. But he seems to be the only one to utilize the phrase "agile micromanagement" in such a meaning.

Mostly, when talking about agile micromanagement, practitioners refer to the dark side, to cargo cult agile:

- Developers often feel pressured to increase efficiency, deliver more output with the same headcount and be totally transparent at the same time. (Read more on this aspect in "Why Engineers Despise Agile".)
- And product managers feel reduced to providing secretarial assistance to product design committees, churning out user stories on behalf of stakeholders.

Question is, whether this fate is inevitable for organizations of any size, maturity and origin—be it tech or legacy.

Check Your Organization for Agile Micromanagement

You can download a PDF for a quick poll to check the state of agile micromanagement within your organization. Thus, you can determine whether your organization is knowledge worker friendly or not.

If you do so, I would appreciate your feedback on how this worked out for you and what kind of agile failure you would add to complete the list in your eyes.

Agile Leadership vs. Management

A leader is best when people barely know he exists, when his work is done, his aim fulfilled, they will say: we did it ourselves. (Lao Tzu)

Lao Tzu was an ancient Chinese philosopher and writer.

In his book "Drive: The Surprising Truth About What Motivates Us", Daniel Pink defines three key ingredients for individual motivation:

- **Autonomy**: the desire to direct our own lives, aka: not to be told what to do and how to do it
- **Mastery**: the urge to get better at something that matters
- **Purpose**: the yearning to do what we do in the service of something larger than ourselves.

What's remarkable is that none of the ingredients has anything to do with money, a corner office, a parking place close to the lift or the size of a desk. In other words: typical insignia of a successful corporate career have little or no impact on the motivation of knowledge workers.

Speaking of which, you may now understand why engineers and product people just hate working on features "sold" by the sales department without asking them in an effort to meet sales objectives for bonuses.

Fostering true motivation therefore requires leadership, not management: don't tell talented people how to do their jobs, but empower them to achieve their objectives on their own.

Leadership in this George S. Patton style — "Don't tell people how to do things, tell them what to do and let them surprise you with their results." — is not a new thing. It has been part of the knowledge of military organizations for a long time. (Read, for example, "The Art of Action: How Leaders Close the Gaps between Plans, Actions, and Results" by Stephen Burgery, who describes in his book the "agile leadership" principles of the nineteenth-century Prussian army to deal with the unpredictable environment of the battlefield.)

This is also increasingly understood at the CxO level of organizations. Service leadership and agile innovation are either understood as a necessity to stay in business, thus preserving rank & privileges. Or they are regarded as worthwhile experiments, that need to be undertaken anyway to preserve an innovative image. In both ways, they are supporting personal agendas.

And it is certainly a truism for those operating in the trenches.

The interesting question here is: have the top and bottom of an organization thus already outflanked the middle management on the quest for the next innovative or disruptive product or technology?

Has everyone, except the middle management, understood that the organization needs to change into a self-organizing and learning organization—embracing experimentation and failure—to survive in the era of an ever increasing pace of innovation?

A middle management that is still entrenched in its functional silos, aiming at local optimizations and seeking fulfillment in budgets, Powerpoints and micromanaging their underlings?

Agile and the Middle Management

Michael McCalla wrote in his post Conservatives, Liberals, and Agilists:

"Having thought about it at length, I have concluded that people within an organization that is undergoing an Agile adoption typically fall into one of three groups: Agilists, liberals, and conservatives. As in "real life," these groups often clash.

Conservatives are the folks who protect their turf and are loyal to a traditional culture. Conservatives are set in their ways and not interested in change. They will most likely resist Agile. Typically, people in roles such as project managers, quality assurance, and architecture are those who fall within this group. The words empowerment, self-organization, and flexibility tend to make them cringe."

