The Scrum Anti-Patterns Guide— A Hands-on Manual from the Trenches

by Stefan Wolpers

Version: 5.70 2023-09-23

Table of Content

Preface

Welcome to my hands-on guide on scrum anti-patterns, detailing over 160 anti-patterns that you might observe in practice.

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 the list of readers of this ebook.

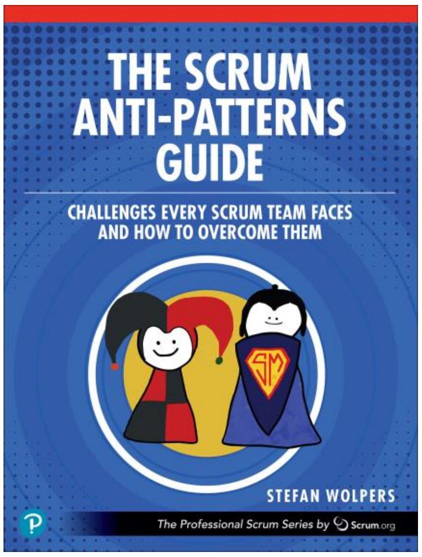
Thank you for your understanding!

Best,
Stefan

The Scrum Anti-Patterns Guide is Available for Preorder

It has been a lot of work. Finally, the Scrum Anti-Patterns Guide is available for preorder on Amazon: As of September 23, 2023, the book will be available on February 25, 2024. (Expect Amazon.com; they are lagging.)

Use the following link to be directed to your local Amazon store: https://geni.us/wtS2aa. (Includes affiliate link — support the effort at no cost to you.)


THE SCRUM ANTI-PATTERNS GUIDE
CHALLENGES EVERY SCRUM TEAM FACES AND HOW TO OVERCOME THEM
SM
STEFAN WOLPERS
The Professional Scrum Series by Scrum.org

Scrum Mastery in 300 Words

How to make Scrum work? Read on to learn more about my top three objectives for Scrum Masters striving to achieve Scrum Mastery.

The Top Three Objectives

Achieving Scrum Mastery is no rocket-science: Make sure that the Scrum Team delivers a valuable, potentially shippable Product Increment every single Sprint with the precision of a Swiss clockwork. Delivering at this level builds a happy customer base, trust within the organization as well as high morale among the team members. To do so, focus your activities on three objectives:

1. Defend the **Product Backlog** tooth and nail to ensure it represents the best possible use of the Development Team's work from a customer value perspective at any given moment—garbage in, garbage out. In other words, your Scrum Team's Product Backlog has to be actionable 24/7. By my standards for Scrum Mastery, that means that you need to be capable of running a meaningful Sprint Planning instantly.

2. Keep **technical debt** at bay from day #1: Make sure that the Development Team members are embracing Xtreme Programming techniques from TDD, pair programming, relentless refactoring to supporting an emergent architecture from the start. Also, fight for their Slack time—at least 20% of their theoretical capacity—uncompromisingly. Pay serious attention to the concept of 'Done,' as represented in the Definition of Done.

3. **Support the middle management** by educating them on how to become servant leaders, thus alleviating their fear of obsolescence. If the ability to pay for a mortgage is no longer an issue, personal agendas, managers might harbor, will be overcome, and we can address the necessary change within the organization collaboratively.

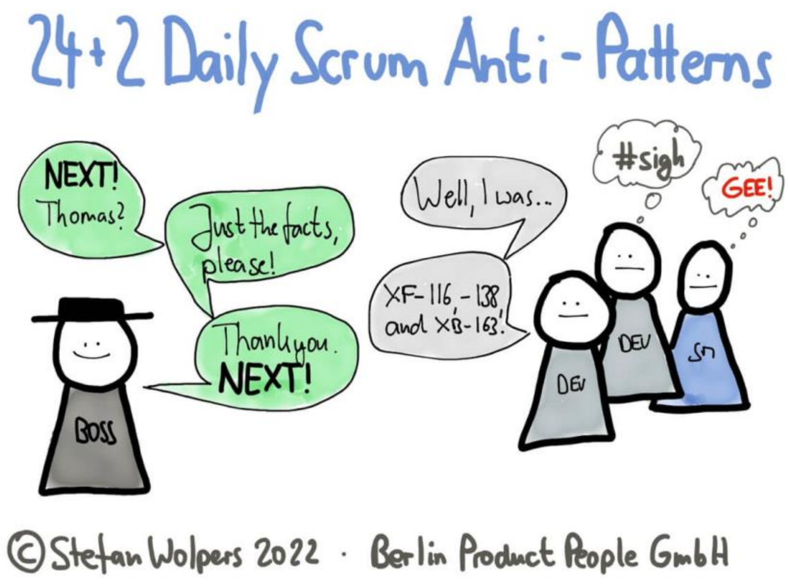
Of course, as so often, the devil is in the details. Some 160 of those devils we will address in this ebook.

Scrum Event Anti-Patterns

24 Daily Scrum Anti-Patterns

The Daily Scrum

In my experience, the Daily Scrum is the Scrum event with the highest anti-pattern density among all events. Learn more about Daily Scrum anti-patterns that threaten your Scrum team's success, from becoming a reporting session to assignments to answering these three questions.



The Purpose of the Daily Scrum

The purpose of the Daily Scrum is clearly described in the Scrum Guide — no guessing is necessary:

The purpose of the Daily Scrum is to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary, adjusting the upcoming planned work.

The Daily Scrum is a 15-minute event for the Developers of the Scrum Team. To reduce complexity, it is held at the same time and place every working day of the Sprint. If the Product Owner or Scrum Master are actively working on items in the Sprint Backlog, they participate as Developers.

The Developers can select whatever structure and techniques they want, as long as their Daily Scrum focuses on progress toward the Sprint Goal and produces an actionable plan for the next day of work. This creates focus and improves self-management.

Daily Scrums improve communications, identify impediments, promote quick decision-making, and consequently eliminate the need for other meetings.

The Daily Scrum is not the only time Developers are allowed to adjust their plan. They often meet throughout the day for more detailed discussions about adapting or re-planning the rest of the Sprint's work.

Source: Scrum Guide 2020.

The Daily Scrum is an essential event for inspection and adaption, run by the Developers, and guiding it for the next 24 hours on its path to achieving the Sprint Goal. The Daily Scrum is hence the shortest planning horizon in Scrum and thus highly effective to guide the Scrum team's efforts.

Contrary to popular belief, its 15-minutes timebox is not intended to solve all the issues addressed during the Daily Scrum. It is about creating transparency, thus triggering the inspection. If an adaption of the plan or the Sprint Backlog, for example, is required, the Developers are free to handle the resulting issues at any time. In my experience, most Daily Scrum anti-patterns result from a misunderstanding of this core principle.

Daily Scrum Anti-Patterns

Typically, a good Scrum Team won't need more than 10 to 15 minutes to inspect its progress towards the Sprint Goal. Given this short period, it is interesting to observe that the Daily Scrum is often so riddled with anti-patterns. The anti-patterns often cover a broad spectrum, ranging from behaviors driven by dysfunctional Scrum teams to apparent failures at an organizational level.

My unprioritized list of notorious Daily Scrum anti-patterns is as follows:

Daily Scrum Anti-Patterns of the Scrum Team

- **Orientation lost:** The Daily Scrum serves one purpose as it answers a simple question: Are we still on track to meet the Sprint Goal? Or do we need to adapt the plan or the Sprint Backlog or both? Sometimes though, the Developers cannot answer that question immediately. (In that respect, visualizing the progress towards the Sprint Goal is useful. Removing the Developers' task of maintaining a mandatory burndown chart from the Scrum Guide a few years ago does not imply that such a visualization is useless.)

- **Status report:** The Daily Scrum is a status report meeting, and Developers are waiting in line to "report" progress to the Scrum Master, the Product Owner, or maybe even a stakeholder. (The “three Daily Scrum questions” often serve as a template for this anti-pattern.)

- **No routine:** The Daily Scrum 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 the Daily Scrum. Think of it as a spontaneous drill: don’t put too much thought into the Daily Scrum, just do it. Skipping Daily Scrums can turn out to be a slippery slope: if you skip one Daily Scrums or two, why not skip every second one?)

- **Disrespect I:** Other team members are talking while someone is sharing their progress with the Developers. (The use of talking tokens among adults to avoid this behavior does not qualify as a solution in my eyes.)

- **Disrespect II:** Team members are late to the Daily Scrum or do not show up at all. (This failure poses a massive risk for the Developers as they are inspecting and probably adapting their plan to accomplish the Sprint Goal based on incomplete information, thus reducing the probability of achieving the Sprint Goal.)

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

- **Overcrowded:** The Daily Scrum is ineffective due to a large number of active participants. (There is a reason why the Scrum Guide recommends to limit the number of team members to ten.)

Anti-Patterns of the Developers

- **Cluelessness:** Developers are not prepared for the Daily Scrum. (“I was doing some stuff, but I cannot remember what. Was important, though.”)

- **Planning meeting:** The Developers hijack the Daily Scrum to discuss new requirements, refine user stories, or have a sort of (Sprint) Planning Meeting, probably collaborating with the Product Owner.

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

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

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

- **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.”)

- **No impediments:** No one reports any obstacles; no one needs any help or support from their teammates. (Congratulations on your seemingly well-oiled machine! However,
maybe, Developers do not feel safe to address issues, challenges, or problems?)

Daily Scrum Anti-Patterns of the Product Owner

- **Assignments:** The Product Owner assigns tasks directly to team members. (The Developers self-organize their work: "No one else tells them how to turn Product Backlog items into Increments of value." **Source**: Scrum Guide 2020.)

- **Additional work:** The Product Owner or even other stakeholders attempt to introduce new work to the current Sprint during the Daily Scrum. (This behavior may be acceptable for high priority bugs, although the Developers should be aware of those before the Daily Scrum. Otherwise, the composition of the Sprint Backlog is the sole responsibility of the Developers: they may accept new work during the Sprint; however, that is their sole decision.)

Anti-Patterns of the Scrum Master

- **Daily Scrum enforcer:** The Scrum Master is enforcing the Daily Scrum. (It is the task of the Scrum Master to ensure "[…] that all Scrum events take place and are positive, productive, and kept within the timebox." (Source.) However, if none of the Developers believes that Daily Scrum is a sound investment, the Scrum Master cannot simply compensate for this lack of intrinsic motivation by using the Scrum Guide as a forcing function. They need to help to create the motivation on the side of the Developers to run the Daily Scrum in the first place. Enforcing the Daily Scrum is a common Taylorism artifact where trust in the Developers' capability to self-organize is missing.)

- **Not supporting struggling Developers:** A Developer experiences difficulties in accomplishing an issue over several consecutive days and nobody is offering help. Moreover, the Scrum Master fails to facilitate the necessary discussion. (Often, this result is a sign that people either may not trust each other or do not care for each other. Alternatively, the workload of the Developers has reached an unproductive level as they no longer can support each other. The use of 'work item age' has proven to be a helpful practice.)

- **Not preventing stakeholders from attendance:** Although the Developers decided to limit attendance to the Daily Scrum to the Scrum team members, stakeholders show up uninvited, and the Scrum Master is not addressing the issue. (It is the Developers' prerogative to decide how to run the Daily Scrum. If they choose to keep stakeholders away from it, this decision needs to be respected by everyone else. If stakeholders fail to do so, the Scrum Master needs to step in.)

- **Not enforcing the time-box:** The Developers extend the Daily Scrum regularly beyond the 15-minute time-box. (This is unfortunate, as the extension of the time-box invites problem-solving and other activities into the Daily Scrum, thus distracting from answering the critical question: are we still on track to accomplish the Sprint Goal?)

Daily Scrum Anti-Patterns of the Stakeholders

- **Command & control by the management:** Line managers attend the Daily Scrum to gather "performance data" on individual team members. (This behavior is defying the very purpose of self-managing Scrum teams.)

- **"A word, please":** Line managers are waiting until the Daily Scrum is over and then reach out to individual Developers for specific reporting from them. (Nice try. However, this hack is also unwanted behavior and distracts the Developers.)

- **Talkative chickens:** "Chickens" actively participate in the Daily Scrum. (Stakeholders are supposed to listen in but not distract the Developers members during their inspection.)

- **Communicating via body language:** Formally, stakeholders remain silent. However, they participate in the Daily Scrum via their body language. (Again, stakeholders are supposed to listen in but not distract the Developers. Yet rolling eyes and face-palming are as powerful as the spoken word. Moreover, even subtle forms of body language represent communication, as one cannot "not communicate." Furthermore, some stakeholders may have a naturally intimidating presence that can prove harmful to the communication among the Developers.)

Food for Thought

There are some issues that are worthwhile considering as a Scrum Team:

Synchronous and asynchronous Daily Scrum events: Some teams like to have their Daily Scrum in Slack, particularly those not co-located. But, of course, using Slack does not manifest an anti-pattern per se; it is the prerogative of the Developers to run the Daily Scrum in any fashion that serves its purpose: inspect the plan for the next 24 hours to meet the Sprint Goal. I was even working with a co-located Scrum team—sitting around a large table—that used a synchronous Slack session as their preferred way of having their Daily Scrum. It worked well. But what about an asynchronous Daily Scrum in Slack?

Substituting the Daily Scrum: What about the idea to skip the Daily Scrum altogether, as the Developers are heavily using more or less automated messaging systems on most aspects of their daily work, while they can handle the rest during other events, for example, pairing or mobbing sessions? Is it beneficial to allocate 15 minutes every day to the Daily Scrum under these circumstances? Or is that mere dogmatism?

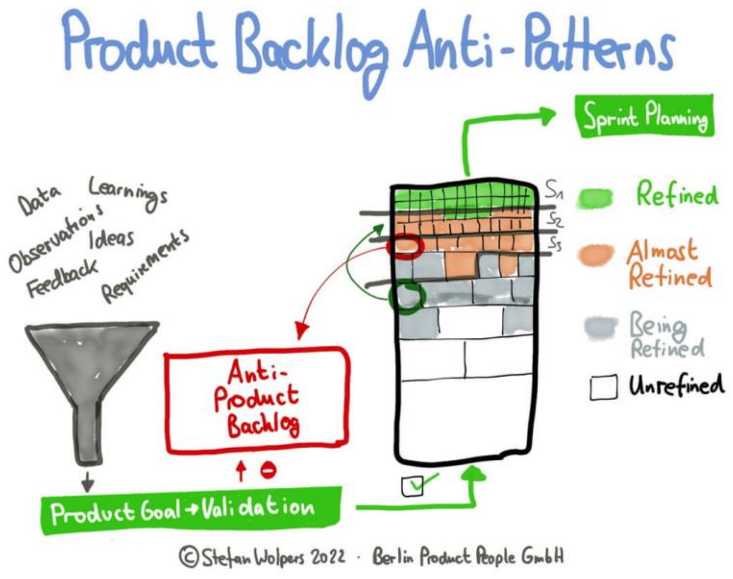
Conclusion

Given the importance of the Daily Scrum for the success of the Scrum Team's effort of achieving the Sprint Goal, its anti-patterns density is no surprise. Unfortunately, people seem to be either ignorant (or at least less well educated) about its purpose. Or they — intentionally or not — interfere with the Developers' self-organization. One way or another, it is a crucial responsibility of the Scrum Master to help all participants overcome typical Daily Scrum anti-patterns.

27 Product Backlog and Refinement Anti-Patterns

The Product Backlog

Scrum is a tactical framework to build products, provided you identify what is worth making in advance. But even after a successful product discovery phase, you may struggle to create the right thing in the right way if your Product Backlog is not up to the job—garbage in, garbage out. The following article points to 27 common Product Backlog anti-patterns – including the Product Backlog refinement process – limiting your Scrum team's success.



The Product Backlog According to the Scrum Guide

First of all, let's have a look at the current edition of the Scrum Guide on the purpose of the Product Backlog:

"The Product Backlog is an emergent, ordered list of what is needed to improve the product. It is the single source of work undertaken by the Scrum Team.

Product Backlog items that can be Done by the Scrum Team within one Sprint are deemed ready for selection in a Sprint Planning event. They usually acquire this degree of transparency after refining activities. Product Backlog refinement is the act of breaking down and further defining Product Backlog items into smaller more precise items. This is an ongoing activity to add details, such as a description, order, and size. Attributes often vary with the domain of work.

The Developers who will be doing the work are responsible for the sizing. The Product Owner may influence the Developers by helping them understand and select trade-offs."

Source: Scrum Guide 2020.

Scrum, being a tactical framework, is good at effectively getting validated hypotheses into the hands of the customers to close the learning loop, allowing for the inspection of the initial assumption and decision process to build an Increment and subsequently adapting the preparations for the next Sprint.

Scrum is not good at formulating hypotheses and running experiments, validating or falsifying them, otherwise referred to as product discovery. That is not tactics but part of operations, positioning the Scrum team for success. If you look at the flow diagram above, Scrum's part—tactics—is on the right side, while the product discovery part—operations—is on the left. There are ample opportunities to deal with the left part of the process, for example, Lean Startup, Design Thinking, Design Sprint, Lean UX, Dual-Track Agile, just to name a few.

At any given time, a Scrum team needs to practice product discovery and product delivery (or product development) simultaneously and continuously. However, this necessity does not mandate squeezing all work items into the Product Backlog. On the contrary, in my experience, Scrum teams perform below their capabilities when they do not strictly adhere to maintaining two different artifacts for both parts of the process:

1. There shall be a transparent artifact to keep track of ideas, hypotheses, experiments, and outcomes. (I like to refer to a part of that artifact as the Anti-Product Backlog, a living repository of issues that a Scrum team decides not to pursue.)

2. The second artifact is, of course, the Product Backlog. An effective Scrum team puts a lot of effort into keeping this artifact "actionable." An actionable Product Backlog reflects the continuous effort of the Scrum team in product discovery in general—developing an informed opinion of which idea may be valuable to the customers—and refining those validated hypotheses into proper Product Backlog items.

   At this point, everyone on the Scrum team understands why the team pursues this opportunity over others, what the Developers shall build, how they shall accomplish this, and, probably already at this stage, who will work on it. Typically, the size of an action Product Backlog comprises 3-6 Sprints of work, and given that the Product Owner orders Product Backlog items by value, the Scrum team also has a good understanding of when they might work on this.

Common Product Backlog Anti-Patterns

Despite being relatively straightforward, the process of creating and refining a Product Backlog often suffers from various anti-patterns. I have identified five different categories for Product Backlog anti-patterns:

General Product Backlog Anti-Patterns

1. **Prioritization by proxy:** A single stakeholder or a committee of stakeholders prioritizes the Product Backlog. (The strength of Scrum is building on the solid position of

the Product Owner. The Product Owner is the only person to decide what tasks become Product Backlog items. Hence, the Product Owner also decides on ordering the Product Backlog. Take away that empowerment, and Scrum turns into a pretty powerful waterfall 2.0 process.)

2. **Over-sized:** The Product Backlog contains more items than the Scrum team can deliver within three to six sprints. (This practice very likely will result in wasted efforts: You will refine work items that the Developers will never turn into Increments. Moreover, by investing all the efforts upfront, you may fall victim to the sunk cost fallacy, delivering less value than possible.)

3. **Outdated issues:** The Product Backlog contains items that haven't been touched for several weeks or more. (That is typically the length of three 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.)

4. **Everything is detailed and estimated:** All Product Backlog items are entirely detailed and estimated. (That is too much upfront work and bears the risk of misallocating the Scrum team's time. The refinement of Product Backlog items is a continuous effort only to the point where the Scrum team feels comfortable turning these items into Increments.)

5. **INVEST?** The Scrum team does not apply the INVEST principle to Product Backlog items. (Failing to create independent, valuable, and small work items increases the risk of implementation failure during the Sprint. Therefore, a Scrum team is well-advised to invest—no pun intended—in a proper Product Backlog refinement practice to address possible problems before they start working.)

6. **Component-based items:** The Product Backlog items are sliced horizontally based on components instead of vertically based on end-to-end functionality. (This may be either caused by your organizational structure, an effect called Conway's law. In this case, overcome this organizational debt by moving to cross-functional teams to improve the Scrum team's delivery ability. Otherwise, the Scrum team should invest in a workshop on writing better Product Backlog items.)

7. **Missing acceptance criteria:** There are Product Backlog items that need additional acceptance criteria without listing them. (It is unnecessary to have all acceptance criteria ready at the beginning of the refinement cycle, although they would make the task much more accessible. However, all Product Backlog items need to meet the Definition of Done and—probably—specific requirements at the work item level. If the Definition of Done does not provide the latter ones, the now necessary acceptance criteria shall result from the refinement process.)

8. **Too detailed acceptance criteria:** There are Product Backlog items with an extensive list of acceptance criteria. (This is the other extreme: the Product Owner covers each edge case without negotiating with the Developers. Typically, three to five acceptance

criteria are more than sufficient. If you believe that you need more criteria, this may indicate that the work item is still too large and needs splitting.)

9. **100% in advance:** The Scrum team creates a Product Backlog covering the complete project or product upfront because the release scope is believed to be limited. (Let me ask a question: How can you be sure to know today what to deliver six months from now when you work on a complex, adaptive problem? Isn't not being able to predict the future the reason that you employ Scrum in the first place?)

10. **No more than a title:** The Product Backlog contains items that comprise little more than a title. (Creating noise by adding numerous "reminders" to the Backlog, thus obfuscating the signal it shall provide, will diminish the Scrum team's capability to create valuable Increments for the customers. Ideas do not belong in the Product Backlog; they are part of the product discovery system.)

11. **No research:** The Product Backlog contains few to no spikes or time-boxed research tasks. (This effect often correlates with a Scrum team spending too much time discussing future problems instead of researching them with a spike as part of a continuous Product Backlog refinement process.)

12. **Last minute Product Backlog**: The Product Owner and the rest of the Scrum team do not invest adequately in Product Backlog creation and refinement. Consequently, they rush to fill the Product Backlog immediately before the Sprint Planning to ensure there is enough work for the upcoming Sprint. (Garbage in, garbage out. This approach points to a fundamental misunderstanding or ignorance regarding the purpose of Scrum: when to use it, and what Scrum's critical success factors creating value for everyone involved are.)

Product Backlog Anti-Patterns of the Product Owner

1. **Storage for ideas**: The Product Owner uses the Product Backlog as a repository of ideas and requirements. (A large Product Backlog is probably considered a sign of a "good" Scrum team: You are fully transparent, and this is proof of your usefulness to the organization. However, being "busy" does not equal value to customers and the organization. The additional noise created by the sheer number of issues may also cloud the detection of valuable items. Lastly, the Product Backlog size may have a crowding-out effect on stakeholders as they feel overwhelmed. The idea-inflated Product Backlog may impede critical communication with them as a consequence.)

2. **Part-time PO**: The Product Owner is not working daily on the product backlog. (The Product Backlog needs to represent the best use of the Developers' time at any given time. Suppose an update to the Product Backlog happens only occasionally, for example, shortly before the next Sprint Planning. Due to a lack of refinement, it is likely to leave a lot of value on the table. The value of the Product Backlog results in a significant part from the open discussion between the Product Owner and the Developers. In a good Scrum team, the Developers constantly challenge the Product Owner regarding

the value of suggested Product Backlog items. This checks & balances approach reduces the probability of the Product Owner falling victim to confirmation bias, thus mitigating the risk that the Scrum team makes a wrong investment decision in the next Sprint Planning. As the saying goes: Love the customers' problem, not your solution.)

3. **Copy & paste PO**: The Product Owner creates Product Backlog items 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: Refining Product Backlog items is a team exercise. Moreover, using practices like user stories templates helps everyone understand the Why, the What, and the How. Remember Karl Popper: "Always remember that it is impossible to speak in such a way that you cannot be misunderstood.")

4. **Dominant PO**: The Product Owner creates Product Backlog items by providing not just the 'Why' but also the 'How' and the 'What.' (Just stick with the Scrum Guide and its built-in checks & balances: The Developers answer the 'How' question–the technical implementation–, and both the team and the Product Owner collaborate on the 'What' question: What scope is necessary to achieve the desired purpose?)

5. **User story author**: The Product Owner invests too much time upfront in creating Product Backlog items, making them too detailed. (If a work item looks complete, the Developers might not see the necessity to get involved in further refinement. This way, a "fat" Product Backlog item reduces the team's engagement level, 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.)

6. **The '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 who is a communication gateway is a risk to the Scrum team's success. Capable Scrum teams in general and Product Owners, in particular, know well when to reach out to others to better understand a matter at hand.)

Product Backlog Anti-Patterns at Portfolio and Product Roadmap Level

1. **Roadmap?** The Product Backlog is not in sync with the product roadmap. (The Product Backlog is supposed to be detailed enough to accomplish the current Product Goal. In my experience, this medium planning target often may require up to three to fours Sprints to complete. Beyond that point, the Product Backlog should instead focus on themes from the product roadmap in broad strokes, pointing at the upcoming Product Goal but nothing already too specific. The realization risk of these themes at this point is too high. In that respect, the Product Backlog reflects a "rolling" subset of the product roadmap at any given time.)

2. **Annual roadmaps**: The organization's portfolio plan, release plans, or product roadmaps are created once a year in advance, never to be touched again in the mean-

time. (If the Product Backlog stays aligned to these plans, it probably introduces waterfall planning through the backdoor. Agile planning is always "continuous" to mitigate the risk of misallocating efforts to outdated plans. This kind of "rolling planning" also applies to product roadmaps that benefit from being revised at least every 6-12 weeks, depending on the industry.)

3. **Roadmaps secrets**: The portfolio planning, the release plan, or the product roadmap are not visible to everybody involved in creating Increments. ("If you don't know where you are going, any road will get you there." The "big picture" is crucial for any Scrum team's success. It needs to be available to everybody as product success is a team sport, dependent on every team member's and stakeholder's ideas, insights, and skills. Excluding team members from this information risks jeopardizing value creation, as they may fail to connect the dots. The same applies, for example, to restricting access to essential data related to the Scrum team's work, such as the revenue generated by the team's work, utilization of the product or service, customer information, etc.)

4. **China in your hands**: The portfolio planning, the release plan, or the product roadmap are not considered achievable and believable by those supposed to deliver it. (If this is reflected in the Product Backlog, refining work items will probably be a waste.)

Product Backlog Anti-Patterns of the Developers

1. **A submissive Scrum team**: The Developers submissively follow all demands of the Product Owner. (Challenging the Product Owner whether their selection of work items is the best use of the Developers' time is the noblest obligation of every team member: Why shall we do this? Scrum does not work without everyone living up to the checks & balances built into the framework. It is easy to fall in love with "your solution" over addressing the real customer problem. If Developers simply make whatever the Product Owner "suggests," the overall value created by the Scrum team will likely be below its potential. That is why you want missionaries on your team, not just mercenaries.)

2. **What technical debt?** The Developers do not demand adequate time to tackle technical debt and bugs and preserve the quality standard of the product. Instead, they fully embrace the feature factory, shipping one new feature after the other. (My rule of thumb is that the Developers shall consider allocating up to 20 % of their time to fixing bugs and refactoring the codebase. A high-quality tech stack is at the core of being able as a Scrum team to adapt the following steps to lessons learned and recent market trends. Technical excellence is a prerequisite of any form of business agility; preserving this state is a continuous process that requires a steady and substantial investment.)

3. **No slack time**: The Developers do not demand 20% slack time—or unplanned capacity—from the Product Owner. (This overlaps with the Sprint Planning, creating Sprint Goals, and the team's ability to forecast. However, it cannot be addressed early

enough. If a team's capacity is always utilized at 100 %, its performance will decrease. Everyone will focus on getting their tasks done. There will be less time to support teammates or to pair. Minor issues will no longer be addressed immediately. And ultimately, the 'I am busy' attitude will reduce the creation of a shared understanding among all team members why they do what they are doing.)

Product Backlog Anti-Patterns of the Scrum Team

1. **Involving the Scrum team—why?** The Product Owner is not involving the entire Scrum team in the Product Backlog refinement process and instead relies on just the "lead engineer" and a designer. Moreover, the Developers are okay with this approach as it allows them to write more code which they prefer over figuring out what is worth building. (When trying to solve a complex problem, there are no experts but many competing ideas. Therefore, limiting the active participants in refinement activities to a few team members instead of the whole Scrum team increases the risk of falling victim to confirmation bias as the diversity of opinion is artificially limited.)

2. **No time for refinement**: The Scrum team does not have enough Product Backlog refinement sessions, resulting in a low-quality Product Backlog. (The Scrum Guide 2017 initially advised spending up to 10 % of the Scrum team's time on the Product Backlog refinement. This is a sound business decision: Nothing is more expensive than an ill-designed feature delivering little or no value.)

3. **Too much refinement**: The Scrum team has too many refinement sessions, resulting in a too detailed Product Backlog. (Too much refinement isn't healthy either. There is a moment when the marginal return of an additional refinement effort is zero or probably even negative—think analysis-paralysis. The only way for a Scrum team to understand whether the previous validation of the underlying hypotheses of a new feature is correct is to build and ship this thing. There is no way to figure this out at the green table, refining and discussing the issue endlessly.)

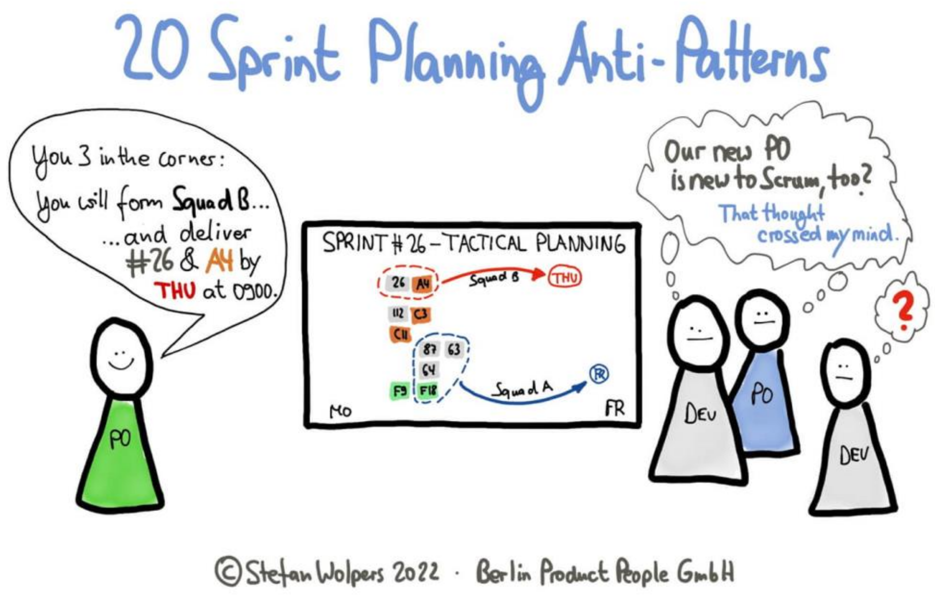
Conclusion

Even if you have successfully identified what is worth building next, your Product Backlog and its refinement process will likely provide room for improvement. Just take it to the team and address possible Product Backlog anti-patterns in the next Retrospective. In my experience, it is the easiest way to improve the Scrum team's performance and thus the team's standing among stakeholders and customers.

20 Sprint Planning Anti-Patterns

The Sprint Planning

The Sprint Planning is a core event that defines how your customers' lives will improve with the following Product Increment. Learn more on how to improve its effectiveness by avoiding 20 common Sprint Planning anti-patterns.


20 Sprint Planning Anti-Patterns
You 3 in the corner: You will form Squad B... ...and deliver #26 & A4 by THU at 0900.
SPRINT #26 – TACTICAL PLANNING
26 A4
Squad B
THU
112 C3
C11
87 63
C4
F3 F18
Squad A
FR
MO
FR
Our new PO is new to Scrum, too?
That thought crossed my mind.
?
PO
DEV
PO
DEV
©Stefan Wolpers 2022 · Berlin Product People GmbH

The Purpose of the Sprint Planning

Scrum's Sprint Planning aims to align the Developers and the Product Owner on what to build next, delivering the highest possible value to customers. First, the Product Owner points to the team's Product Goal and introduces the business objective of the upcoming Sprint. The Scrum Team then collaboratively creates a Sprint Goal, considering who is available and the target the team shall accomplish. Next, the Developers forecast the work required to achieve the Sprint Goal by picking the right items from the Product Backlog and transferring them to the Sprint Backlog. Also, the Developers need to create a plan on how to accomplish their forecast.

No longer mandatory since the 2020 edition of the Scrum Guide is that the Scrum team agrees to tackle at least one high-priority improvement issue from the previous Sprint Retrospective.

The Scrum Guide characterizes the Sprint Planning as follows:

Sprint Planning initiates the Sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire Scrum Team.

The Product Owner ensures that attendees are prepared to discuss the most important Product Backlog items and how they map to the Product Goal. The Scrum Team may also invite other people to attend Sprint Planning to provide advice.

Basically, the Sprint Planning answers three questions:

1. ***Why is this Sprint valuable?*** *(The Product Owner proposes how the product could increase its value and utility in the current Sprint. The whole Scrum Team then collaborates to define a Sprint Goal that communicates why the Sprint is valuable to stakeholders. The Sprint Goal must be finalized prior to the end of Sprint Planning.)*

2. ***What can be Done this Sprint?*** *(Through discussion with the Product Owner, the Developers select items from the Product Backlog to include in the current Sprint. The Scrum Team may refine these items during this process, which increases understanding and confidence. Selecting how much can be completed within a Sprint may be challenging. However, the more the Developers know about their past performance, their upcoming capacity, and their Definition of Done, the more confident they will be in their Sprint forecasts.*

3. ***How will the chosen work get done?*** *(For each selected Product Backlog item, the Developers plan the work necessary to create an Increment that meets the Definition of Done. This is often done by decomposing Product Backlog items into smaller work items of one day or less. How this is done is at the sole discretion of the Developers. No one else tells them how to turn Product Backlog items into Increments of value. The Sprint Goal, the Product Backlog items selected for the Sprint, plus the plan for delivering them are together referred to as the Sprint Backlog.)*

Source: Scrum Guide 2020.

Suppose the Scrum Team has been successfully utilizing the Product Backlog refinements to create and maintain an actionable Product Backlog. In that case, the Sprint Planning consumes much less time than the eight hours the Scrum Guides lists for a month-long Sprint. Basically, the Developers and the Product Owner adjust the previously discussed scope of the upcoming Sprint to the available capacity.

Alternatively, a valuable new task may have appeared overnight, and the Product Owner wants this task to become a part of the next Sprint, too. Consequently, some previously considered Product Backlog items won't make it into the Sprint Backlog. A good Scrum team can handle that in ten to 30 minutes before moving on to the decomposition tasks and the initial planning of how the Developers intend to accomplish the work of the Sprint.

Sprint Planning Anti-Patterns

There are mainly three categories of Sprint Planning anti-patterns. They concern the Developers, the Product Owner, and the Scrum team:

Sprint Planning Anti-Patterns of the Developers

- **Capacity?** The Developers overestimate their capacity and take on too many tasks. (The Developers should instead consider everything 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 events as practices such as Product Backlog refinement, and other meetings, to name a few.)

- **What technical debt?** The Developers do not demand adequate time to tackle technical debt and bugs and preserve the quality standard of the product. Instead, they fully embrace the feature factory, shipping one new feature after the other. (My rule of thumb is that the Developers shall consider allocating up to 20 % of their time to fixing bugs and refactoring the codebase. A high-quality tech stack is at the core of being able as a Scrum team to adapt the following steps to lessons learned and recent market trends. Technical excellence is a prerequisite of any form of business agility; preserving this state is a continuous process that requires a steady and substantial investment. **Read more**: Technical debt and Scrum.)

- **No slack time:** The Developers do not demand 20% slack time—or unplanned capacity—from the Product Owner. (This overlaps with the Sprint Planning, creating Sprint Goals, and the team's ability to forecast. However, it cannot be addressed early enough. If a team's capacity is always utilized at 100 %, its performance will decrease. Everyone will focus on getting their tasks done. There will be less time to support teammates or to pair. Minor issues will no longer be addressed immediately. And ultimately, the 'I am busy' attitude will reduce the creation of a shared understanding among all team members why they do what they are doing.)

- **Planning too detailed:** During the Sprint Planning, the Developers plan every single task of the upcoming Sprint in advance. (Don't become too granular. One-quarter of the tasks are more than sufficient to not just start with the Sprint, but also start learning. The Sprint Backlog is emergent, and doing too much planning upfront might result in waste. Scrum is not a time-boxed form of the waterfall planning practice.)

- **Too much estimating:** The Developers even estimate sub-tasks. (That looks like accounting for the sake of accounting to me. So don't waste your time on that. Remember: the purpose of estimating is to identify misalignment among the Developers regarding the What and How of items from the Product or Sprint Backlog. **Read more**: Estimates Are Useful, Just Ditch the Numbers.)

- **Too little planning:** The Developers skip planning altogether. (Skipping planning is unfortunate, as it is also an excellent opportunity to talk about how to spread knowledge among the Developers, where the architecture is heading, or whether the tools are adequate. For example, the team might also consider who will be pairing with whom on what task. The Developers' planning part is also well-suited to consider how to reduce technical debt, see above.)

- **Team leads?** The Developers do not devise a plan to deliver their forecast collaboratively. Instead, a 'team lead' does all the heavy lifting and probably even assigns tasks to individual Developers. (I know that senior developers do not like the idea, but there is no 'team lead' on 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 align the business objective of the upcoming Sprint with the Product Goal and overall product vision. (A serious objective answers the "What are we fighting for?" question. It is also a negotiation between the Product Owner and the Developers to a certain extent. The objective is focused and measurable, as the Sprint Goal and the Developers' forecast go hand in hand.)

- **No business objective, no Sprint Goal, just random stuff**: The Product Owner proposes a direction that resembles a random assortment of tasks, providing no cohesion. Consequently, the Scrum Team does not create a Sprint Goal. (If this is the natural way of finishing your Sprint Planning, 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 aiming to accomplish the Product Goal, composing and ordering 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 Product Backlog items that have not been refined. (Principally, it is the prerogative of the Product Owner to make such kinds of changes to ensure that the Developers work only on the most valuable tasks 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 ordering the Product Backlog and improving team communication. Or the Product Owner needs support to deal with pushy stakeholders.)

- **Output focus:** The Product Owner pushes the Developers to take on more tasks than they could realistically handle. Probably, the Product Owner is referring to former team metrics such as velocity to support their desire. (This is also a road to becoming a feature factory and deserves attention from the team's Scrum Master. It violates the Developers' prerogative to pick Product Backlog items for the Sprint Backlog and

Scrum Values.)

- **No preparation:** The Product Owner does not invest adequately in continuously refining an actionable Product Backlog in collaboration with the Developers. (The Product Backlog needs to represent the best possible use of the Developers' time from a customer value perspective at any given moment. In other words, your Scrum Team's Product Backlog has to be actionable 24/7. By my standards, you need to be capable of running a meaningful Sprint Planning instantly. Preparing a few basic Product Backlog items an hour before the beginning of the Sprint Planning is not enough.)

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 or the Sprint Goal at hand. (The Sprint length is not static but adjustable to offer the Scrum team the best balance between being up-to-date with customer needs while having sufficient, uninterrupted time to focus on meaningful work. When the learning curve is steep, Sprints are short. Once the Scrum team becomes more educated, Sprint lengths might expand. Also, it is pretty common, for example, to extend the Sprint length at the end of the year when most of the team members are on holiday. However, changing the Sprint length flexibly as "needed" is a dark pattern. Instead of changing the Sprint length to accommodate the Sprint Goal, the Scrum Team should invest more effort into the appropriate sizing of tasks.)

- **Kanban through the backdoor:** The Scrum team habitually takes on too many tasks and moves unfinished work directly to the next Sprint. (If two or three items spill over to the next Sprint while the Developers meet the Sprint Goal, so be it. On the other hand, if regularly 30 to 40 percent of the original forecast 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. If the team considers Scrum still to be its choice, I would recommend putting more energy into Product Backlog refinement and creating meaningful Sprint Goals.)

- **Stage-Gate® by DoR:** The Scrum Team decides that a "definition of ready" is advantageous but handles it dogmatically, thus creating a stage-gate-like approval process. (That is an exciting topic for 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 corresponding Product Backlog item into the Sprint — that is fine. However, suppose the "definition of ready" is used as a checklist, rejecting everything that is not 100 percent covered. In that case, you are reintroducing waterfall through the backdoor, only this time it is the Developers doing that. **Read More:** The Dangers of a Definition of Ready.)

- **No standard for considering Product Backlog items 'ready:'** The Scrum team does not have a shared understanding of what a "Sprint-ready" Product Backlog item requires. (This is the opposite side of being dogmatic about applying a "definition of ready." So now, the Developers select unsuited work items that may cause unnecessary disruptions during the Sprint, possibly even endangering achieving the Sprint Goal, into the Sprint. Laissez-faire does not help either.)

- **Forecast imposed:** The Sprint forecast 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 Developers by defining their scope of the forecast. Or a stakeholder points at the team's previous velocity demanding to take on more user stories. ("We need to fill the free capacity.") Or the 'tech lead' of the Developers makes a forecast on behalf of everyone else.)

- **Planning ignored:** The Developers are not participating collectively in the Sprint Planning. Instead, two team members, for example, the "tech lead" and "UX lead," represent the team. (As far as the idea of one or two "leading" teammates in a Scrum team are concerned, there are none, see above. And unless you are using Nexus or 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's voice hence needs to be heard. Otherwise, transparency will suffer and flawed decisions might be made, reducing value creation and increasing risk.)