To my experience, the "conservatives" are most likely to be found in the middle management. And they have plenty of (personal) reasons to resist change. Try walking in their shoes for a while:

1. The middle management usually lives to please their superiors. Mostly in the hope to be promoted themselves, once their superiors will climb the next step on the career ladder.
2. Thus, change is generally not serving their personal agendas.
3. And this is particularly true for agile change, given the risk of failure. "Agile" is a perceived as a loss of control, threatening their career goals.
4. Ask yourself: would you entrust a bunch of hoodie-wearing nerds to deliver on your career objects in time and on budget?
5. Self-organization, e.g. think Zappos and holacracy, is therefore actually a horror scenario for any career-minded middle manager: Being made obsolete by those reporting to her. (Reading tip: Here's what happened to Zappos' HR boss when the company got rid of managers and her job became obsolete.)
6. Therefore, middle managers have an incentive to treat today's knowledge workers like the farmers' sons from Iowa in 1905, who back then were working at an assembly line for the first time. (Also know as "agile Taylorism".)
7. Which is the reason that we still have to cope with silo structures within companies. With "us vs. them" thinking, resulting in local optimization efforts instead of teaming up for corporate-wide, cross-functional collaboration.
8. Moving from people management to service leadership—"it's my job to make you successful, what do you need?"—is hence frightening them. It would require a totally different mindset by comparison: EQ, empathy, honesty and trust.
9. Service leadership requires the ability to embrace failure as an integral part of the innovation and learning progress. Think of Thomas Edison: "I have not failed. I've just found 10,000 ways that won't work."
10. Service leadership also requires to let go and resist the temptation to tell people how to do their job. (And enjoy the instant gratification it delivers.)
11. It also collides with the western ideal of innovation based on the Steve Jobs model: a single creative individual is pushing through disruptive technology. However, a new study out of Harvard and the London School of Economics shows that the heroic inventor Is a myth; great ideas are team efforts.
12. Agile innovation also interferes with the typical "we know what we need to build for our customers" attitude of the middle management. A "good manager" knows what to do and how to solve a problem. She is not "weak" by admitting that she has to

guess herself. That she has to observe, to ask, and to include the customer in the product discovery, improvement and delivery process.

Conclusion

From the conservative middle manager's point of view, there are many reasons why sticking to a command and control management style looks personally beneficial. Hence there is a temptation to adapt any agile transition to the requirements of her very own local optimum, resulting in agile micromanagement. Which ultimately contradicts the purpose of empowering self-organizing team at an organizational level.

In my understanding, knowledge workers therefore don't despise agile per se. They just despise what managers turn the idea of self-organizing teams into in everyday operations. Because they aren't leaders. Because they put their own personal agendas above the organization's future success. Because they fail to contribute to evolve an organization's culture to embrace experiments and failure by providing autonomy, mastery and purpose to their people.

Software is eating the world. Every company with a bright future is nowadays also a software company. All are depending to a great deal on the input from their knowledge workers in an ever increasing competition. Think Google, Facebook, Uber, Tesla, and Amazon. The majority of them is leadership-driven. Even Microsoft seems to be no longer stuck in the past, pursuing a management concept that worked well in the heydays of General Motors and Taylorism.

Today's management-driven organizations, that fail to recognize this development, will make themselves obsolete over time. They will fail to compete with these leadership-driven organizations at product-level. And they will suffer a really unhealthy disadvantage in the war for talent.

So, the question remains: how do we—the knowledge workers—free yourselves from old-school management practices? From managers that are stuck in their socialization by command & control practices?

Is quitting a management-driven and joining a leadership-driven organization the only solution? (As the saying goes: there is no project interesting enough that you just couldn't walk away from it.) Or can those managers be taught new tricks, can they change their mindset and become leaders instead?

13 Signs of a Toxic Team Culture

Introduction

What looked like a good idea back in the 1990ies—outsourcing, for example, software development as a non-essential business area—has meanwhile massively backfired for a lot of legacy organizations. And yet, they still do not understand what it takes to build a decent product/engineering culture.

Learn more about typical anti-patterns and are signs that the organization has a toxic team culture.