Sprint Planning Anti-Patterns of the Scrum Master

The Product Owner is responsible for the business objective of the upcoming Sprint. They hence need to guide the Scrum team's effort during Product Backlog refinement to provide an appropriately prepared, actionable Product Backlog before the actual Sprint Planning. At the same time, the Developers are in charge of selecting the necessary Product Backlog items to meet the collaboratively created Sprint Goal. So, what are Sprint Planning anti-patterns of the Scrum Master, you may ask yourself?

Besides being complicit in the Sprint Planning anti-patterns sketched above by not addressing them, I can think of one Sprint Planning anti-pattern that directly falls into the responsibility of the Scrum Master. Despite the changes to the Scrum Guide in 2020, addressing an important improvement issue from the previous Sprint Retrospective every Sprint is still an important step to help the Scrum Team improve continuously. Therefore, failing to support the Scrum team in that matter constitutes a Sprint Planning anti-pattern in my eyes.

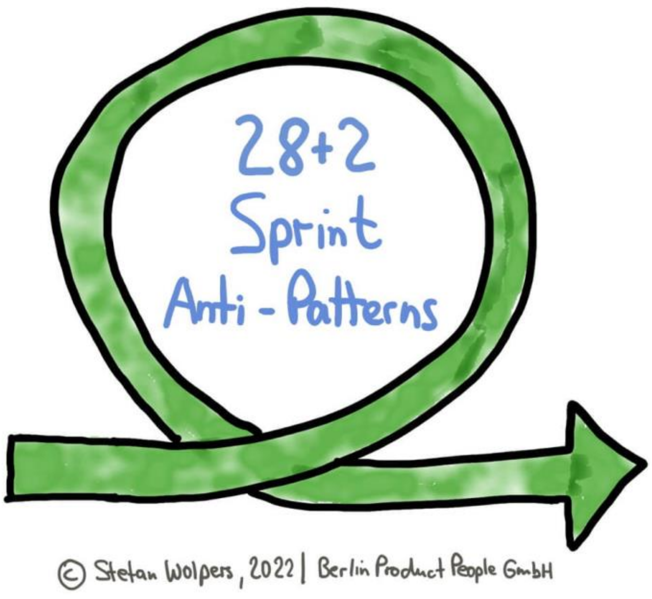
Conclusion

The Sprint Planning is a core event, defining how your customers' lives will improve with the next Product Increment and not be taken lightly. Fortunately, most of the beforementioned Sprint Planning anti-patterns are simple to fix. Just take it to the team, respect Scrum Values, self-management, and Scrum's built-in checks & balances.

28 Sprint Anti-Patterns

The Sprint

Welcome to the Sprint anti-patterns article from our series on Scrum anti-patterns, covering the three Scrum roles—pardon me: accountabilities—and addressing the contributions of stakeholders and the IT/line management. Moreover, we add some food for thought. For example, could a month-long Sprint be too short for accomplishing something meaningful? And if so, what are the consequences?



The Purpose of the Sprint

The purpose of the Sprint is clearly described in the Scrum Guide — no guessing is necessary:

Sprints are the heartbeat of Scrum, where ideas are turned into value.

They are fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint.

All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happen within Sprints.

Sprints enable predictability by ensuring inspection and adaptation of progress toward a Product Goal at least every calendar month. When a Sprint's horizon is too long the Sprint Goal may become invalid, complexity may rise, and risk may increase. Shorter Sprints can be *employed to generate more learning cycles and limit risk of cost and effort to a smaller time frame. Each Sprint may be considered a short project.*

Various practices exist to forecast progress, like burn-downs, burn-ups, or cumulative flows. While proven useful, these do not replace the importance of empiricism. In complex environments, what will happen is unknown. Only what has already happened may be used for forward-looking decision making.

Source: Scrum Guide 2020.

Scrum as a framework is mainly of a tactical nature. The Sprint is about delivering more value to customers, guided by the Sprint Goal, based on previously explored and validated hypotheses. It is all about getting things out of the door, thus closing the feedback loop and starting another round of inspection and adaption.

Therefore, in my eyes, the actual Sprint is not the place to have meta-level discussions. However, there is plenty of time for these during product discovery, aligning with stakeholders, and refining the Product Backlog.

Sprint Anti-Patterns

This list of notorious Sprint Anti-Patterns applies to all Scrum roles and beyond: the Product Owner, the Development Team, the Scrum Master, the Scrum Team itself, 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 Developers. (As the Sprint Backlog is emergent and the Developers may identify necessary new work or the scope of previously identified work may need to be adjusted, the Product Owner's absence can leave the Developers in the dark, risking the accomplishment of the Sprint Goal.)

- **PO clinging to tasks:** The Product Owner cannot let go of Product Backlog items once they become part of the Sprint Backlog. For example, the Product Owner increases the scope of a work item or changes acceptance criteria once the Developers select this Product Backlog item for the Sprint Backlog. (There is a clear line: before a Product Backlog item becomes part of the Sprint Backlog, the Product Owner is responsible. However, once it moves from one backlog to the other, the Developers become responsible. If changes become acute during the Sprint, the team will collaboratively decide 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 core Scrum principles.)

- **Delaying PO:** The Product Owner does not provide feedback on work items from the Sprint Backlog once those are done. Instead, they wait until the end of the Sprint. (The Product Owner should immediately inspect Product Backlog items that meet the acceptance criteria. Otherwise, the Product Owner will create an artificial queue within the Sprint, unnecessarily increasing the cycle time. This habit also puts reaching the Sprint Goal at risk. Note: Inspecting Product Backlog items does equal some sort of work acceptance or quality gate. There is no such thing in Scrum. Once a Product Backlog item meets the Definition of Done, it can be released into the hands of users.)

- **Sprint stuffing:** The Developers accomplished the Sprint Goal early, and the Product Owner is pushing them hard to accept new work from the Product Backlog to fill the "void." (The Scrum Team decided collaboratively on the Sprint Goal, and the Developers committed to delivering. Consequently, it is the prerogative of the Developers to decide on the composition of the Sprint Backlog. Should they manage to accomplish the Sprint Goal before the Sprint's end, it is their sole decision to fill the remaining time. Accepting new work from the Product Backlog may be one way of filling the remaining Sprint time-box. This also applies to refactoring parts of the tech stack, exploring new technology that might be useful, fixing some bugs, or sharing knowledge with fellow teammates. Scrum is not in the business of maximizing the utilization rates of team members. Achieving the Sprint Goal is sufficient.)

- **Sprint cancellations without consultation:** The Product Owner cancels Sprints without consulting the Scrum Team. (It is the prerogative of the Product Owner to cancel Sprints. However, the Product Owner should not do this without a serious cause. The Product Owner should also never abort a Sprint without consulting the rest of the team. Maybe, someone has an idea on how to save the Sprint Goal, provided it is still useful? Misusing the cancellation privilege indicates a serious team collaboration issue and a lack of commitment to live Scrum Values.)

- **No Sprint cancellation:** The Product Owner does not cancel a Sprint whose Sprint Goal can no longer be achieved or has become obsolete. (If the Scrum Team identified a unifying Sprint Goal, for example, integrating a new payment method, and the management then abandons that functionality mid-sprint, continuing working on the Sprint Goal would be wasteful. In such a case of obsolescence, the Product Owner has to consider canceling the Sprint.)

Sprint Anti-patterns of the Developers

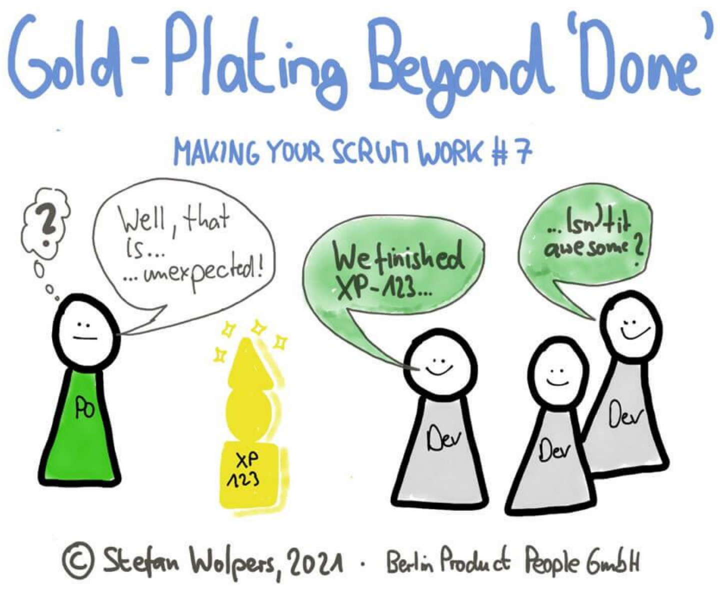
- **No WiP limit:** There is no work in progress limit. (The purpose of the Sprint is to deliver a potentially shippable Product Increment that provides value to the customers and thus to the organization. This goal requires focused work to accomplish the necessary work to meet the Sprint Goal by the end of the Sprint. The flow theory suggests that a team's productivity improves with a work-in-progress (WiP) limit. The WiP limit defines the maximum number of tasks the Developers can work on simultaneously. Exceeding this WiP number creates additional queues that consequently reduce the overall throughput of the Developers. The cycle time—the period between starting and finishing a ticket—measures this effect.)

- **Cherry-picking:** The Developers cherry-pick 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 may notice tickets queueing in the code-review-column, for example. It is also a sign that the Developers are not yet fully self-organizing. Look for Daily Scrum events that support this notion and address the issue during the Sprint Retrospective.)

- **Board out-of-date:** The Developers do not update tickets on the Sprint board in time to reflect the current statuses. (The Sprint board, no matter if it is a physical or digital board, is not only vital for coordinating the Developers' 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 confidence may then cause counter-measures on the side of the stakeholders. Consequently, the (management) pendulum may swing back toward traditional methods. The road back to PRINCE II is paved with abandoned boards.)

- **Side-gigs:** The Developers are 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 for the return on investment the Scrum Team is creating—is unacceptable. This behavior also signals a substantial 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 Developers are probably rather a group than a team anyway.)

- **Gold-plating:** The Developers increase the scope of the Sprint beyond the Sprint Goal by adding unnecessary work to the Product Backlog items of the Sprint Backlog. (This effect is often referred to as scope-stretching or gold-plating. The Developers ignore 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 development time. However, there is a simple solution: the Developers and the Product Owner need to talk more often, creating a shared understanding from product vision down to the individual Product Backlog item, thus improving the trust level. If the Product Owner is not yet co-located with the Developers, now would be the right moment to reconsider. Alternatively, a distributed Scrum Team may invest more time in how to communicate synchronously and asynchronously best to improve its level of collaboration and alignment.)


Gold-Plating Beyond 'Done'
MAKING YOUR SCRUM WORK #7
?
Well, that is... ...unexpected!
We finished XP-123...
...Isn't it awesome?
PO
XP 123
Dev
Dev
Dev
© Stefan Wolpers, 2021 · Berlin Product People GmbH

Sprint Anti-patterns of the Scrum Master

- **Flow disruption:** The Scrum Master allows stakeholders to disrupt the flow of the Scrum Team during the Sprint. There are several possibilities for how stakeholders can interrupt the team's flow during a Sprint. Any of the following examples will impede the team's productivity and might endanger accomplishing the Sprint Goal:

    - The Scrum Master has a laissez-faire policy as far as access to the Developers is concerned. Notably, they are not educating stakeholders on the negative impact of disruptions and how those may endanger accomplishing the Sprint Goal. **Note**: I do not advocate that Scrum Masters shall restrict stakeholders' access to team members in general.

    - The Scrum Master does not oppose line managers taking team members off the Scrum Team, assigning them to other tasks.

    - The Scrum Master does not oppose line managers adding members to the Scrum Team without prior consultation of the team members. (Preferably, the Scrum Team members should decide who is joining the team.)

    - Lastly, the Scrum Master allows stakeholders or managers to turn the Daily Scrum into a reporting session. (This behavior will impede the Developer's productivity by undermining their self-management while reintroducing command & control through the backdoor.)

- **Lack of support:** The Scrum Master does not support team members that need help. Often, Developers create tasks an engineer can finish within a day during their planning sessions. However, if someone struggles with such a job for more than two days without voicing that they need support, the Scrum Master should address the issue and offer their support. Making this issue visible is also the reason for marking tasks on Sprint boards with red dots each day if work items do not move to the next column. (In other words: we are tracking the work item age.)

- **Micro-management:** The Scrum Master does not prevent the Product Owner—or anyone else—from assigning tasks to Developers directly. (The Developers organize themselves without external intervention. The Scrum Master is the first line of defense of the Developers in that respect.)

- **#NoRetro:** There is no Retrospective as the team believes there is nothing to improve, and the Scrum Master accepts this notion. (There is no such thing as an agile Nirwana where everything is perfect. As people say: becoming agile is a journey, not a destination, and there is always something to improve.)

Note: I do not believe that it is the task of the Scrum Master to maintain a Sprint board, for example, by moving tickets. The Developers should do this during the Daily Scrum if they consider this helpful. 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 Developers consider a burn-down chart helpful, they should also update the chart themselves. #justsaying #scrummasterisnotascribeorsecretary

Sprint Anti-patterns of the Scrum Team

- **Not delivering the Sprint Goal:** The Scrum Team misses the Sprint Goal more often than delivering it. (We are not getting paid to practice Scrum. We are paid to solve our customers' problems within the given constraints, allowing our organization to build a sustainable business in the process. Given Scrum's delivery focus and tactical nature, agreeing on a Sprint Goal and then delivering it is the most critical contribution of the Scrum Team. While there may be many situations where the team cannot accomplish the Sprint Goal occasionally, for example, because of technical issues, skill deficits, and the complexity and uncertainty of life itself, this failure should be the exception, not the rule. 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 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.)

- **The maverick & the Sprint Backlog:** Someone adds an item to the Sprint Backlog without consulting the Developers first. (The control of the Sprint Backlog by the Developers is at the core of enabling the team to make a forecast. The scope is hence initially untouchable. Changes in the composition of Sprint Backlog are possible, for example, when a critical bug pops up after a Sprint's start or new work vital to accomplishing the Sprint Goal is identified. However, adding such an issue to the Sprint Backlog requires approval by the Developers and probably a level of compensation. Another task of a similar size might need to go back to the Product Backlog. All these exceptions have in common that the Developers collectively have the final say. No individual member of the Scrum team can add or remove an item to or from the Sprint Backlog single-handedly.)

- **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. Instead, the Sprint's objective is to deliver a valuable, potentially shippable Increment. For that purpose, the Scrum Team creates a Definition of Done to ensure that everyone understands the required quality level for a Product Increment to be "shippable" to customers. Declaring buggy tasks "done" thus violates this core Scrum principle of collaboration. Furthermore, hardening Sprints are commonly a sign of a low grade of adopting agile principles by the team or the organization. This anti-pattern probably results from the Scrum Team not yet being cross-functional. Or quality assurance is still handled by a functional, non-agile silo within the product delivery organization. Alternatively, the Developers might have felt pressured to deliver to meet an (arbitrary) deadline, and they decided to cut corners by reducing quality. No matter the reason, in such a situation, the Scrum team needs to go back to Scrum's first principles; here: "done" means releasable.)

- **Delivering Y instead of X:** The Product Owner believes in getting X. The Developers are working on Y. (This is not merely a result of an inferior Product Backlog refinement. This anti-pattern indicates that the Scrum Team failed to create a shared understanding. There are plenty of reasons for this to happen, just to list a few:

    - The Product Owner and the Developers are not talking enough during the Sprint. (For example, the Product Owner is too busy to answer the team's questions or attend the Daily Scrum, or Product Backlog refinement sessions are scarce.)
    - No Developer has ever participated in user tests or interviews. Hence there is a lack of understanding of the users' problems among engineers. (This is the reason why engineers should interview users regularly.)
    - The Product Owner presented a too granular user story. Therefore, Developer did not care enough to have a thorough look as the user story seemed ready. (A typical crowding-out effect.)
    - Probably, the user story was missing acceptance criteria altogether, or existing acceptance criteria missed the problem. No matter the reason, the Scrum Team should address the issue during the next Retrospective.)

- **New kid on the block:** The Scrum Team welcomed a new team member during the Sprint. Unfortunately, they also forgot to address the issue during Sprint Planning, thus ending up overextended. (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 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 metric. Extending the Sprint length is not agile; it is just inconsequential. Additionally, it has a devastating effect on stakeholder inclusion, as Scrum events, for example, the Sprint Review, lack a proper cadence, reducing your stakeholder's willingness to collaborate. So, stop lying to yourself, and address the underlying issues why the team outcome does not meet the Sprint Goal.)

Sprint Anti-patterns of the IT Management

- **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 and a bias for action to feel "in control" of the situation on their side. Most likely, canceling Sprints and gathering the Scrum Teams would also solve the issue.)

- **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 Scrum Team members can build trust among each other. The longevity of Scrum Teams has proven beneficial to build that trust. Moving people between teams, on the contrary, reflects a project-minded idea of management, rooted in utilization optimization of "human resources" of the industrial paradigm. It also ignores the preferred team-building practice that Scrum Teams should select themselves. All members need to be voluntarily on a team. Scrum does not work if team members are pressed into service. **Note**: It is not an anti-pattern, though, if Scrum Teams decide to exchange teammates temporarily. It is an established practice that specialists spread knowledge this way or mentor other colleagues. Also, dynamic re-teaming does not constitute an anti-pattern when the members of the involved Scrum Teams decide to do so.)

- **Special forces:** A manager assigns specific tasks directly to Developers, thus bypassing the Product Owner and ignoring the Developer's self-organization prerogative. Alternatively, the manager removes a Developer 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 of command and control practices. They continue to micromanage subordinates, although a Scrum Team could accomplish the task self-organized. This behavior demonstrates a level of ignorance that may require support for the Scrum Master from a higher management level to deal with.)

Sprint Review Anti-patterns of Stakeholders

- **(Regular) emergency work:** Someone in your organization has sold a non-existing feature or functionality to a customer to close a deal, probably already including delivery dates and contractual penalties in the case of non-delivery. Now, they want the Scrum Team to focus on delivering this item. (There might be moments where this outside intervention in the Scrum process may be unfortunate but acceptable. The more concerning issue here is the prospect of this behavior becoming a regularity. If the leadership does not acknowledge the exceptionality of the situation, it may derail using Scrum in the organization.)

- **Pitching developers:** The stakeholders try to sneak in small tasks by pitching them directly to Developers. (Nice try #1. However, all suggestions for the future allocation of the Scrum Team's time compete with each other, and the Product Owner is the referee in this process.)

- **Everything's a bug:** The stakeholders try to speed up delivery by relabeling their tasks are 'serious bugs.' (Nice try #2. A particular case is an "express lane" for bug fixes and other urgent issues which some Scrum teams establish. In my experience, every stakeholder will try and make their tasks eligible for that express lane.)

Food for Thought

There are some issues that are worthwhile considering as a Scrum Team regarding the Sprint:

What if a month-long Sprint is still too short? There are areas where a month-long Sprint may prove to be too short, for example, in hardware development or machine learning when training new models. Would it be okay to extend the length of a Sprint? Or would such a situation signal abandoning Scrum altogether and moving to alternatives such as Shape Up?

Dying in beauty? Is there a moment when circumstances become so desperate that abandoning Sprints is a valid option? For example, think of a startup running out of cash, and the only way to survive is achieving a particular milestone defined by a prospective new investor. Would that be a moment when to abandon Scrum's rigorous process to have a fighting chance?

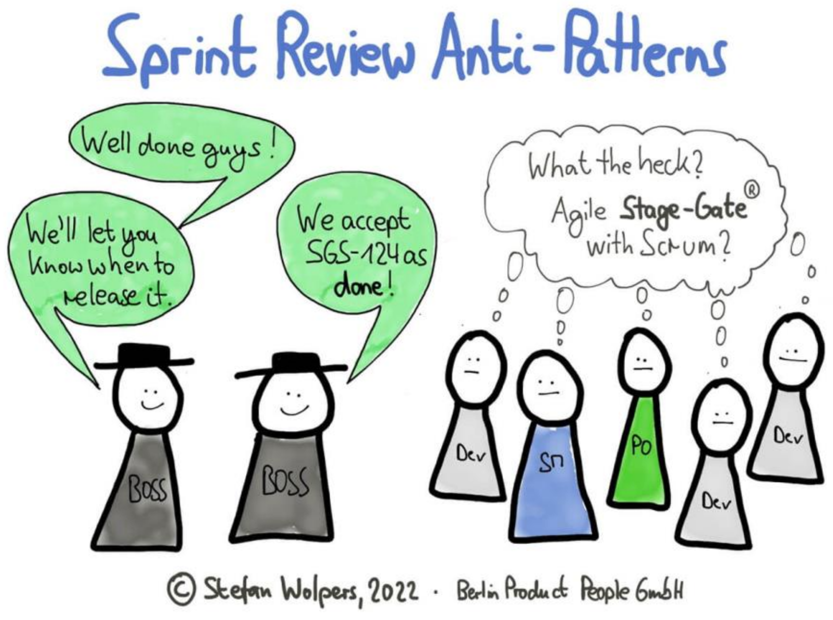
Conclusion

Although the Sprint itself is just a container for all other Scrum events, there are plenty of Sprint anti-patterns to observe. A lot of them are easy to fix by the Scrum Team or the Scrum Master. However, some sprint anti-patterns point at organizational issues that probably will require more than a Sprint Retrospective to change.