Team Building: Internals vs. Externals

20 years ago, many large companies tagged along with Jack Welch's philosophy of outsourcing non-core business areas — such as software development — to third parties. Today, they find it hard to compete in the war for product and engineering talent with the GAFAs and other agile and technology-focused organizations. Software is finally eating the world.

The lack of a product/engineering culture in those legacy organizations usually results in hiring numerous contractors and freelancers to get at least some projects going. Which in

return often leads to some typical anti-patterns as the internals find it hard to team up with the externals:

No equality: There is a pecking order among team members. This order is not based on an individual's contribution or capability but whether that person is on pay-role or not.

Externals to the shop floor: Externals are expected to deliver the work items. Accepting accountability and developing a sense of product ownership is regarded impeding this purpose.

Career issues: Internals focus on advancing their careers by other means than building an outstanding product, for example, by getting involved in the organization's politics game.

Hiring minions: Internals claim the final say whom to hire and tend to use it to select submissive minions. (As the saying goes: B people hire C people.)

Lonesome decisions: Internals consider themselves responsible for the product and hence insist on making all decisions themselves—often single-handedly without involving the team or by overriding the team's decision.

Assigning tasks: Internals dispatch work items to either externals or juniors. (It is even worse when externals accept the situation and ask internals what is the next work item for them is.)

No WiFi for you: Externals are excluded from the use of 'internal' infrastructure, for example, WiFi and calendar applications.

Equality and Diversity

And then there are other issues beyond the internal vs. external question that might prevent a group of people that happen to be in the same place at the same time from becoming a team:

Not all developers are created equal: Merges need to be requested and are utilized as code quality stage-gates beyond a reasonable level.

Outsourcing to juniors: The senior team members consider writing tests, fixing bugs or documentation as a minor task below their pay-grade. Hence, it is outsourced to the junior team members.

Remote minions: Remote team members are not fully included, for example, they never meet the co-located team members in person.

The silent treatment: Team members are deliberately ignoring communication only to point at a later stage at a perceived lack of communication.

No diversity: The team members basically look the same, probably white dudes in their twenties and thirties.

Voting with their feet: The team suffers from a high fluctuation among its members.

Conclusion

What is difficult to understand is that legacy organizations complain that they cannot hire top engineering talent. On the other side, they do not invest in making the company a great place to work for in the first place. And by "great" I am not referring to sushi chefs on the premise or sparkling water from eight different countries in the fridge.

Creating balanced, diverse teams where rank does not have privileges—that are ready and willing to accept accountability—is essential for organizations striving to build a product/engineering culture or even become agile.

Their return on investment will largely depend on achieving this goal as early as possible in the transition.

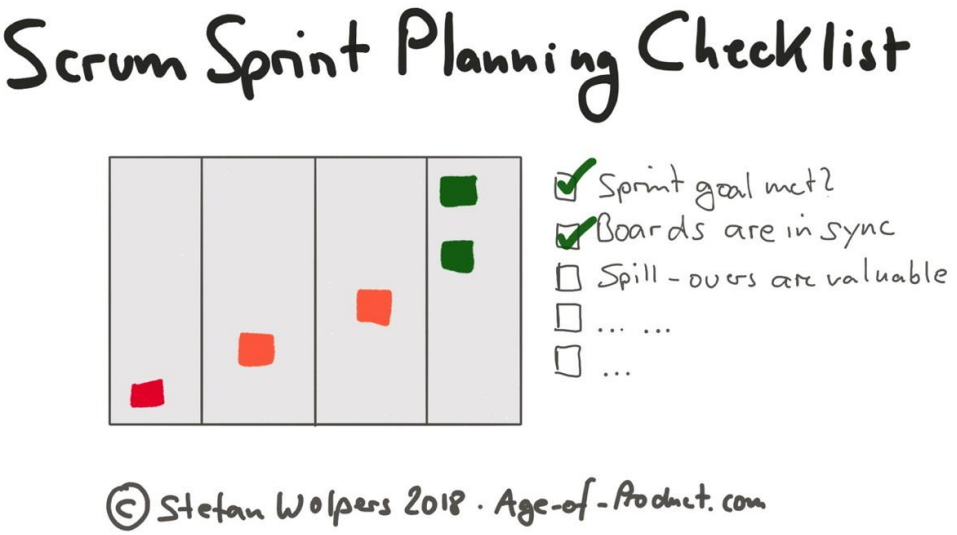
Addendum

Checklists

Scrum Sprint Planning Checklist

A sprint planning checklist? How dare you: Scrum is a mindset, not a methodology. It is a journey, not a destination. There is no one-size-fits-all-and what else could you possibly cover with a checklist, the mother of all standardized processes?

Well, it always depends on the purpose of a tool's application. Read more why scrum checklists are a handy tool if applied at an operational, hands-on level, reducing your cognitive load and freeing up time for more relevant things.



The Magic of Checklists

Some of you may be aware that checklists originate from an aviation accident. A new plane with a crew of experienced test-pilots crashed during take-off. It turned out that plane had no mechanical problems at all, the flight crew just forgot a simple step during the take-off procedure.

Probably, it was over-confidence meeting complexity, a feeling of "we know what we have to do, as we do it all the time" that led to the mistake. No matter what it was in the end, the

consequence was to make checklists mandatory in aviation. Or in hospitals. Or anywhere, where the complexity of a task at hands may prove to be too high of a cognitive load to trust everything will go smoothly.

So, from my point of view, checklists are not an evil means of imposing standardized processes but a helpful tool for the practitioner even if he or she is a scrum master using a sprint planning checklist.

Scrum Sprint Planning Checklist - The Details

This sprint planning checklist is tailored to the way my current team is working. In other words, you will likely not be able to apply this checklist to your team without modification. For example, we put a lot of effort into preparing the sprint planning during the weekly product backlog refinement sessions. (We typically plan two to three sprints.)

Practically, the sprint planning itself is a kind of confirmation of what we already decided on during last backlog refinement. We probably adjust the scope of the sprint backlog during the capacity planning. However, it rarely happens that we switch the sprint goal, for example. A typical sprint planning #1 hence takes only between 30 and 60 minutes.

Therefore, if you are working more traditionally, the following sprint planning checklist will lack some steps.

Regarding the following timeline, T=0 refers to the start-date of the upcoming sprint, and T-1 is the day before the start of this sprint. Moreover, T+1 is the day after the sprint planning:

Preparing the Sprint Planning:

- ☐ T-2: Address the number of open tickets in the code review & ready for acceptance columns. Ask the team members to focus on moving tickets to done before starting work on new tickets.
- ☐ T-1: Ask the team members to update the boards.
- ☐ T-1: Run the sprint review.
- ☐ T-1: Run the sprint retrospective.

During the Sprint Planning:

- ☐ T=0: Kick-off the sprint planning by sharing a Zoom session for those team members who cannot participate in the sprint planning in person.
- ☐ T=0: Are all team members present to run the sprint planning? (The absence of the product owner might be a challenge.)
- ☐ T=0: Clean up the old board(s) with the whole team by checking statuses of each ticket and moving tickets if necessary. Sync the offline boards with the Jira board. (A questions to answer in advance: which board is the leading one? If some team members work remotely during sprint planning, choose the online board.)
- ☐ T=0: Discuss possible spill-overs: Are those still valuable to be continued? (Spill-overs are a suitable team metric and a good topic for a retrospective. If spill-overs persist over several sprints this could trigger various discussions, for example:
    - ☐ Is the sizing of a user story or ticket right?
    - ☐ Is the quality of user stories or tickets matching the definition of ready?
    - ☐ Would Kanban be better suitable for the team?
    - ☐ If user stories or tickets that are not yet done shall not spill-over to the upcoming sprint move them to the product backlog or delete/archive them.
- ☐ T=0: If not yet available, create a new "sprint" in the team's online tool, for example, Jira.