15 Sprint Review Anti-Patterns

The Sprint Review

Are we still on track to accomplish the Product Goal? Moreover, how did the previous Sprint contribute to our Scrum team's mission? Answering these questions and adapting the Product Backlog in a collaborative effort of the Scrum Team with 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.


Sprint Review Anti-Patterns
Well done guys!
We'll let you know when to release it.
We accept SGS-124 as done!
What the heck? Agile Stage-Gate® with Scrum?
BOSS
BOSS
Dev
SM
PO
Dev
Dev
© Stefan Wolpers, 2022 · Berlin Product People GmbH

The Scrum Guide on the Sprint Review

According to the Scrum Guide, the Sprint Review serves the following purpose:

The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed.

During the event, the Scrum Team and stakeholders review what was accomplished in the Sprint and what has changed in their environment. Based on this information, attendees collaborate on what to do next. The Product Backlog may also be adjusted to meet new opportunities. The Sprint Review is a working session and the Scrum Team should avoid limiting it to a presentation.

The Sprint Review is the second to last event of the Sprint and is timeboxed to a maximum of four hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.

Source: Scrum Guide 2020.

The Sprint Review is Empiricism at work: inspect the Product Increment and adapt the Product Backlog. The Developers, the Product Owner, the Scrum Master, and the stakeholders need to figure out whether the Scrum team is still on track to accomplishing its Product Goal. 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 time, thus maximizing the value delivered to customers within the given constraints while contributing to the viability of the organization. It is also because of this context that calling the Sprint Review a "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 a Scrum team as soon as possible.

Sprint Review Anti-Patterns

Often, you can observe some of the following Sprint Review anti-patterns.

Sprint Review Anti-Patterns of the Product Owner

- **The Selfish PO:** Product Owners present "their" accomplishments to the stakeholders. (Scrum is a remarkably egalitarian affair: No one has any authority to tell teammates what, when, or how to do something. There are no sub-roles on Scrum teams, and there is no hierarchy. Instead, Scrum builds everything on self-management and collective responsibility—the Scrum team wins together, and the Scrum team loses together. Remember the old saying: There is no "I" in "team?")

- **"Acceptance" by the PO:** The Product Owner uses the Sprint Review to "accept" Product Backlog items that Developers consider "done." (A formal "acceptance" of work packages by the Product Owner is not foreseen in Scrum, for there is the Definition of Done. However, a feedback loop—did the Developers deliver the agreed-upon functionality?—is valuable. However, Product Owners should also decouple this feedback loop from the Sprint Review. Instead, Product Owners should communicate with the Developers whenever needed or when work items meet the Definition of Done.)

- **Unapproachable PO:** The Product Owner is not accepting feedback from stakeholders or the Developers. (I get it: Loving your solution over the customers' problems feels good. Moreover, a bit of confirmation bias may prevent our Product Owners from getting distracted from pursuing their beloved solution. But unfortunately, they do not get paid to roll out their solution. This "living in their PO bubbles" approach hence violates the prime purpose of the Sprint Review event: Figure out collaboratively whether the Scrum team is still on track to meeting the Product Goal—which requires openness on the side of the Product Owners in the first place.)

Sprint Review Anti-patterns of the Developers

- **Death by PowerPoint:** Participants of the Sprint Review 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 discovery. The Sprint Review is not a "demo," carefully avoiding all obstacles to preserve the illusion of progress and control. Instead, it is an essential opportunity to inspect what the Scrum team accomplished, receive valuable feedback, and adapt the Product Backlog collectively. It is about creating transparency on the state of the progress toward the Product Goal.)

- **Same faces again:** It is always the same Developers who participate in the Sprint Review; however, they are not all Developers. (Unless the organization scales Scrum based on LeSS or Nexus and we are talking about the overall Sprint Review, this Sprint Review anti-pattern of limited attendance of Scrum team members is a bad sign. The Sprint Review needs all Scrum team members on deck to maximize the learning. The challenge is that you cannot enforce your teammates' participation either. Instead, make it interesting enough that everyone wants to participate. If this is not happening, you should ask yourselves how you have contributed to this situation in the past. It is lucky a coincidence that an event tailored to this need follows immediately after the Sprint Review — the Retrospective.)

- **Side gigs:** The Developers worked on issues outside the Sprint Goal, and the Product Owner learned about those for the first time during the Sprint Review. (This Sprint Review anti-pattern is truly "rocking the boat." Focus, commitment, and openness — just to name the most apparent Scrum values violated here — are the first principles for collaboration among Scrum team members. Anything that the Developers want to address during the Sprint needs to be communicated during the Sprint Planning. However, a particular case is when the Product Owner is usually so pushy about shipping new features that there is little or no time to attend to refactoring or bug fixing unless the Developers tackle these jobs under the radar. In this situation, I would have sympathy for the approach. Nevertheless, the Scrum team needs to fix this problem. Generally, allocating 20 % of the team's capacity to the before-mentioned tasks could be a start.)

- **Undone is the new "done:"** More often than not, the Developers show work items that are not "done." (There is a good reason to show unfinished work on some occasions. For example, to provide transparency to stakeholders on essential yet challenging tasks. However, regularly reporting to Sprint Review attendees on partially finished work violates the concept of "Done," one of Scrum's first principles. There is no need for a successful Scrum team to demonstrate to stakeholders that they are worth their pay-cheques.)

Sprint Review Anti-Patterns of the Scrum Team

- **No Sprint Review**: There is no Sprint Review, as the Developers did not meet the Sprint Goal. (A rookie mistake. Particularly in such a situation, a Sprint Review is necessary to create transparency with stakeholders and inspect the Increments that the Developers nevertheless managed to accomplish.)

- **Following the plan**: The Scrum Team does not use the Sprint Review to discuss the current state of progress toward the Product Goal with the stakeholders. (Again, creating transparency and receiving feedback is the purpose of the exercise. A we-know-what-to-build attitude is bordering on hubris. We do not want to recreate "watermelon" status reports regarding the Product Goal: green on the outside; however, deep red on the inside. 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. (My suggestion: Tell a compelling story at the beginning of the Sprint review on the task the Scrum team ventured out to accomplish and engage the stakeholders with that narrative. Leave out those work items that are probably not relevant to the story. Do not bore stakeholders by including everything that was accomplished. We are not accountants; the output is far less relevant by comparison to the outcome.)

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 Sprint Review anti-pattern is typical for organizations that use an "agile"-waterfall hybrid: there are some happy agile islands, for example, our Scrum team, surrounded by a sea of waterfall, driven by functional silos, budgeting, and top-down goal-setting. Still, in such a world, the Scrum team is tasked with accomplishing a Product Goal. Therefore, it is the prerogative of the Scrum Team to decide what to ship and when.)

- **No stakeholders:** Stakeholders do not attend the Sprint Review. (There are several reasons why stakeholders do not participate in the Sprint Review: they do not see any value in the event, or it is conflicting with another important meeting. In addition, 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 event within the organization, at least at the beginning of using Scrum.)

- **No customers present:** External stakeholders—also known as customers—do not attend the Sprint Review. (Break out of your organization's echo chamber and invite some customers and users to your Sprint Review. And do not let the sales folks object to this idea. Ignoring the direct feedback from customers at the Sprint Review inevitably leads to a lesser outcome.)

- **Starting over again:** There is no continuity in the attendance of stakeholders. (Longevity is not just beneficial at the Scrum team level but also applies to stakeholder attendance. If they change too often, for example, because of a rotation scheme, their ability to provide in-depth feedback might be limited. If this pattern appears, the Scrum 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 where team members introduce solutions to specific problems the Sprint addressed. Shift & Share is an excellent Liberating Structure microstructure for that purpose.)

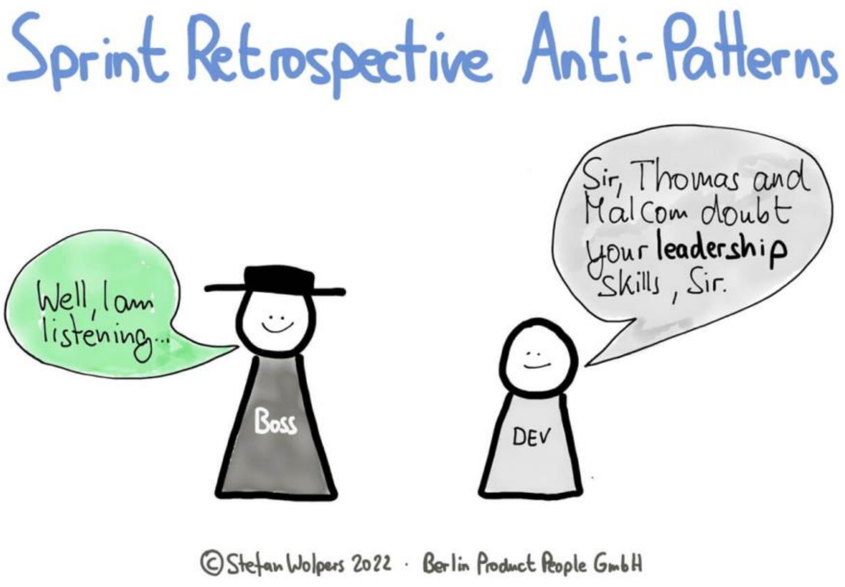
Conclusion

Scrum's Sprint Review is a critical Scrum event. It answers whether the Scrum Team is still on track to deliver the best possible value to the customers and the organization as envisioned by the current Product Goal. Therefore, avoiding the before-mentioned Sprint Review anti-patterns can significantly improve a Scrum Team's effectiveness in making your customers' and users' lives easier while contributing to a viable organization.

21 Sprint Retrospective Anti-Patterns Impeding Scrum Teams

Introduction

What event could better embody Scrum's principle of empiricism than the Sprint Retrospective? I assume all peers agree that even the simplest form of a Retrospective—if only held regularly—is far more helpful than having a fancy one once in a while, not to mention having none. Moreover, I am convinced there is always room for improvement; just avoid dogmatism. Hence, learn more about 21 common Sprint Retrospective anti-patterns that will hold back your Scrum team.


Sprint Retrospective Anti-Patterns
Well, I am listening...
Boss
Sir, Thomas and Malcom doubt your leadership skills, Sir.
DEV
©Stefan Wolpers 2022 · Berlin Product People GmbH

The Scrum Guide on the Sprint Retrospective

According to the Scrum Guide, the Sprint Retrospective serves the following purpose:

The purpose of the Sprint Retrospective is to plan ways to increase quality and effectiveness. The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done. Inspected elements often vary with the domain of work. Assumptions that led them astray are identified and their origins explored. The Scrum Team discusses what went well during the Sprint, what problems it encountered, and how those problems were (or were not) solved.

The Scrum Team identifies the most helpful changes to improve its effectiveness. The most impactful improvements are addressed as soon as possible. They may even be added to the Sprint Backlog for the next Sprint.

The Sprint Retrospective concludes the Sprint. It is timeboxed to a maximum of three hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.

Source: Scrum Guide 2020.

By any standard, this is a compelling pitch of the benefits of Retrospectives, addressing the why, the what, and the how.

Suppose a Scrum team takes inventing on behalf of customers and creating valuable, viable, feasible, and usable products seriously. In that case, it should be highly motivated to diligently close this feedback loop at the team level every Sprint. Furthermore, everyone should be looking forward to identifying how to make the next Sprint better and more enjoyable: Are we still progressing towards the team goal? (Regarding the "goal" aspect, the Retrospective resembles the Daily Scrum — inspecting the progress towards the Sprint Goal — and the Sprint Review: inspecting the progress towards the Product Goal.)

But unfortunately, many Scrum teams experience something different. Fortunately, though, many of the Retrospective anti-patterns listed below can be addressed by the Scrum team members themselves.

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 Developers, 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, and the Scrum Master accepts this notion. (There is no such thing as an agile Nirwana where everything is perfect. As people say: becoming agile is a journey, not a destination, and there is always something to improve.)

- **Dispensable buffer:** The Scrum team cancels Retrospectives if more time is needed to accomplish the Sprint Goal. (The Retrospective as a Sprint emergency reserve is a common sign of cargo cult Scrum. 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 a Retrospective to achieve a Sprint Goal clearly shows that the team does not understand basic principles, such as empiricism and continuous improvement. If the Scrum team repeatedly does not meet the Sprint Goal, it should inspect what is happening here. Guess which Scrum event is designed for that purpose?)

- **Rushed Retrospective:** The team is in a hurry and allocates much less than the necessary 60 to 180 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 sufficient time or consider not running Retrospectives. And while you are at it, why don't you abandon Scrum altogether?)

- **Let's have the retro next Sprint:** The Scrum team postpones the Retrospective to 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. Additionally, the Scrum team shall inspect their way of working and identify opportunities to make the next Sprint more effective and fun. This is why we have the Retrospective before the Sprint Planning of the upcoming Sprint. Moreover, postponing the Retrospective into the next Sprint may interrupt the team's flow and leave one or more critical team issues unattended for the length of a Sprint. Also, it is a slippery slope. If postponing Retrospectives becomes an acceptable approach, why have them in the first place?)

- **Excluding stakeholders all the time:** The Scrum team categorically rejects the idea of having Retrospectives with their stakeholders. (The presence of stakeholders or line managers at team Retrospectives is generally not a good idea. However, excluding these two groups from all kinds of Retrospectives is a short-sighted practice. As a result, the Scrum team will never manage to evolve to its true potential as they deliberately reduce available feedback and thus flatten their learning curve. Several opportunities in Scrum address stakeholders' communication and information needs: the Sprint Review, listening in at the Daily Scrum, probably even the Product Backlog refinement, not to mention opportunities of having a conversation in Zoom, at water coolers, over coffee, or during lunchtime. However, if that spectrum of possibilities is not sufficient, consider having additional meetings if your team deems them necessary. For example, there is the opportunity to have a meta-level Retrospective, explicitly including the Scrum team's stakeholders. However, the Retrospective as a Scrum team-internal event is off-limits to stakeholders.)

- **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 to this rule.)

- **Extensive whining:** The Scrum team uses the Retrospective primarily to complain about the situation and assumes the victim's role. (Change requires reflection, and occasionally it is a suitable 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.)

- **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, 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:** At the last Retrospective, the team members accepted action items. However, no one took responsibility for the delivery. (If the "team" is supposed to fix X, probably everyone will rely on their teammates to handle it. So make someone responsible instead.)
- **What improvement?** The team does not check the status of the action items from 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 Scrum Master

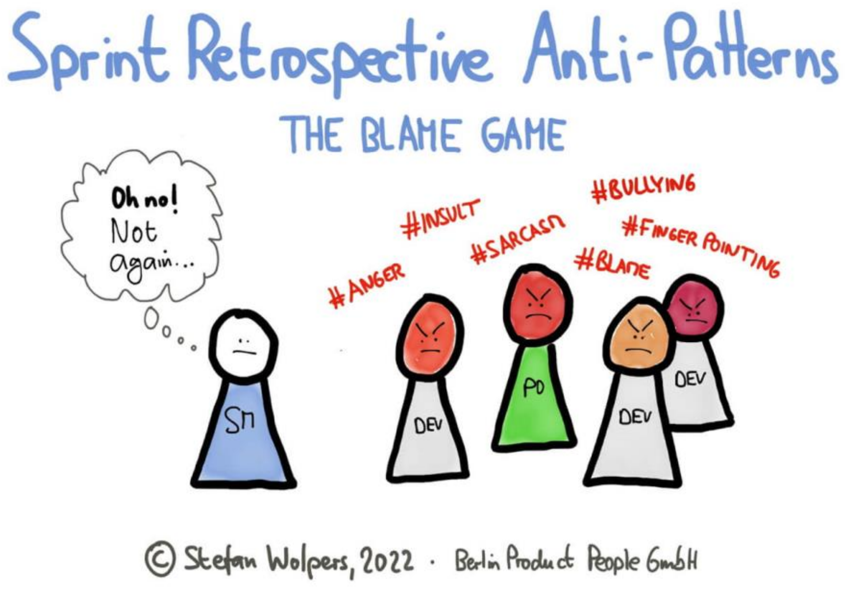
- **Waste of time**: The Scrum team does not collectively value the Retrospective. (If some team members consider the Retrospective to be of little or no value, it is most often the Retrospective itself that sucks. Is it the same procedure every time, ritualized and boring? Have a meta-retrospective 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, if appropriately facilitated.)

- **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 force. My tip: Retromat's "Why are you here?" exercise is a good opener for a Retrospective from time to time.)

- **Groundhog day:** The Sprint Retrospective never changes in composition, venue, or length. There is a tendency in this case that the Scrum team might revisit the same issues repeatedly–it's Groundhog Day without the happy ending, though.

- **#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 fails together. The blame game documents both the failure of the Scrum Master as the facilitator of the Retrospective and the Scrum team's lack of maturity and communication skills.)



- **Bullying is accepted:** One or two team members dominate 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 their feedback free from third-party influence. If some of the team members dominate the conversation and probably even bully or intimidate 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 less valuable. It is the primary responsibility of the Scrum Master as a facilitator to ensure that everyone will be heard and has an opportunity to voice their 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.

- **Passivity**: The Scrum team members are present but are not participating. (There are plenty of reasons for such a behavior: they regard the Retrospective as a waste of time, it is an unsafe place, or the participants are bored to death by its predictiveness or likely lack of progress. Probably, the team members are afraid of negative repercussions in the case of their absence and hence decided to show up. Unfortunately, in this case, there is no quick fix, and the Scrum Master needs to figure out what kind of Retrospective works in their team's context. Take it to the team.)

Sprint Retrospective Anti-Patterns of the Organization

- **No suitable venue:** There is no good place available to run the Retrospective. (Let's assume that we won't spend the rest of our professional lives in virtual team settings for a moment. Then 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 regularly participate in Retrospectives. (This is among the worst Sprint Retrospective anti-patterns I can imagine. It turns the Retrospective into an unsafe place. And who would expect that an unsafe place would trigger an open discussion among the team members? Any line manager who insists on such a proceeding signals their lack of understanding of basic Scrum practices.)

Sprint Retrospective Anti-Patterns
LINE MANAGER ALERT
BOYS, I WILL BE JOINING YOUR RETROS!
Looking forward to guiding you...
NO!
???
What the heck?
Boss
SM
PO
Dev
Dev
Dev
Dev
© Stefan Wolpers, 2022 · Berlin Product People GmbH

- **Reporting structures within Scrum teams:** There are hierarchies among Scrum team members. For example, a junior Developer reports to a senior Developer on the same Scream team. (This situation requires the immediate attention of the Scrum Master as it — at least in my experience — instantly turns the Retrospective into an awkward and significantly less helpful event.)
- **Let us see your minutes:** Someone from the organization — outside the Scrum team — requires access to the Retrospective minutes. (This is almost as bad as line managers who want to participate in a Retrospective. But, of course, this information is solely available to team members.)

Conclusion

There are many ways in which a Sprint Retrospective can be a failure, even if it looks suitable at first glance. From my perspective, the top three Sprint Retrospective anti-patterns are: not making the Retrospective a safe place, forcing team members to participate in an exercise they consider a waste of their time, and not taking the Retrospective seriously in the first place.

Scrum Role Anti-Patterns

Product Owner Anti-Patterns

If you are working as a Product Owner, there is—very likely—room for improvement. This list of some of the most common Product Owner anti-patterns might be a starting point. Hence, if you recognize some anti-patterns in your daily work, why don't you ask the rest of the Scrum Team for support? For example, run a Retrospective with teammates and stakeholders on how the team is doing regarding figuring out what is worth building.

Please remember: No other role in Scrum can contribute to mediocre outcomes like the Product Owner—garbage in, garbage out.

Product Owner Anti-Patterns
What about Refinement?
#facepalm
Card, Conversation, Confirmation anyone?
Our stakeholders are awesome!
We now have an epic PRD!
I copied it to Jira—chop, chop, get to work!
DEV
PO
DEV
PO
©Stefan Wolpers 2022 · Berlin Product People GmbH

The Role of the Product Owner According to the Scrum Guide

The Product Owner is accountable for maximizing the value of the product resulting from the work of the Scrum Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.

The primary way to achieve this goal as a PO is to manage the Product Backlog effectively. According to the Scrum Guide, this activity comprises:

- *Developing and explicitly communicating the Product Goal;*
- *Creating and clearly communicating Product Backlog items;*
- *Ordering Product Backlog items; an*
- *Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next; and,*

- *Ensuring that the Product Backlog is transparent, visible and understood.*

The Product Owner may do the above work or may delegate the responsibility to others. Regardless, the Product Owner remains accountable.

For Product Owners to succeed, the entire organization must respect their decisions. These decisions are visible in the content and ordering of the Product Backlog, and through the inspectable Increment at the Sprint Review.

Source: Scrum Guide 2020.

In my experience, most Product Owner anti-patterns result from a misunderstanding of this Product Backlog management task. Notably, less successful Product Owners tend to fail at the communication part with teammates and stakeholders. Maximizing the value of the Scrum team's work from a customer perspective while contributing to the bottom-line of the organization is less a question of project management skills, juggling with product requirement documents, and keeping the Jira up-to-date.

Therefore, successful Product Owners spend less time on actually handling the Product Backlog than they allocate to product discovery in general and creating alignment among all players in particular. To them, the Product Backlog is a means to an end, not the purpose of their existence.

Consequently, successful Product Owners spend time creating a transparent system that allows identifying potentially valuable ideas from all sources, relentlessly communicating the "Why," and collaborating with the Developers on the "What."

Product Backlog and Refinement Anti-Patterns

You can spot many of the Product Owner anti-patterns in the PO's backyard — the Product Backlog and its refinement:

- **Storage for ideas:** The Product Owner uses the Product Backlog as a repository of ideas and requirements. (A large Product Backlog is probably considered a sign of a "good" Scrum team: You are fully transparent, and this is proof of your usefulness to the organization. However, being "busy" does not equal value to customers and the organization. The additional noise created by the sheer number of issues may also cloud the detection of valuable items. Lastly, the Product Backlog size may have a crowding-out effect on stakeholders as they feel overwhelmed. The idea-inflated Product Backlog may impede critical communication with them as a consequence.)

- **Part-time PO:** The Product Owner is not working daily on the product backlog. (The Product Backlog needs to represent the best use of the Developers' time at any given time. Suppose an update to the Product Backlog happens only occasionally, for example, shortly before the next Sprint Planning. Due to a lack of refinement, it is likely to leave a lot of value on the table. The value of the Product Backlog results in a significant part from the open discussion between the Product Owner and the Developers. In a good Scrum team, the Developers constantly challenge the Product Owner regarding the value of suggested Product Backlog items. This checks & balances approach reduces the probability of the Product Owner falling victim to confirmation bias, thus mitigating the risk that the Scrum team makes a wrong investment decision in the next Sprint Planning. As the saying goes: Love the customers' problem, not your solution.)

- **Copy & paste PO:** The Product Owner creates Product Backlog items 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: Refining Product Backlog items is a team exercise. Moreover, using practices like user stories templates helps everyone understand the Why, the What, and the How. Remember Karl Popper: "Always remember that it is impossible to speak in such a way that you cannot be misunderstood.")

- **Dominant PO:** The Product Owner creates Product Backlog items by providing not just the 'Why' but also the 'How' and the 'What.' (Just stick with the Scrum Guide and its built-in checks & balances: The Developers answer the 'How' question–the technical implementation–, and both the team and the Product Owner collaborate on the 'What' question: What scope is necessary to achieve the desired purpose?)

- **Prioritization by proxy:** A single stakeholder or a committee of stakeholders prioritizes the Product Backlog. (The strength of Scrum is building on the solid position of the Product Owner. The Product Owner is the only person to decide what tasks become Product Backlog items. Hence, the Product Owner also decides on ordering the Product Backlog. Take away that empowerment, and Scrum turns into a pretty powerful waterfall 2.0 process.)

- **100% in advance:** The Scrum team creates a Product Backlog covering the complete project or product upfront because the release scope is believed to be limited. (Let me ask a question: How can you be sure to know today what to deliver six months from now when you work on a complex, adaptive problem? Isn't not being able to predict the future the reason that you employ Scrum in the first place?)

- **Over-sized:** The Product Backlog contains more items than the Scrum team can deliver within three to six sprints. (This practice very likely will result in wasted efforts: You will refine work items that the Developers will never turn into Increments. Moreover, by investing all the efforts upfront, you may fall victim to the sunk cost fallacy, delivering less value than possible.)

- **Outdated issues:** The Product Backlog contains items that haven't been touched for several weeks or more. (That is typically the length of three 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 detailed and estimated:** All Product Backlog items are entirely detailed and estimated. (That is too much upfront work and bears the risk of misallocating the Scrum team's time. The refinement of Product Backlog items is a continuous effort only to the point where the Scrum team feels comfortable turning these items into Increments.)

- **Component-based items:** The Product Backlog items are sliced horizontally based on components instead of vertically based on end-to-end functionality. (This may be either caused by your organizational structure, an effect called Conway's law. In this case, overcome this organizational debt by moving to cross-functional teams to improve the Scrum team's delivery ability. Otherwise, the Scrum team should invest in a workshop on writing better Product Backlog items.)

- **Missing acceptance criteria:** There are Product Backlog items that need additional acceptance criteria without listing them. (It is unnecessary to have all acceptance criteria ready at the beginning of the refinement cycle, although they would make the task much more accessible. However, all Product Backlog items need to meet the Definition of Done and—probably—specific requirements at the work item level. If the Definition of Done does not provide the latter ones, the now necessary acceptance criteria shall result from the refinement process.)

- **No more than a title:** The Product Backlog contains items that comprise little more than a title. (Creating noise by adding numerous "reminders" to the Backlog, thus obfuscating the signal it shall provide, will diminish the Scrum team's capability to create valuable Increments for the customers. Ideas do not belong in the Product Backlog; they are part of the product discovery system.)

- **User story author:** The Product Owner invests too much time upfront in creating Product Backlog items, making them too detailed. (If a work item looks complete, the Developers might not see the necessity to get involved in further refinement. This way, a "fat" Product Backlog item reduces the team's engagement level, 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.)

- **No research:** The Product Backlog contains few to no spikes or time-boxed research tasks. (This effect often correlates with a Scrum team spending too much time discussing future problems instead of researching them with a spike as part of a continuous Product Backlog refinement process.)

- **Last minute Product Backlog**: The Product Owner and the rest of the Scrum team do not invest adequately in Product Backlog creation and refinement. Consequently, they rush to fill the Product Backlog immediately before the Sprint Planning to ensure there is enough work for the upcoming Sprint. (Garbage in, garbage out. This approach points to a fundamental misunderstanding or ignorance regarding the purpose of Scrum: when to use it, and what Scrum's critical success factors creating value for everyone involved are.)

- **Involving the Scrum team—why?** The Product Owner is not involving the entire Scrum team in the Product Backlog refinement process and instead relies on just the "lead engineer" and a designer. Moreover, the Developers are okay with this approach as it allows them to write more code which they prefer over figuring out what is worth building. (When trying to solve a complex problem, there are no experts but many competing ideas. Therefore, limiting the active participants in refinement activities to a few team members instead of the whole Scrum team increases the risk of falling victim to confirmation bias as the diversity of opinion is artificially limited.)

Sprint Planning Anti-Patterns

The number two area on my list of Product Owner anti-patterns is the Sprint Planning itself:

- **What are we fighting for?** The Product Owner cannot align the business objective of the upcoming Sprint with the Product Goal and overall product vision. Consequently, the Scrum team fails to create a meaningful Sprint Goal. (A serious objective answers the "What are we fighting for?" question. It is also a negotiation between the Product Owner and the Developers to a certain extent. The objective is focused and measurable, as the Sprint Goal and the Developers' forecast go hand in hand.)

- **No business objective, no Sprint Goal, just random stuff:** The Product Owner proposes a "direction" for the following Spring that resembles a random assortment of tasks, providing no cohesion. Consequently, the Scrum Team does not or cannot create a Sprint Goal. (If picking the work first and trying to create a Sprint Goal after the fact is the natural way of finishing your Sprint Planning, you probably have outlived the usefulness of Scrum as a product development framework. If you already precisely know what is creating the most value during the upcoming Spring, you no longer need to use Scrum. Depending on the maturity of your product, Kanban may prove to be a better solution in this case. Otherwise, the randomness may signal a weak Product Owner who listens too much to stakeholders instead of aiming to accomplish the Product Goal, composing and ordering 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 Product Backlog items that have not been refined. (Principally, it is the prerogative of the Product Owner to make such kinds of changes to ensure that the Developers work only on the most valuable tasks 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 ordering the Product Backlog and improving team communication.

Or the Product Owner needs support to deal with pushy stakeholders.)

- **Output focus:** The Product Owner pushes the Developers to take on more tasks than they could realistically handle. Probably, the Product Owner is referring to former team metrics such as velocity to support their desire. (This is also a road to becoming a feature factory and deserves attention from the team's Scrum Master. It violates the Developers' prerogative to pick Product Backlog items for the Sprint Backlog and Scrum Values.)

- **No preparation:** The Product Owner does not invest adequately in continuously refining an actionable Product Backlog in collaboration with the Developers. (The Product Backlog needs to represent the best possible use of the Developers' time from a customer value perspective at any given moment. In other words, your Scrum Team's Product Backlog has to be actionable 24/7. By my standards, you need to be capable of running a meaningful Sprint Planning instantly. Preparing a few basic Product Backlog items an hour before the beginning of the Sprint Planning is not enough.)

Sprint Anti-Patterns

Another area prone to Product Owner anti-patterns is the Sprint:

- **Absent PO**: The Product Owner is absent most of the Sprint and is not available to answer questions of the Developers. (As the Sprint Backlog is emergent and the Developers may identify necessary new work or the scope of previously identified work may need to be adjusted, the Product Owner's absence can leave the Developers in the dark, risking the accomplishment of the Sprint Goal.)

- **PO clinging to tasks**: The Product Owner cannot let go of Product Backlog items once they become part of the Sprint Backlog. For example, the Product Owner increases the scope of a work item or changes acceptance criteria once the Developers select this Product Backlog item for the Sprint Backlog. (There is a clear line: before a Product Backlog item becomes part of the Sprint Backlog, the Product Owner is responsible. However, once it moves from one backlog to the other, the Developers become responsible. If changes become acute during the Sprint, the team will collaboratively decide how to handle them.)

- **Delaying PO**: The Product Owner does not provide feedback on work items from the Sprint Backlog once those are done. Instead, they wait until the end of the Sprint. (The Product Owner should immediately inspect Product Backlog items that meet the acceptance criteria. Otherwise, the Product Owner will create an artificial queue within the Sprint, unnecessarily increasing the cycle time. This habit also puts reaching the Sprint Goal at risk. **Note**: Inspecting Product Backlog items does not equal some sort of work acceptance or quality gate. There is no such thing in Scrum. Once a Product Backlog item meets the Definition of Done, it can be released into the hands of users.)

- **Sprint stuffing**: The Developers accomplished the Sprint Goal early, and the Product Owner is pushing them hard to accept new work from the Product Backlog to fill the "void." (The Scrum Team decided collaboratively on the Sprint Goal, and the Developers committed to delivering. Consequently, it is the prerogative of the Developers to decide on the composition of the Sprint Backlog. Should they manage to accomplish the Sprint Goal before the Sprint's end, it is their sole decision to fill the remaining time. Accepting new work from the Product Backlog may be one way of filling the remaining Sprint time-box. This also applies to refactoring parts of the tech stack, exploring new technology that might be useful, fixing some bugs, or sharing knowledge with fellow teammates. Scrum is not in the business of maximizing the utilization rates of team members. Achieving the Sprint Goal is sufficient.)

- **Sprint cancellations without consultation**: The Product Owner deems the Sprint Goal obsolete and cancels the Sprint without consulting the Scrum Team. (Technically, it is the prerogative of the Product Owner to cancel Sprints. However, the Product Owner should not do this without a serious cause. Moreover, the Product Owner should never abort a Sprint without consulting the rest of the team. Maybe, someone has an idea on how to save the Sprint Goal, provided it is still useful? Misusing the cancellation privilege indicates a serious team collaboration issue and a lack of commitment to live Scrum Values.)

- **No Sprint cancellation**: The Product Owner does not cancel a Sprint whose Sprint Goal can no longer be achieved or has become obsolete. (If the Scrum Team identified a unifying Sprint Goal, for example, integrating a new payment method, and the management then abandons that functionality mid-sprint, continuing working on the Sprint Goal would be wasteful. In such a case of obsolescence, the Product Owner has to consider canceling the Sprint.)

PO Anti-Patterns during the Daily Scrum

By comparison to other Scrum events, the Daily Scrum is remarkably resilient to Product Owner anti-patterns:

- **Assignments:** The Product Owner assigns tasks directly to team members. (The Developers self-organize their work: "No one else tells them how to turn Product Backlog items into Increments of value." Source: Scrum Guide 2020.)

- **Additional work:** The Product Owner or even other stakeholders attempt to introduce new work to the current Sprint during the Daily Scrum. (This behavior may be acceptable for high priority bugs, although the Developers should be aware of those before the Daily Scrum. Otherwise, the composition of the Sprint Backlog is the sole responsibility of the Developers: they may accept new work during the Sprint; however, that is their sole decision.)

Sprint Review Anti-Patterns

Finally, there is the Sprint Review. Despite that it is an outstanding opportunity for the Product Owner to improve the collaboration with both stakeholders and Developers and figure out collectively in what direction to take the product next, some Product Owners do not get the message:

- **Selfish PO:** Product Owners present "their" accomplishments to the stakeholders. (Scrum is a remarkably egalitarian affair: No one has any authority to tell teammates what, when, or how to do something. There are no sub-roles on Scrum teams, and there is no hierarchy. Instead, Scrum builds everything on self-management and collective responsibility—the Scrum team wins together, and the Scrum team loses together. Remember the old saying: There is no "I" in "team?")

- **"Acceptance" by the PO:** The Product Owner uses the Sprint Review to "accept" Product Backlog items that Developers consider "done." (A feedback loop—did the Developers deliver the agreed-upon functionality?—is valuable. However, Product Owners should also decouple this feedback loop from the Sprint Review. Instead, Product Owners should communicate with the Developers whenever needed or when work items meet the Definition of Done.)

- **Unapproachable PO:** The Product Owner is not accepting feedback from stakeholders or the Developers. (I get it: Loving your solution over the customers' problems feels good. Moreover, a bit of confirmation bias may prevent our Product Owners from getting distracted from pursuing their beloved solution. But unfortunately, they do not get paid to roll out their solution. This "living in their PO bubbles" approach hence violates the prime purpose of the Sprint Review event: Figure out collaboratively whether the Scrum team is still on track to meeting the Product Goal—which requires openness on the side of the Product Owners in the first place.)

Conclusion

Admittedly, the Product Owner role is the most challenging Scrum role, and the higher the expectations are, the easier it is to fail them.

We do not get paid as a Scrum team to practice Scrum. No stakeholder cares for that aspect. However, we do get paid to improve the lives of our customers while contributing to the sustainable business of our organization. In that model, the Product Owner is the linchpin of value creation as they decide where to take the Scrum team next. No other role in Scrum can contribute to mediocre outcomes like the Product Owner—garbage in, garbage out.

Fortunately, there is hope as continuous improvement also applies to the Product Owner role. This list of Product Owner anti-patterns may be a starting point.

Scrum Master Anti-Patterns

Introduction

The reasons Scrum Masters violate the spirit of the Scrum Guide are multi-faceted. Typical Scrum Master anti-patterns run from ill-suited personal traits to complacency to pursuing individual agendas to frustration with the team itself.

Read on and learn in this post on Scrum anti-patterns how you can identify if your Scrum Master needs support from the team.

20
Scrum Master Anti-Patterns
© Stefan Wolpers | stefan@age-of-product.com 2019

The Scrum Master According to the Scrum Guide

Before we start dissecting probable reasons and manifestations of Scrum Master anti-patterns let us revisit how the Scrum Guide defines the role of the Scrum Master:

The Scrum Master is accountable for establishing Scrum as defined in the Scrum Guide. They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization.

The Scrum Master is accountable for the Scrum Team's effectiveness. They do this by enabling the Scrum Team to improve its practices, within the Scrum framework.

Scrum Masters are true leaders who serve the Scrum Team and the larger organization.

Source: Scrum Guide 2020.

The keystone of the definition of the Scrum Master role is the leadership aspect. (Too bad, it is no longer called "servant leadership;" I found that description to be better suited.) Unfortunately, in most cases of Scrum Master anti-patterns, it is precisely this idea that an individual is not meeting. (See also the detailed lists of services rendered to the Product Owner, the Developers, and the organization by the Scrum Master as defined by the Scrum Guide.)

Possible Reasons Why Scrum Masters Leave the Path

The reasons Scrum Masters violate the spirit of the Scrum Guide are multi-faceted. They run from ill-suited personal traits to pursuing their agendas to frustration with the Scrum team. Some often-observed reasons are:

- **Ignorance or laziness:** One size of Scrum fits every team. Your Scrum Master learned the trade in a specific context and is now rolling out precisely this pattern in whatever organization they are active, no matter the context. Why go through the hassle of teaching, coaching, and mentoring if you can shoehorn the "right way" directly into the Scrum team?

- **Lack of patience:** Patience is a critical resource that a successful Scrum Master needs to field in abundance. But, of course, there is no fun in readdressing the same issue several times, rephrasing it probably, if the solution is so obvious—from the Scrum Master's perspective. So, why not tell them how to do it 'right' all the time, thus becoming more efficient? Too bad that Scrum cannot be pushed but needs to be pulled—that's the essence of self-management.

- **Dogmatism:** Some Scrum Masters believe in applying the Scrum Guide literally, which unavoidably will cause friction as Scrum is a framework, not a methodology. Nevertheless, teaching Scrum that way feels good:

    - Team members come and ask for help; now, the Scrum Master has a purpose.

    - When Scrum team members follow the rules, the Scrum Master has influence or authority.

    - Being among the chosen few who interpret the Scrum Guide "correctly" secures status and respect among teammates and the broader organization.

    - The Scrum Master may easily attribute the Scrum team's progress or success to their teaching; now, they also have proof regarding their mastery of Scrum.

    - Finally, their mastering of Scrum is a convincing argument for the organization to keep the dogmatic Scrum Master on the pay role; apparently, the Scrum teams need an interpreter of the Scrum Guide to reap the framework's benefits.

- **Laissez-faire turned into indifference:** Pointing the team in a direction where the team members can find a solution for an issue is good leadership. However, letting them run without guidance probably turns into indifference, or worse, into an I-do-not-care mentality.

- **Dolla, dolla, bill ya'll—the Scrum Master imposter:** Secretly, the Scrum Master is convinced that this Scrum thingy is a fad but recognizes that it is a well-paid one: "I will weather the decline in demand for project managers by getting a Scrum Master certificate. How hard can this probably be—the manual is merely 13 pages?" This conviction will bring out their true colors over time inevitably.

- **Pearls before swine — the frustrated Scrum Master:** The Scrum Master has been working their butt off for months, but the team is not responding to the effort. The level of frustration is growing. There are many potential reasons for a failure at this level: the lack of sponsoring from the C-level of the organization, a widespread belief that 'Agile' is just the latest management fad, and thus ignorable. The team composition is wrong. There is no psychological safety to address the team's problems, and the company culture values neither transparency nor radical candor. Or individual team members harbor personal agendas unaligned with the team's objective—just to name a few. If the Scrum Team does not manage to turn its ship around, it will probably lose its Scrum Master. Please note that the Scrum Master cannot solve this issue by themselves. Fixing this issue is an effort of the whole Scrum Team.

- **The tactical Scrum Master:** These Scrum Masters drank HR's cool-aid that Scrum Master is a position, not a role. Moreover, there is a career path from Junior Scrum Master to VP of Agile Coaching. Consequently, they constrain their work strictly to the Scrum Team level until being promoted.

- **Lastly, the rookie:** If you apply Occam's razor to the situation, you may conclude that your Scrum Master has not yet defected to the dark side. They might merely be inexperienced. Given that we all need to learn new skills regularly, cut some slack in this case, and reach out to support the learning effort. Remember, it is a journey, not a destination, and you do not travel alone.

The Scrum Master as Agile Manager

In my eyes, 'agile management' is an oxymoron. The primary purpose of any agile practice is empowering those closest to a problem with finding a solution. In other words, the team shall become self-organizing over time, thus contributing at an appropriate level to the business agility of the organization. Self-organizing teams need coaches, mentors, and servant leaders, however, not a manager in the Taylorist meaning of the word.

Watch out for the Scrum Master anti-patterns corresponding to this 'agile manager' attitude:

- **Agile management:** Self-organization does not mean the absence of management: Why would a Scrum Master or Scrum team assume, for example, responsibility for pay-role? Would that help with creating value for the customer? Probably less so. Hence, being a self-organizing team does not mean the absence of management per se.