- ☐ T=0: Close the previous sprint:
    - ☐ Did we meet the sprint goal?
    - ☐ If you use an online tool, make sure that you move all tasks that spill-over into the right buckets, for example, the upcoming sprint or the product backlog.
    - ☐ Clear the physical boards off old stickies of the previous sprint.
- ☐ T=0: Kick-off the next sprint planning:
    - ☐ Figure out the available sprint capacity of the team: who can contribute work over the course of the next sprint?
    - ☐ Ask the product owner to define the sprint goal.
    - ☐ Match capacity with the sprint goal of the product owner: Is this realistic?
    - ☐ If the sprint goal and team capacity do not match, try to strip down the scope of the sprint: Can the team deliver a smaller version of the sprint goal?
    - ☐ If the team cannot deliver the proposed sprint goal, ask the product owner to come up with a realistic sprint goal.
    - ☐ Let the team pick the user stories, and other tasks to meet the sprint goal.
    - ☐ Ask the team whether all user stories and other tasks needed to meet the sprint goal meet the team's definition of ready. (If this is not the case, make sure the issue is addressed, and the team agrees how to deal with the issue collaboratively.)
    - ☐ Ask the team whether the scope of work leaves slack time to address unexpected issues.
    - ☐ Ask the team whether the scope of work provides the capacity to tackle technical debt as well as bugs to keep technical debt at bay.
    - ☐ Create stickies with user stories and tasks for the physical boards. (Make sure that the color codes are followed; spikes, user stories, technical task, sub-tasks, and bug all have distinctly different colors.)

- ☐ T=0: Run an anonymous sprint survey regarding the previous sprint.
- ☐ T=0: Summarize the results from the retrospective and update the board with the action items.
- ☐ T=0: Summarize the results of the sprint review.

After the Sprint Planning:

- ☐ T+1: Sync the offline board(s) with the online board.
- ☐ T+1: Probably, start collecting data for the upcoming retrospective, for example, by putting up a sprint mailbox.
- ☐ T+2: Kindly remind the team members to participate in the outstanding sprint survey. The recommended minimum number of participants is eight.
- ☐ T+3: Publish the results of the sprint survey of the previous sprint.

Scrum Sprint Planning Checklist - The Conclusion

Checklists serve both the junior practitioner — what do I have to do — as well as the experienced agile practitioner to deal with the complexity at hands. Checklists like this example of a sprint planning checklist are by no means a violation of the agile manifesto but lessen the cognitive load of running ceremonies and practices.

Think of scrum checklists as work in progress that needs to be revisited and adjusted regularly. In that respect, scrum checklists are not much different from, for example, working agreements or a definition of done.

About the Author

Stefan has worked many years as a product manager, product owner, and agile coach (Scrum, LeSS, Lean Startup, Lean Change). He's founded multiple companies and has led the development of B2C and B2B software, primarily for startups, but also for other organizations — including a former Google subsidiary. He is a steward of the XSCALE Alliance and an XBA Exponential Business Agility Coach (XBAC).

Stefan curates 'Food for agile Thought' — the largest weekly newsletter on agile product development with more than 15,000 subscribers. He also hosts the largest global Slack community for agile people — 'Hands-on Agile' — with more than 2,700 members. (As of March 2018.)

Despite initially studying chemistry Stefan has never worked in a laboratory, and instead continued his education in business administration and law. Following school, he discovered a passion for software and, in 1996, launched the first online e-commerce platform to feature SAP R/3 connectivity — only to learn that the early bird does not necessarily catch the worm. After moving from his hometown of Hamburg to Berlin, Germany, Stefan created Susuh GmbH, a marketplace for local services. Other ventures followed, and in 2011 he founded Startup Camp Berlin — one of the largest German startup conferences today.

Stefan's latest project, Age of Product, focuses on the exchange of knowledge between the people involved in product development: product managers, product owners, Scrum Masters, designers, and developers. The goal is to help those involved in product development with lessons learned and best practices for continuous agile product discovery and delivery.

Read more about Stefan at LeSS Works or Scrum Alliance, and connect with him via LinkedIn, or Twitter, or privately via email.