However, it does mean that there is no need for micromanagement comparable to practices at a General Motors assembly plant in 1926. The Scrum Master is neither a supervisor nor a dispatcher.

- **Running meetings by allowing someone to speak:** When team members seek eye contact with the Scrum Master before speaking out, the Scrum Master has already left the facilitation role in favor of the supervisor mode.

- **Keeping the Scrum team dependent:** In this scenario, the Scrum Master pampers the Scrum team to a level that keeps the team dependent on their services:

    - Organizing meetings.
    - Purchasing stickies and sharpies.
    - Taking notes.
    - Updating Jira—you get the idea of this service level.

    (More critical, however, is when the Scrum Master decides to keep the team in the dark about principles and practices to secure their job. This behavior is only a tiny step away from the dark side.)

- **Pursuing flawed or dangerous metrics:** While utilizing predefined Jira reports is probably a sign of ignorance or laziness, keeping track of individual performance metrics—such as story points per Developer per Sprint and reporting those to that person's line manager—is a critical warning sign. That is a classic supervisor hack to reintroduce command & control through the back door. It inevitably leads to cargo-cult Scrum.

- **Escalating under-performance:** During the Sprint, the Scrum Master reports to the management whether the Scrum team will meet the current forecast or not. I took this from a job offer I received: "You will coordinate and manage the work of other team members, ensuring that timescales are met, and breaches are escalated."

- **Focusing on team harmony:** The Scrum Master sweeps conflict and problems under the rug by not using Sprint Retrospectives to address those openly. This behavior is often a sign of bowing to politics. Instead, the Scrum Master uses manipulation to meet organizational requirements that oppose Scrum values and principles. If the organization values its underlings for following the 'rules' instead of speaking the truth, why would you run Retrospectives in the first place? (A 'Scrum Master' participating in cargo-cult Scrum is again a supervisor than an agile practitioner.)

Scrum Master Anti-Patterns by Scrum Events

The Sprint Planning

The following anti-patterns focus on the Sprint Planning:

- **100% utilization:** The Developers regularly bow to the hard-pushing Product Owner: "Last Sprint, you delivered 25 story points, and now you are choosing only 17?" Consequently, they accept more issues into the Sprint Backlog than they can stomach without the Scrum Master's intervention. They do not address that this is a disrespectful behavior, negating the Developer's prerogative of self-management and ignoring their need for slack time. (The Scrum team's effectiveness will be significantly impeded if the team does not address technical debt every Sprint. It will also suffer if there is no time for pairing, for example, or supporting each other. A level of 100% utilization always reduces the Scrum team's long-term productivity; it is classic Taylorist line management thinking. If 50% of all issues spill over to the next Sprint at the end of a Sprint, and this is a pattern, then your Scrum team is not practicing Scrum. Probably, it is a sort of time-boxed Kanban—which would be okay, too. Just make up your mind about how you intend to improve your customers' life. Perhaps, Kanban would be a good choice.)

- **Unrefined Sprint Backlog items:** The Scrum Master does not address accepting "unrefined" Product Backlog issues into the Sprint Backlog. This choice is a sure way the Developers will probably not accomplish the Sprint Goal, rendering a core Scrum principle useless: reliably providing valuable, potentially shippable Product Increments every single Sprint. (This refers to regular work, not emergencies.)

The Sprint

The following anti-patterns focus on the mishandling of the Sprint itself:

- **Flow disruption:** The Scrum Master allows stakeholders to disrupt the flow of the Scrum Team during the Sprint. There are several possibilities for how stakeholders can interrupt the team's flow during a Sprint. Any of the following examples will impede the team's productivity and might endanger accomplishing the Sprint Goal:

    - The Scrum Master has a laissez-faire policy as far as access to the Developers is concerned. Notably, they are not educating stakeholders on the negative impact of disruptions and how those may endanger accomplishing the Sprint Goal. Note: I do not advocate that Scrum Masters shall restrict stakeholders' access to team members in general.

    - The Scrum Master does not oppose line managers taking team members off the Scrum Team, assigning them to other tasks.
    - The Scrum Master does not oppose line managers adding members to the Scrum Team without prior consultation of the team members. (Preferably, the Scrum Team members should decide who is joining the team.)

    - Lastly, the Scrum Master allows stakeholders or managers to turn the Daily Scrum into a reporting session. (This behavior will impede the Developer's productivity by undermining their self-management while reintroducing command & control through the backdoor.)

  - Assigning work to Developers: The Scrum Master does not prevent the Product Owner—or anyone else—from assigning tasks directly to Developers. (They organize themselves without external intervention. And the Scrum Master is the shield of the Scrum team in that respect.)

- **Defining technical solutions:** An engineer turned Scrum Master is now 'suggesting' how the Developers implement issues. (Alternatively, the Product Owner or an outsider is pursuing the same approach, for example, a technical lead.)

- **Lack of support:** The Scrum Master does not support team members that need help. Often, Developers create tasks an engineer can finish within a day during their planning sessions. However, if someone struggles with such a job for more than two days without voicing that they need support, the Scrum Master should address the issue and offer their support. Making this issue visible is also the reason for marking tasks on Sprint boards with red dots each day if work items do not move to the next column. (In other words: we are tracking the work item age.)

The Retrospective

The final set of anti-patterns addresses the Sprint Retrospective:

- **Waste of time**: The Scrum 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, if appropriately facilitated.)

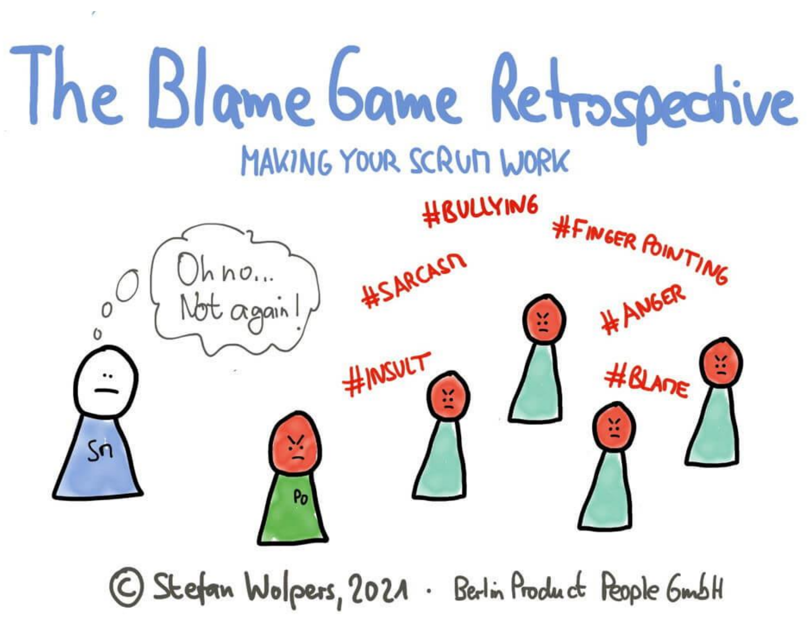
- **Groundhog day:** The Sprint Retrospective never changes in composition, venue, or length. There is a tendency in this case that the Scrum team might revisit the same issues repeatedly–it's Groundhog Day without the happy ending, though.

20 Scrum Master Anti-Patterns
Groundhog Day
I want you to note what went well and what can be improved...
WTF?
NOT AGAIN...
This retrospective format is getting us nowhere...
SM
© Stefan Wolpers 2019 · Berlin Product People GmbH

- **Let's have the retro next Sprint:** The team postpones the Retrospective to 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 Scrum team can focus on the new Sprint. Additionally, the Scrum team shall inspect their way of working and identify opportunities to make the next Sprint more effective and fun. This is why we have the Retrospective before the Sprint Planning of the upcoming Sprint. Moreover, postponing the Retrospective into the next Sprint may interrupt the team's flow and leave one or more critical team issues unattended for the length of a Sprint. Also, it is a slippery slope. If postponing Retrospectives becomes an acceptable approach, why have them in the first place?)

- **#NoRetro:** There is no Retrospective as the team believes there is nothing to improve, and the Scrum Master accepts this notion. (There is no such thing as an agile Nirwana where everything is perfect. As people say: becoming agile is a journey, not a destination, and there is always something to improve.)

- **#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 fails together. The blame game documents both the failure of the Scrum Master as the facilitator of the Retrospective and the Scrum team's lack of maturity and communication skills.)



- **Bullying is accepted:** One or two team members dominate 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 their feedback free from third-party influence. If some of the team members dominate the conversation and probably even bully or intimidate 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 less valuable. It is the primary responsibility of the Scrum Master as a facilitator to ensure that everyone will be heard and has an opportunity to voice their 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. (Several opportunities in Scrum address stakeholders' communication and information needs: the Sprint Review, listening in at the Daily Scrum, probably even the Product Backlog refinement, not to mention opportunities of having a conversation in Zoom, at water coolers, over coffee, or during lunchtime. If that spectrum of possibilities is not sufficient, consider having additional meetings if your team deems them necessary. For example, there is the opportunity to have a meta-level Retrospective, explicitly including the Scrum team's stakeholders. However, the Retrospective as a Scrum team-internal event is off-limits to stakeholders.)

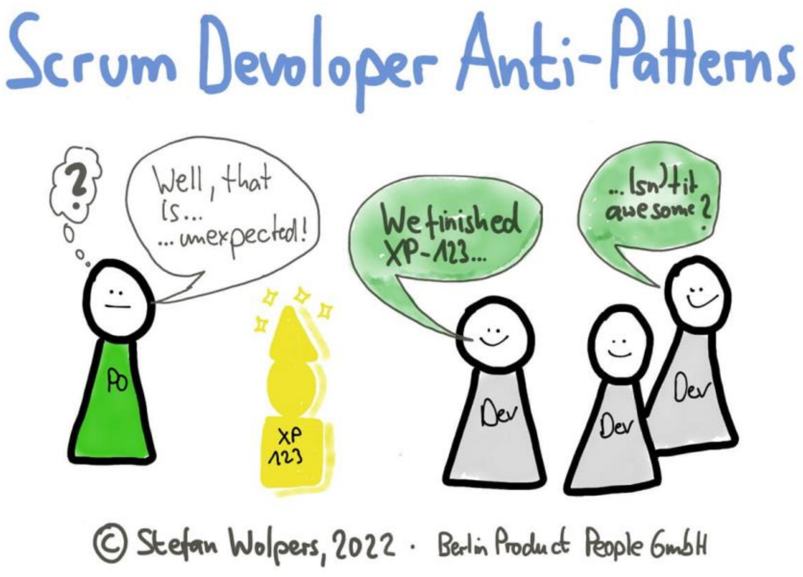
Conclusion

There are plenty of possibilities to fail as a Scrum Master. Sometimes, it is the lack of organizational support. Some people are not suited for the job. Others put themselves above their teams for questionable reasons. Some Scrum Masters simply lack feedback from their Scrum Teams and stakeholders. Whatever the case may be, though, try and lend your Scrum Master in need a hand to overcome the misery. Scrum is a team sport.

Scrum Developer Anti-Patterns

Introduction

After covering the anti-patterns of the Scrum Master, the Product Owner, and the stakeholders, this article addresses Scrum Developer anti-patterns, covering all Scrum Events and the Product Backlog artifact. Continue reading and learn more about what to look out for if you want to support your teammates who build the Increment.


Scrum Developer Anti-Patterns
Well, that is... ...unexpected!
We finished XP-123...
...Isn't it awesome?
PO
XP 123
Dev
Dev
Dev
© Stefan Wolpers, 2022 · Berlin Product People GmbH

The Role of the Developers in Scrum

According to the Scrum Guide, the "Developers are the people in the Scrum Team that are committed to creating any aspect of a usable Increment each Sprint:"

The specific skills needed by the Developers are often broad and will vary with the domain of work. However, the Developers are always accountable for:

- *Creating a plan for the Sprint, the Sprint Backlog;*
- *Instilling quality by adhering to a Definition of Done;*
- *Adapting their plan each day toward the Sprint Goal; and,*
- *Holding each other accountable as professionals.*

Also, Developers enjoy complete autonomy regarding the technical side of their work:

For each selected Product Backlog item, the Developers plan the work necessary to create an Increment that meets the Definition of Done. This is often done by decomposing Product Backlog items into smaller work items of one day or less. How this is done is at the sole dis-

cretion of the Developers. No one else tells them how to turn Product Backlog items into Increments of value.

Source: Scrum Guide 2020.

(Check out the complete Scrum Guide 2020 list on the Developers by downloading the Scrum Guide Reordered.)

The term "Developer" seems to limit the role to technical people, for example, software engineers. However, in the spirit of the Scrum Guide, the term Developer is much more inclusive. In my experience, given proper support from the Scrum Master and the Product Owner, even lawyers and marketers can get comfortable with this designation when utilizing Scrum.

So, let's dive into an array of common Scrum Developer anti-patterns that signal Scrum Masters that their teammates need support.

Developer Anti-Patterns by Scrum Events

The following list of Developer anti-patterns addresses all four Scrum events plus the Sprint itself:

Sprint Anti-Patterns

Most of the following Developer anti-patterns mostly result from a lack of focus or procrastination:

- **No WiP limit:** There is no work in progress limit. (The purpose of the Sprint is to deliver a potentially shippable Product Increment that provides value to the customers and thus to the organization. This goal requires focused work to accomplish the necessary work to meet the Sprint Goal by the end of the Sprint. The flow theory suggests that a team's productivity improves with a work-in-progress (WiP) limit. The WiP limit defines the maximum number of tasks the Developers can work on simultaneously. Exceeding this WiP number creates additional queues that consequently reduce the overall throughput of the Developers. The cycle time—the period between starting and finishing a ticket—measures this effect.)

- **Cherry-picking:** The Developers cherry-pick 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 may notice tickets queueing in the code-review-column, for example. It is also a sign that the Developers are not yet fully self-organizing. Look for Daily Scrum events that support this notion and address the issue during the Sprint Retrospective.)

- **Board out-of-date:** The Developers do not update tickets on the Sprint board in time to reflect the current statuses. (The Sprint board, no matter if it is a physical or digital board, is not only vital for coordinating the Developers' 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 confidence may then cause counter-measures on the side of the stakeholders. Consequently, the (management) pendulum may swing back toward traditional methods. The road back to PRINCE II is paved with abandoned boards.)

- **Side-gigs:** The Developers are 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 for the return on investment the Scrum Team is creating—is unacceptable. This behavior also signals a substantial 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 Developers are probably rather a group than a team anyway.)

- **Gold-plating:** The Developers increase the scope of the Sprint by adding unnecessary work to the Product Backlog items of the Sprint Backlog. (This effect is often referred to as scope-stretching or gold-plating. The Developers ignore 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 development time. However, there is a simple solution: the Developers and the Product Owner need to talk more often, creating a shared understanding from product vision down to the individual Product Backlog item, thus improving the trust level. If the Product Owner is not yet co-located with the Developers, now would be the right moment to reconsider. Alternatively, a distributed Scrum Team may invest more time in how to communicate synchronously and asynchronously best to improve its level of collaboration and alignment.)

- **Cutting corners to meet a deadline:** The Developers reduce product quality below the level defined in the Definition of Done to meet an imposed, probably arbitrary deadline. (The Definition of Done serves a critical role in the Scrum framework: It represents the quality standard that signals to everyone what the Developers need to accomplish to create a potentially releasable Increment. By "releasable," we mean that we can deliver the Increment into the hands of our customers and users without any legal, financial, or ethical repercussions. Watering this quality level down to meet an arbitrary deadline imposed on the Scrum team puts the team's success at risk as it introduces undone work and technical debt. Both of these consequences of neglecting the quality standard will impede the Scrum team's ability to be innovative at a later stage. Therefore, it is crucial for a Scrum team's success that processes in the organization that result in arbitrary deadlines must be addressed. While there are valid reasons for deadlines, for example, a general legal requirement is imposed, we are interested in the less transparent reasons. Has the sales department, for example, sold features unbeknownst to the Scrum team? Probably already legally enforceable based on a contract? I would consider that an unfriendly act, as the sales department is aiming for a local optimum, here: meeting a sales quota. In this case, the Scrum team would be well-advised to understand the process and its motivation and seek allies to work on overcoming this impediment in the future. An alternative scenario would be when stakeholders consider imposing deadlines as means of last resort, as the Scrum team is notoriously unreliable about forecasting the availability of critical features. Here, the use of deadline is instead a sign of—probably justifies—mistrust than a sinister move on behalf of the stakeholders. The best way of action for the Scrum team would be to embark on a journey of trust-building. The best way to do that is to deliver valuable Increment like a Swiss clockwork regularly. As the saying goes: It may take days, weeks, or months to build trust; it always takes just seconds to destroy it. Please also note that there is no business agility without technical excellence. The latter starts with adhering to the Definition of Done at all times.)

- **What technical debt?** The Developers do not demand adequate time to tackle technical debt and bugs and preserve the quality standard of the product. Instead, they fully embrace the feature factory, shipping one new feature after the other. (My rule of thumb is that the Developers shall consider allocating up to 20 % of their time to fixing bugs and refactoring the codebase. A high-quality tech stack is at the core of being able as a Scrum team to adapt the following steps to lessons learned and recent market trends. Technical excellence is a prerequisite of any form of business agility; preserving this state is a continuous process that requires a steady and substantial investment. **Read more**: Technical debt and Scrum.)

- **No slack time:** The Developers do not demand 20% slack time—or unplanned capacity—from the Product Owner. (This overlaps with the Sprint Planning, creating Sprint Goals, and the team's ability to forecast. However, it cannot be addressed early enough. If a team's capacity is always utilized at 100 %, its performance will decrease. Everyone will focus on getting their tasks done. There will be less time to support teammates or to pair. Minor issues will no longer be addressed immediately. And ultimately, the 'I am busy' attitude will reduce the creation of a shared understanding among all team members why they do what they are doing.)

Sprint Planning Anti-Patterns of the Developers

Scrum's Sprint Planning aims to align the Developers and the Product Owner on what to build next, delivering the highest possible value to customers:

- **No capacity check:** The Developers overestimate their capacity and take on too many tasks. (The Developers should instead consider everything 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 events as practices such as Product Backlog refinement, and other meetings, to name a few.)

- **Planning too detailed:** During the Sprint Planning, the Developers plan every single task of the upcoming Sprint in advance. (Don't become too granular. One-quarter of the tasks are more than sufficient to not just start with the Sprint, but also start learning. The Sprint Backlog is emergent, and doing too much planning upfront might result in waste. Scrum is not a time-boxed form of the waterfall planning practice.)

- **Too little planning:** The Developers skip planning altogether. (Skipping planning is unfortunate, as it is also an excellent opportunity to talk about how to spread knowledge among the Developers, where the architecture is heading, or whether the tools are adequate. For example, the team might also consider who will be pairing with whom on what task. The Developers' planning part is also well-suited to consider how to reduce technical debt, see above.)

- **Team leads?** The Developers do not devise a plan to deliver their forecast collaboratively. Instead, a 'team lead' does all the heavy lifting and probably even assigns tasks to individual Developers. (I know that senior developers do not like the idea, but there is no 'team lead' on a Scrum Team. **Read More**: Why Engineers Despise Agile.)

Anti-Patterns of the Daily Scrum

The Daily Scrum is an essential event for inspection and adaption, run by the Developers, and guiding it for the next 24 hours on its path to achieving the Sprint Goal. The Daily Scrum is hence the shortest planning horizon in Scrum and thus highly effective to guide the Scrum team's efforts.

Contrary to popular belief, its 15-minutes timebox is not intended to solve all the issues addressed during the Daily Scrum. It is about creating transparency, thus triggering the inspection. If an adaption of the plan or the Sprint Backlog, for example, is required, the Developers are free to handle the resulting issues at any time. In my experience, most Daily Scrum anti-patterns result from a misunderstanding of this core principle.

- **No routine:** The Daily Scrum 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 the Daily Scrum. Think of it as a spontaneous drill: don't put too much thought into the Daily Scrum, just do it. Skipping Daily Scrums can turn out to be a slippery slope: if you skip one Daily Scrums or two, why not skip every second one?)

- **Orientation lost:** The Daily Scrum serves one purpose as it answers a simple question: Are we still on track to meet the Sprint Goal? Or do we need to adapt the plan or the Sprint Backlog or both? Sometimes though, the Developers cannot answer that question immediately. (In that respect, visualizing the progress towards the Sprint Goal is useful. Removing the Developers' task of maintaining a mandatory burndown chart from the Scrum Guide a few years ago does not imply that such a visualization is useless.)

- **Cluelessness:** Developers are not prepared for the Daily Scrum. ("I was doing some stuff, but I cannot remember what. Was important, though.")

- **Status report:** The Daily Scrum is a status report meeting, and Developers are waiting in line to "report" progress to the Scrum Master, the Product Owner, or maybe even a stakeholder. (The "three Daily Scrum questions" often serve as a template for this anti-pattern.)

- **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.")

- **Planning meeting:** The Developers hijack the Daily Scrum to discuss new requirements, refine user stories, or have a sort of (Sprint) Planning Meeting, probably collaborating with the Product Owner.

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

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

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

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

- **No use of work-item age:** A Developer experiences difficulties in accomplishing an issue over several consecutive days and nobody is offering help. (Often, this result is a sign that people either may not trust each other or do not care for each other. Alternatively, the workload of the Developers has reached an unproductive level as they no longer can support each other. Note: Of course, the Scrum Guide does not mention the 'work item age.' However, it has proven to be a useful practice.)

- **There are never any impediments:** No one reports any obstacles; no one needs any help or support from their teammates. (Congratulations on your seemingly well-oiled machine! However, maybe, Developers do not feel safe to address issues, challenges, or problems?)

Developers Anti-Patterns concerning the Sprint Review

Are we still on track to accomplish the Product Goal? Moreover, how did the previous Sprint contribute to our Scrum team's mission? Answering these questions and adapting the Product Backlog in a collaborative effort of the Scrum Team with internal and external stakeholders is the purpose of the Sprint Review. Some common anti-patterns are:

- **Death by PowerPoint**: Participants of the Sprint Review 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 discovery. The Sprint Review is not a "demo," carefully avoiding all obstacles to preserve the illusion of progress and control. Instead, it is an essential opportunity to inspect what the Scrum team accomplished, receive valuable feedback, and adapt the Product Backlog collectively. It is about creating transparency on the state of the progress toward the Product Goal.)

- **Same faces again**: It is always the same Developers who participate in the Sprint Review; however, they are not all Developers. (Unless the organization scales Scrum based on LeSS or Nexus and we are talking about the overall Sprint Review, this Sprint Review anti-pattern of limited attendance of Scrum team members is a bad sign. The Sprint Review needs all Scrum team members on deck to maximize the learning. The challenge is that you cannot enforce your teammates' participation either. Instead, make it interesting enough that everyone wants to participate. If this is not happening, you should ask yourselves how you have contributed to this situation in the past. It is lucky a coincidence that an event tailored to this need follows immediately after the Sprint Review — the Retrospective.)

- **Side gigs**: The Developers worked on issues outside the Sprint Goal, and the Product Owner learned about those for the first time during the Sprint Review. (This Sprint Review anti-pattern is truly "rocking the boat." Focus, commitment, and openness — just to name the most apparent Scrum values violated here — are the first principles for collaboration among Scrum team members. Anything that the Developers want to address during the Sprint needs to be communicated during the Sprint Planning. However, a particular case is when the Product Owner is usually so pushy about shipping new features that there is little or no time to attend to refactoring or bug fixing unless the Developers tackle these jobs under the radar. In this situation, I would have sympathy for the approach. Nevertheless, the Scrum team needs to fix this problem. Generally, allocating 20 % of the team's capacity to the before-mentioned tasks could be a start.)

- **Undone is the new "done:"** More often than not, the Developers show work items that are not "done." (There is a good reason to show unfinished work on some occasions. For example, to provide transparency to stakeholders on essential yet challenging tasks. However, regularly reporting to Sprint Review attendees on partially finished work violates the concept of "Done," one of Scrum's first principles. There is no need for a successful Scrum team to demonstrate to stakeholders that they are worth their pay-cheques.)

Sprint Retrospective Anti-Patterns of Developers

What event could better embody Scrum's principle of empiricism than the Sprint Retrospective? I assume all peers agree that even the simplest form of a Retrospective—if only held regularly—is far more helpful than having a fancy one once in a while, not to mention having none. Some notorious anti-patterns from the Developers' side include:

- **#NoRetro:** There is no Retrospective as the team believes there is nothing to improve, and the Scrum Master accepts this notion. (There is no such thing as an agile Nirwana where everything is perfect. As people say: becoming agile is a journey, not a destination, and there is always something to improve.)

- **Dispensable buffer**: The Developers cancel Retrospectives if more time is needed to accomplish the Sprint Goal. (The Retrospective as a Sprint emergency reserve is a common sign of cargo cult Scrum. 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 a Retrospective to achieve a Sprint Goal clearly shows that the team does not understand basic principles, such as empiricism and continuous improvement. If the Scrum team repeatedly does not meet the Sprint Goal, it should inspect what is happening here. Guess which Scrum event is designed for that purpose?)

- **Not improving the Definition of Done**: The Developers do not advocate regularly revisiting the Definition of Done and discussing whether it needs to improve. (As the saying goes: Standing still is moving backward. When the Scrum team increasingly better understands how to solve its customers' problems, this advance in knowledge needs to be represented in a regularly adapted Definition of Done. Otherwise, the Scrum team would risk diminishing its contribution to the organization's overall path to business agility.)

- **Extensive whining**: The Developers use the Retrospective primarily to complain about the situation and assumes the victim's role. (Change requires reflection, and occasionally it is a suitable 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.)

Anti-Patterns at the Product Backlog Level

Scrum is a tactical framework to build products, provided you identify what is worth making in advance. But even after a successful product discovery phase, you may struggle to create the right thing in the right way if your Product Backlog is not up to the job—garbage in, garbage out. Some forms by which Developers contribute to mediocre team success are:

- **No time for refinement:** The Developers do not have enough Product Backlog refinement sessions, resulting in a low-quality backlog. (The Scrum Guide no longer advises spending up to 10% of the Developers' time on the Product Backlog refinement. However, investing in good refinement is still a sound business decision: Nothing is more expensive than a feature that is not delivering any value.)

- **Too much refinement:** The Scrum team has too many refinement sessions, resulting in a too detailed Product Backlog. (Too much refinement isn't healthy either. There is a moment when the marginal return of an additional refinement effort is zero or probably even negative—think analysis-paralysis. The only way for a Scrum team to understand whether the previous validation of the underlying hypotheses of a new feature is correct is to build and ship this thing. There is no way to figure this out at the green table, refining and discussing the issue endlessly.)

- **Too much estimating:** The Developers even estimate sub-tasks. (That looks like accounting for the sake of accounting to me. So don't waste your time on that. Remember: the purpose of estimating is to identify misalignment among the Developers regarding the What and How of items from the Product or Sprint Backlog. **Read more**: Estimates Are Useful, Just Ditch the Numbers.)

- **Over-sized Product Backlog:** The team creates a massively comprehensive Product Backlog. (Building an "inventory of work-items" is an anti-pattern. In a complex environment with a lot of uncertainty, though, this upfront investment likely leads to waste in the form of Product Backlog items that are refined but will never be released as more valuable items are later identified. The Developers should point to this challenge and support the Product Owner in keeping the Product Backlog concise and actionable; an inventory of three to six Sprints is typically more than adequate.

- **A submissive Scrum team:** The Developers submissively follow all demands of the Product Owner. (Challenging the Product Owner whether their selection of work items is the best use of the Developers' time is the noblest obligation of every team member: Why shall we do this? Scrum does not work without everyone living up to the checks & balances built into the framework. It is easy to fall in love with "your solution" over addressing the real customer problem. If Developers simply make whatever the Product Owner "suggests," the overall value created by the Scrum team will likely be below its potential. That is why you want missionaries on your team, not just mercenaries.)

Conclusion

Given the essential role Developers have to live up to make Scrum successful, it is no surprise that there are plenty of anti-patterns to observe. The good news is, though, that many of those are entirely under the control of the Scrum Team. All it takes to tackle these Developer anti-patterns for the team is starting to inspect and adapt. Why not use your next Retrospective for this purpose?

Scrum Master Anti-Patterns Derived from Job Ads

Job ads for scrum master or agile coach positions reveal a great insight into an organization's progress on becoming agile. Learn more about what makes job ads such a treasure trove with the following 22 scrum master anti-patterns. To gain these, I analyzed more than 50 job ads for scrum master or agile coach positions.

22 Scrum Master Anti-Patterns from Job Ads
TechPO
SCAPE
GOAT
CSM®
Don't Kill the Messenger!
ERSATZ
PM
P(O)roxy
minion
¿WTF?
The Whip
PSYCHIC
SUPER SM
MOM or
Secretary
Mr Bean
(Counter)
© S. Wolpers 2017 | Age-of-Product.com

Analyzing a Job Advertisement for a Scrum Master or Agile Coach position

Probably, you are considering a position as a scrum master or agile coach in a particular organization. I suggest that before going all in (the application process), you should consider analyzing the job description for scrum master anti-patterns first.

How Large Organizations Create Job Ads

Usually, the organization's HR department will create the final text of the job advertisement and post it to the chosen job sites. Hopefully, and depending on their process and level of collaboration (and agile mindset) in the organization, the team for which the new position was advertised may have participated in creating the job ad. This certainly avoids advertising a wrong description to prospective candidates.

Too often, however, advertisements may read like a copy and paste from positions that an organization's HR believes to be similar to that of a scrum master (for example, a project manager). Or, sometimes, the HR department copies from other scrum master job ad which they believe correctly reflect the requirements of the organization. So, don't be too surprised to see a job advertisement that reads like a list of scrum master anti-patterns.

Red Flags: A Sign of Cargo Cult Agile or just on Organization at the Beginning of the Agile Transition?

This is often the case when an organization's HR does not have a lot of experience in hiring agile practitioners because they are in the early stages of the agile transition. Therefore, an unusual job description does not imply that the organization is not trying to become agile, it may just mean that the HR department has not yet caught up with the new requirements. Such an advertisement can actually help raise the topic and be of benefit during the job interview.

Be aware, however, that if an organization which claims to be agile is using this kind of advertisement despite being well underway on its agile transition, it then raises a red flag: miscommunication in the hiring process may indicate deeper issues or problems at the organizational level. It could be as critical as someone at management level, to whom the new scrum master would likely report, having no clue what becoming agile is all about.

Scrum Master Anti-Patterns from Job Ads in 22 Examples

As mentioned previously, here are some examples of scrum master advertisement anti-patterns (from more than 50 actual job descriptions) that should raise a red flag:

1. **Ersatz PM**: The scrum master position is labeled as "Project manager/Scrum master", "Agile Project Manager", or "Agile scrum master". (Are there un-agile scrum masters mentioned in the Scrum Guide?)

2. **The whip**: The scrum master is expected to communicate the company priorities and goals. (Product backlog-wise priorities are the job of the product owner. Scrum-wise it is a good idea that the scrum master spreads scrum values and, for example, coaches the scrum team to become self-organizing. Whether this is aligned with the company goals remains to be seen.)

3. **Technical PO**: The scrum master is also supposed to act as a (technical) product owner. (There is a reason why scrum knows three roles and not just two. Avoid assuming more than one role at a time in a scrum team.)

4. **Outcome messenger**: The scrum master reports to stakeholders the output of the scrum team (velocity, burndown charts). (Velocity—my favorite agile vanity metric.) (**Read More**: Agile Metrics — The Good, the Bad, and the Ugly.)

5. **SuperSM**: The scrum master is supposed to handle more than one or two teams simultaneously. (Handling two scrum teams is already challenging, any number beyond that is not feasible.)

6. **Scrum secretary**: The scrum master is supposed to do secretarial work (room bookings, facilitation of ceremonies, ordering office supplies). (**Read More**: Scrum Master Anti-Patterns: Beware of Becoming a Scrum Mom (or Scrum Pop).)

7. **Scrum mom**: The scrum master is removing impediments on behalf of the team. (How is the scrum team supposed to become self-organizing if the scrum master handles all obstacles?).

8. **Team manager**: The scrum master is responsible for team management. (If nothing else helps read the manual Scrum Guide: Is there anything said about team management by the scrum master?)

9. **Delivery manager**: The scrum master is responsible for the "overall delivery of the committed sprint". (I assume the organization does not understand scrum principles very well. The forecast and the sprint goal seem to be particularly challenging.)

10. **CSM®, CSP® & CST®**: CSM or equivalent certification is listed as mandatory. (A typical save-my-butt approach to hiring. A CSM certification only signals that someone participated in a workshop and passed a multi-choice test.)

11. **Delivery scapegoat**: The scrum master is expected to accept full responsibility of the delivery process. (That is rather the responsibility of the scrum team.)

12. **Proxy PO**: The scrum master is expected to drive functional enhancements and continuous maintenance. (Maybe someone should talk to the product owner first?)

13. **Keeper of the archives**: The scrum master is expected to maintain relevant documentation. (Nope, documentation is a team effort.)

14. **The PM Reloaded**: The scrum master organizes the scrum team's work instead of the project manager. (Why use scrum in the first place if creating self-organizing teams is not the goal?)

15. **Risk detector**: The scrum master is expected to monitor progress, risks, resources, and countermeasures in projects. (The scrum master is neither a project manager nor a risk mitigator. (Risk mitigation is a side-effect of becoming a learning organization built around self-organizing teams.))

16. **Scrum minion**: The scrum master is expected to prepare steering team and core team meetings. (The last time I checked the Scrum Guide there was no 'steering team' men-

tioned.)

17. **WTF?** The scrum master is expected to perform the role for "multiple flavors of agile methodologies". (Multiple what?)

18. **Psychic**: The scrum master is expected to participate in "project plan review and provide input to ensure accuracy". (The scrum master is neither a project manager nor capable of predicting the future any better than another human being.) <

19. **Bean counter**: The scrum master is expected to "review and validate estimates for complex projects to ensure correct sizing of work". (Well, reviewing estimates might be the job of the scrum team during the product backlog refinement process if they see value in that. However, there is no review by the scrum master.)

20. **Discoverer**: The scrum master is expected to provide "design thinking sessions". (I love covering the product discovery process, too. However, this should be a joint effort with the product owner sand the rest of the team.)

21. **Techie**: The scrum master is expected to "walk the product owner through more technical user stories". (Nope, that is the job of the developers. The product backlog refinement meetings are ideal for this purpose.)

22. **Siloed in doing agile**: There is no mention of the scrum master either coaching the organization, or coaching the product owner.

My favorite anti-pattern is:

"…working reliably on projects within a given time and budget frame whilst maintaining our quality standards."

In other words: "Actually, we're happy with our waterfall approach but the C-level wants us to be agile."

Let's close this section with an exemplary job advertisement, posted by Zalando in 2016 for a (senior) agile coach position: (Senior) Agile Coach.

Conclusion

The job ad of the organization of your interest is a best-of of scrum master anti-patterns. Should you in this case immediately drop your interest in becoming a member of that organization? I don't think so. An extensive list of red flags can be beneficial, too.
For example, the HR department might merely be misaligned with the scrum team in question as the organization is still in the early day of its agile transition. That sounds like an attractive opportunity to me.

On the other hand, the organization might just try to attract talented people by sugar-coating its otherwise command & control like management style with some glitzy agile wording. Continuing the application process under these conditions might indeed be a waste of your time. A short phone call/interview will bring clarity.

Scrum Stakeholder Anti-Patterns

Introduction

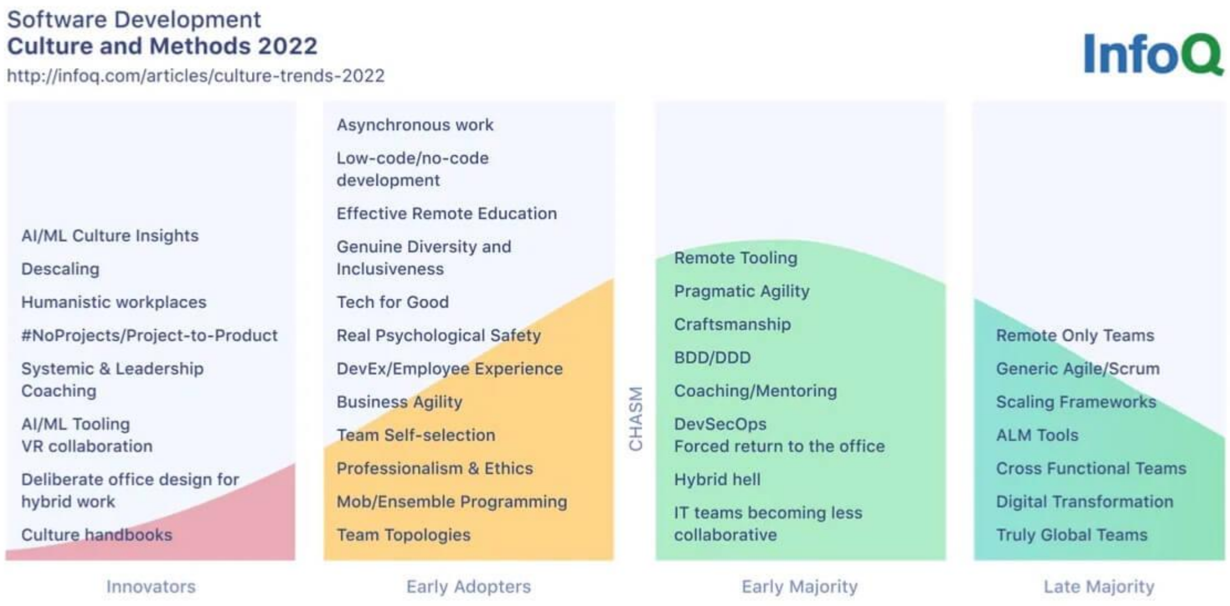
Learn how individual incentives and outdated organizational structures — fostering personal agendas and local optimization efforts — manifest themselves in Scrum stakeholder anti-patterns that easily impede any agile transformation to a product-led organization.

Stakeholder Anti-Patterns
We know what our customers need — follow us!
BOSS
BOSS
BOSS
© Stefan Wolpers, 2022 · Berlin Product People GmbH

The Stakeholder and Organizational Excellence in Legacy Organizations

Regularly, InfoQ applies the 'Crossing the Chasm' metaphor to engineering practices, thus covering a part of the agile movement to create learning organizations. Its 'Culture & Methods Trends Report March 2022' edition found that recent converts to Scrum, for example, will recruit themselves mostly from the late majority and laggards. (The early majority of organizations already embrace business agility, team self-selection, and mob/ensemble programming.)

Those laggards — or legacy organizations — are easy to spot: Some form of applied Taylorism, usually a strict hierarchy to command & control functional silos with limited autonomy, made it into the postindustrial era. Often, these organizations were once created to train farm boys into assembly-line workers within a standardized industrial process churning out standardized products in the name of output optimization. Human beings became cogs in the machinery, rewarded for functioning well without asking questions. Too bad, when nowadays, diversity, autonomy, mastery, and purpose become the driving factors in a highly competitive environment where more of the same for everyone is no longer creating value.



Copyright notice: InfoQ, 2022. All right reserved.

The conflict at the stakeholder level in such legacy organizations is apparent: mostly, the stakeholder is a manager of a functional silo with objectives that do not necessarily align with those of a product or Scrum Team. Where the organization needs to morph into a kind of ‘team of teams’ structure with a shared understanding of purpose and direction as well as the need to create value for the customers at heart, the reality of a legacy organization attempting to become agile is often very different. For managers, it means moving:

- From WIIFM (what-is-in-for-me syndrome) to team-playing — the team wins, the team loses,
- From career planning as an individual to servant leadership in a team of teams structure,
- From knowing it all and being the go-to person to solve problems to trusting those closest to the problem to come up with a solution,
- From ‘failure is no option’ to accepting failure as a means to learn effectively,
- From claiming success as a personal contribution to stepping back and letting the responsible team shine.

Abandoning yesterday's game – and probably its symbols of power, too — and accepting that an agile transformation may provide job security, but most certainly not role security is a monumental undertaking for the majority of the management of a legacy organization. Many of these managers will probably not adapt and even quit the organization sooner or later.

Common Scrum Stakeholder Anti-Patterns

After defining the context, let us consider some Scrum stakeholder anti-patterns in detail. Often, Scrum stakeholder anti-patterns result from a training and coaching void accompanied by preserving individual career objectives. Common in these cases, the organization may limit 'becoming agile' to product and engineering teams while the rest of the organization continues practicing the principles of the industrial age. In this context, stakeholder anti-patterns from a Scrum team perspective manifest themselves in the continued pursuit of local optima and personal agendas. Additionally, the incentive structure of the organization likely still fosters predictable stakeholder behavior that contradicts the organization's goals at a system level.

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

The following list of Scrum stakeholder anti-patterns addresses system-related issues, issues of individual players, and anti-patterns specific to Scrum events.

Scrum Stakeholder Anti-Patterns at System Level

These anti-patterns result mainly from a half-hearted approach to becoming an agile organization. Typically, this attitude ends in a form of cargo-cult agile:

- **Lack of transparency**: The organization is not transparent about vision and strategy hence the Scrum Teams are hindered to become self-managing. ("If you don't know where you are going, any road will get you there." In a complex environment with high levels of competition and uncertainty, everyone needs to understand the Why, the What, the How, and the Who. A lack of transparency will stop any effort to become agile dead in the tracks as missionaries won't flock to the cause but mercenaries. Elon Musk's "The Secret Tesla Motors Master Plan (just between you and me)" is a good example of leadership transparency at the vision and strategic level.)

- **Lack of leadership support**: Senior management is not participating in agile processes, for example, the Sprint Reviews, despite being a role model. Instead, they expect a different form of (push) reporting. (The support of the upper management is mission-critical for any transformation. No Scrum team will be successful if the 'leadership' is not leading the effort; for example, attending Sprint Reviews to signal everyone in the organization: Becoming agile is here to stay; it is not a fad.)

- **Cargo-cult agile by cherry picking**: Agile processes are either bent or ignored whenever it seems appropriate, for example, the Product Owner role is reduced to a project manager role. Or stakeholders are bypassing the Product Owner to get things done and get away with it in the eyes of the senior management, as they would show initiative. There is a lack of discipline to support the agile transformation. (Cherry-picking agile practices while ignoring the context required to make them work is a common form of window dressing 'agility.' However, being a part of a complex system, isolated agile elements typically rarely work if the organization aims to achieve ‘business agility.’ Becoming agile requires a mindset change followed by organizational change. Simply having ‘retrospectives’ once in a while won’t cut it.)

- **Agile on a tight budget**: The organization does not spend enough time and budget on proper communication, training, and coaching to create a shared understanding of purpose and direction among all organization members. (Becoming agile and embracing business agility is a monumental and expensive undertaking. Moreover, while expenditures become immediately apparent, the organization may not reap the return on investment for years to come. Therefore, cutting back on necessary expenses early in the transformation, pointing at a ‘lack of results’ is a sure-fire path to failure.)

- **Telling people how to do things**: In the good old times on the shop floor, it was a valuable trait to train newcomers or workgroups 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. (Let the people closest to the job at hand figure out how to do this; in a complex environment, there are no experts: “How this is done is at the sole discretion of the Developers. No one else tells them how to turn Product Backlog items into Increments of value.” (Source: Scrum Guide 2020.) Guiding by objectives and providing support when requested or needed will be appreciated, though.)

- **Steering committee meetings**: Unimpressed by the agile ways of working, the stakeholder insist on continuing the bi-weekly steering meetings to ensure that the Scrum team will deliver all their requirements in time. (This one has a quick remedy: Just do not participate in meetings that have no value for the Scrum team while reaching out to the stakeholder and supporting their learning path.)

- **Limited to non-existing feedback loops**: The sales organization and other functional silos guard the direct access to customers, thus preventing the Scrum teams from learning directly from customers. (The sales organization probably deems a direct contact of Scrum Team members with customers too risky (for their sales plans) and prevents it from happening. The problem is that filtering feedback from customers mainly results in a limited understanding of the customer’s situation on the side of the Scrum team. Furthermore, due to a likely lack of technical competence on the side of the sales organization, critical information regarding potential solutions may never reach the Scrum team, thus limiting its ability to tailor a solution to the customer’s needs. To become an effective product delivery organization, it is hence essential that the Scrum team directly communicate with customers regularly.)

Sprint Anti-Patterns of the IT Management

Also, there are some typical anti-patterns of those stakeholders closest to the Scrum Teams — the IT management:

- **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 and a bias for action to feel "in control" of the situation on their side. Most likely, canceling Sprints and gathering the Scrum Teams would also solve the issue.)

- **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 Scrum Team members can build trust among each other. The longevity of Scrum Teams has proven beneficial to build that trust. Moving people between teams, on the contrary, reflects a project-minded idea of management, rooted in utilization optimization of "human resources" of the industrial paradigm. It also ignores the preferred team-building practice that Scrum Teams should select themselves. All members need to be voluntarily on a team. Scrum does not work if team members are pressed into service. **Note**: It is not an anti-pattern, though, if Scrum Teams decide to exchange teammates temporarily. It is an established practice that specialists spread knowledge this way or mentor other colleagues. Also, dynamic re-teaming does not constitute an anti-pattern when the members of the involved Scrum Teams decide to do so.)

- **Special forces:** A manager assigns specific tasks directly to Developers, thus bypassing the Product Owner and ignoring the Developer's self-organization prerogative. Alternatively, the manager removes a Developer 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 of command and control practices. They continue to micromanage subordinates, although a Scrum Team could accomplish the task self-organized. This behavior demonstrates a level of ignorance that may require support for the Scrum Master from a higher management level to deal with.)

Personally Motivated Scrum Stakeholder Anti-Patterns

There are numerous ways in which stakeholders can impede the progress of a product team. Five of the most common ones are as follows:

- **'My budget' syndrome**: Stakeholders do not pitch for a Scrum team's attention 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 mainly in organizations that tie additional benefits to individuals. Instead, product development capacity needs to be allocated in the spirit of optimizing the whole organization. Note: 'pet projects' also fall in this category.)
- **'We know what to build'**: The is no user research, nor any other interactions of the product delivery organization with customers. (Several reasons are causing this phe-

nomenon ranging from a founder or entrepreneur who pursues their product vision without engaging in customer discovery activities. Or the product delivery organization is solely briefed indirectly by key account managers. The sales department probably deems direct contact of the Scrum Team members with customers too risky and prevents it from happening. These patterns share 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 challenging to come by as the culprits typically reject the idea that selfish motives guide them. To become an effective product delivery organization, it is essential that the team directly communicate with customers regularly.)

- **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 forwarding those to the product delivery organization as requirements. (The customers' problem is that they usually lack the depth of knowledge required to provide valuable answers to this question. They also lack the level of abstract thinking necessary to develop a valuable, 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 why product folks 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.)

- **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, hoping 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 is demanded in a more pressing fashion, typically four to six 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 much value while making it harder for the product organization to identify the signal in the noise.)

📺 For more information, please watch the webinar: [Product Discovery Anti-patterns](#).

Scrum Stakeholder Anti-Patterns at Scrum Event Level

The Sprint

Anti-patterns of this sort point at stakeholders' ignorance of the core idea of Scrum — self-managing teams:

- **(Regular) emergency work:** Someone in your organization has sold a non-existing feature or functionality to a customer to close a deal, probably already including delivery dates and contractual penalties in the case of non-delivery. Now, they want the Scrum Team to focus on delivering this item. (There might be moments where this outside intervention in the Scrum process may be unfortunate but acceptable. The more concerning issue here is the prospect of this behavior becoming a regularity. If the leadership does not acknowledge the exceptionality of the situation, it may derail using Scrum in the organization.)

- **Pitching Developers:** The stakeholders try to sneak in small tasks by pitching them directly to Developers. (Nice try #1. However, all suggestions for the future allocation of the Scrum Team's time compete with each other, and the Product Owner is the referee in this process.)

- **Everything's a bug:** The stakeholders try to speed up delivery by relabeling their tasks are 'serious bugs.' (Nice try #2. A particular case is an "express lane" for bug fixes and other urgent issues which some Scrum teams establish. In my experience, every stakeholder will try and make their tasks eligible for that express lane.)

- **Flow disruption:** The Scrum Master allows stakeholders to disrupt the flow of the Scrum Team during the Sprint. There are several possibilities for how stakeholders can interrupt the team's flow during a Sprint, for example:

    - Stakeholders constantly address Developers during the Sprint, ignoring that regular interruptions harm the Scrum team's effectiveness; here: accomplishing the Sprint Goal. (**Note**: I do not advocate that Scrum Masters shall restrict stakeholders' access to team members in general. However, they need to educate stakeholders on how to best communicate with the Scrum team.)

    - Line managers take team members off the Scrum Team, assigning them to other tasks. (Creating a Scrum team is an expensive exercise due to the inevitable drop in productivity during the norming and storming phases. Hence, changing their composition is a critical decision that shall include the team members. Scrum teams are no 'talent pools' in disguise at the disposal of line managers.)

    - Line managers add members to the Scrum Team without prior consultation of the team members. (Preferably, the Scrum Team members should decide who joins the team, see above.)

- Stakeholders or managers turn the Daily Scrum into a reporting session. (This behavior will impede the Developer's productivity by undermining their self-management while reintroducing command & control through the backdoor.)

Product Backlog and Refinement Anti-Patterns

These stakeholder anti-patterns result from ignoring the role of the Product Owner, turning them into a scribe. Two important anti-patterns of this kind are:

- **Requirements handed down:** The Product Owner creates Product Backlog items 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: Refining Product Backlog items is a team exercise. Moreover, using practices like user stories templates helps everyone understand the Why, the What, and the How. Remember Karl Popper: "Always remember that it is impossible to speak in such a way that you cannot be misunderstood.")

- **Prioritization by proxy:** A single stakeholder or a committee of stakeholders prioritizes the Product Backlog. (The strength of Scrum is building on the solid position of the Product Owner. The Product Owner is the only person to decide what tasks become Product Backlog items. Hence, the Product Owner also decides on ordering the Product Backlog. Take away that empowerment, and Scrum turns into a pretty powerful waterfall 2.0 process.)

The Daily Scrum

Most anti-patterns in this category result from perceived information needs — think of them as withdrawal symptoms:

- **Status report:** The Daily Scrum is a status report meeting, and the Developers are waiting in line to "report" progress to a stakeholder. (The "three Daily Scrum questions" often serve as a template for this anti-pattern.)

- **Direct assignment of tasks:** A stakeholder assigns tasks directly to a Developer. (It is the prerogative of the Developers to pick all work they deem necessary to accomplish the Sprint Goal during the Sprint Planning. No one is authorized to increase the Developers' workload by assigning new tasks to them.)

- **Command & control by the management:** Line managers are attending the Daily Scrum to gather "performance data" on individual team members. (This behavior is defying the very purpose of self-managing teams.)

- **"A word, please":** Line managers are waiting until the Daily Scrum is over and then reach out to individual Developers for specific reporting from them. (Nice try. However, this hack is also unwanted behavior and distracts the Developers.)

- **Talkative chickens:** “Chickens” actively participate in the Daily Scrum. (Stakeholders are supposed to listen in but not distract the Developers members during their inspection of the progress towards the Sprint Goal.)

- **Communicating via body language:** Formally, stakeholders remain silent. However, they participate in the Daily Scrum via their body language. (Again, stakeholders are supposed to listen in but not distract the Developers. Yet rolling eyes and face-palming are as powerful as the spoken word. Moreover, even subtle forms of body language represent communication, as one cannot “not communicate.” Furthermore, some stakeholders may have a naturally intimidating presence that can prove harmful to the communication among the Developers.)

Sprint Planning Anti-patterns of Stakeholders

Forecast imposed: The Sprint forecast 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 Developers by defining their scope of the forecast. Or a stakeholder points at the team’s previous velocity demanding to take on more user stories. (“We need to fill the free capacity.”) Or the ‘tech lead’ of the Developers makes a forecast on behalf of everyone else.)

The Sprint Review

Again, this category is often a combination of ignorance, fighting a perceived loss of control or pulling rank to override scrum principles:

- **Scrum à la Stage-Gate®**: The Sprint Review is a kind of Stage-Gate® approval process where stakeholders sign off features. (This Sprint Review anti-pattern is typical for organizations that use an “agile”-waterfall hybrid: there are some happy agile islands, for example, our Scrum team, surrounded by a sea of waterfall, driven by functional silos, budgeting, and top-down goal-setting. Still, in such a world, the Scrum team is tasked with accomplishing a Product Goal. Therefore, it is the prerogative of the Scrum Team to decide what to ship and when.)

- **No stakeholders**: Stakeholders do not attend the Sprint Review. (There are several reasons why stakeholders do not participate in the Sprint Review: they do not see any value in the event, or it is conflicting with another important meeting. In addition, 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 event within the organization, at least at the beginning of using Scrum.)

- **No customers present**: External stakeholders—also known as customers—do not attend the Sprint Review. (Break out of your organization’s echo chamber and invite some customers and users to your Sprint Review. And do not let the sales folks object to this idea. Ignoring the direct feedback from customers at the Sprint Review inevitably leads to a lesser outcome.)

- **Starting over again**: There is no continuity in the attendance of stakeholders. (Longevity is not just beneficial at the Scrum team level but also applies to stakeholder attendance. If they change too often, for example, because of a rotation scheme, their ability to provide in-depth feedback might be limited. If this pattern appears, the Scrum 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 where team members introduce solutions to specific problems the Sprint addressed. Shift & Share is an excellent Liberating Structure microstructure for that purpose.)

The Sprint Retrospective

Here, it is mainly about control and line management issues:

- **Stakeholder alert:** Stakeholders participate in the Retrospective. (There are several opportunities in Scrum that address the communication and information needs of stakeholders: the Sprint Review, the Daily Scrum, probably even the Product Backlog refinement, not to mention opportunities of having a conversation at water coolers, over coffee, or during lunchtime. If that spectrum of possibilities is not sufficient, consider having additional meetings if your team deems them necessary. For example, there is the opportunity to have a meta-level Retrospective, explicitly including the Scrum team's stakeholders. However, the Retrospective as a Scrum team-internal event is off-limits to stakeholders.)

- **Line managers present**: Line managers regularly participate in Retrospectives. (This is among the worst Sprint Retrospective anti-patterns I can imagine. It turns the Retrospective into an unsafe place. And who would expect that an unsafe place would trigger an open discussion among the team members? Any line manager who insists on such a proceeding signals their lack of understanding of basic Scrum practices.

Sprint Retrospective Anti-Patterns
LINE MANAGER ALERT
BOYS, I WILL BE JOINING YOUR RETROS!
Looking forward to guiding you...
NO!
???
What the heck?
Boss
SM
Dev
Dev
Dev
Dev
PO
© Stefan Wolpers, 2022 · Berlin Product People GmbH

- **Let us see your minutes**: Someone from the organization — outside the Scrum team — requires access to the Retrospective minutes. (This is almost as bad as line managers who want to participate in a Retrospective. But, of course, this information is solely available to team members.)

Conclusion

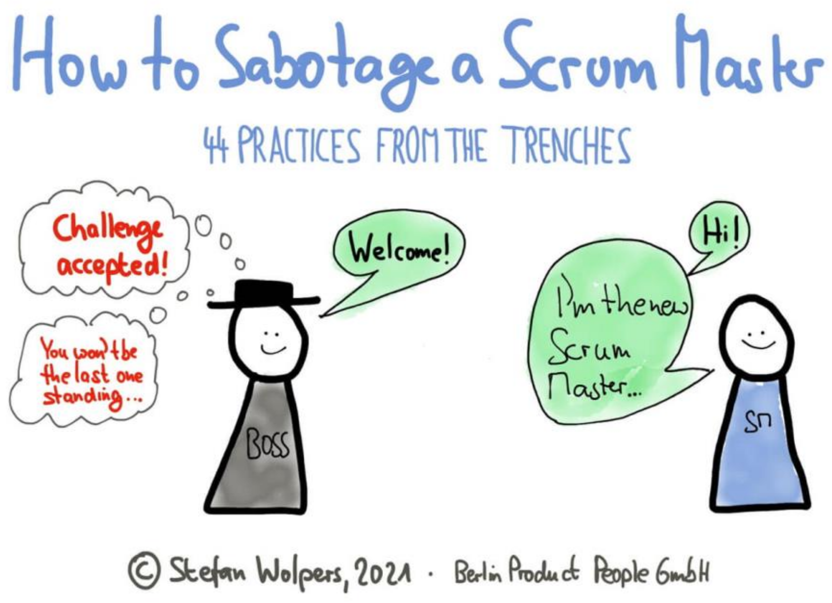
There are many different reasons why Scrum stakeholders do not act in line with agile principles. Some result from organizational debt, particularly in legacy organizations from the industrial area. Some are intrinsically motivated, for example, by personal agendas, while others originate from a lack of training or anxieties. Whatever the reason, Scrum stakeholder anti-patterns need to be overcome to turn an agile transformation into a success. Otherwise, you might end up in some form of cargo-cult agile or Scrumbut.

How to Sabotage A Scrum Master at an Organizational Level

44 Anti-Patterns from the Trenches to Avoid

One of my favorite exercises from my Professional Scrum Master classes is how to best sabotage a Scrum Master as a member of the middle management. The exercise rules are simple: You’re not allowed to use any form of illegal activity. So, outsourcing the task to a bunch of outlaws is out of the question. Instead, you are only allowed to use practices that are culturally acceptable within your organization.

Read on and learn more on how to best sabotage a Scrum Master from the exercise results of more than ten PSM I and PSM II classes. (I slightly edited the suggestions for better readability.)


How to Sabotage a Scrum Master
44 PRACTICES FROM THE TRENCHES
Challenge accepted!
You won't be the last one standing...
Welcome!
BOSS
Hi!
I'm the new Scrum Master...
SM
© Stefan Wolpers, 2021 · Berlin Product People GmbH

The Exercise: Consider How to Best Sabotage A Scrum Master

The exercise is based on a Liberating Structure microstructure called TRIZ.

Here is the briefing for the participants; typically, they have a five-minute timebox to come up with suggestions and observations in a workgroup of three to five people.

- You are a middle manager in the IT organization, and you believe this Scrum thingy is a fad and will go away—with a little help from your side.
- Your task: As a team, come up with ideas on how to best sabotage the new Scrum Master of the first Scrum Team in your organization.

Please note that only legal and culturally accepted practices qualify as a good outcome.

Once we accomplish the first part of the exercise—aggregating sabotaging practices—the workgroup then identifies those practices to sabotage Scrum Masters they have already witnessed, followed by identifying possible ways to counter these practices.

You can categorize the results from the before-mentioned classes into six groups:

Messing with the Scrum Framework in General

The first category of how to best sabotage a Scrum Master is generally about disqualifying Scrum itself as a helpful framework or introducing changes to conflict with the first principles of Scrum:

- Place the blame on Scrum whenever you can, even if it is technically unrelated.
- If Scrum uncovers an obstacle in the organization, blame that on Scrum.
- Find examples of where Scrum failed in other companies to spread around.
- Talk disrespectfully in the coffee breaks with developers and the other middle managers about Scrum and the role of the Scrum Master.
- Challenge anything the Scrum Masters try to say or do.
- Ignore the Scrum Master's offer to learn about Scrum.
- Create an ego-centric incentive system.
- Install multiple Product Owners in a Scrum Team.
- Place a proxy Product Owner in the Scrum Team and overrule all decisions.

Metrics & Reporting

The next bucket of useful sabotage practices are metrics, OKRs, KPIs — you name it. Just turn your Scrum Master into a glorified data-entry clerk with a challenging reporting burden:

- Ask the Scrum Master to prove their value with metrics.
- Create performance KPIs for each team member.
- Ask the Scrum Master to collect all working hours of the team members.
- Ask for individual performance metrics for every Sprint.
- Tie team member performance reviews with their average story points per Sprint using the Bell curve.
- Calculate a Scrum team budget and insist that the utilization rate of team members needs to be higher.
- Demand estimates and treat them as commitments.

Scrum Team Building & Management

If a challenging reporting burden does not help, sabotage your Scrum Master by actively undermining their activities to turn a group of people into a cross-functional Scrum Team:

- Only add members to the Scrum team that don't live any Scrum values.
- Recommend the most bull-headed senior developer as the engineering team lead.
- Constantly switch Developers from one project to another, claiming emergencies that require swift action.
- Have Scrum team members regularly work on multiple different Scrum teams.
- Add in new people to the Scrum team without prior consultation.
- Alternatively, slow down the hiring or replacement processes.
- Promote a member within the Scrum team to act as a proxy manager.

Work Organization

If you are already messing with the Scrum Team team-building process, why not place a few obstacles into the Scrum Team's way of working? Sabotage your Scrum Master by creating unachievable objectives while meddling with the very foundation of Scrum:

- Define unachievable objectives for the Scrum Team.
- Overload the Scrum team with requests, then complain to others that you do not get results on time.
- Hand over only fixed price, time, and scope projects to Scrum Team.
- Change requirements during the Sprint.
- Insist on hard deadlines.
- Ask the Scrum Master to provide a product roadmap with deadlines.
- Make the Scrum Master responsible for accomplishing deadlines.
- Outsource part of the product roadmap creation to an off-site team in a completely different timezone.
- Request work that would switch focus away from the Sprint Goal directly to Developers.
- Assign tasks directly to Scrum team members.
- Don't allow Scrum Team members to speak to the customer; act as the single point of contact.
- Create unnecessary organizational bottlenecks outside of Scrum, for example, approvals gates, etc.
- Only provide inadequate equipment and tools to the Scrum team.

Flow of Information

Does your Scrum Master have an insatiable appetite for data, information, and knowledge? Well, keep them out of the loop then. What could be an easier way of sabotaging Scrum:

- Claim that everyone already knows what to do. There is hence a need for alignment or a Scrum Master.
- Restrain from sharing essential or valuable information with the Scrum Team.
- Encourage silo thinking by promoting a strict "need to know" basis for sharing information and knowledge.

Scrum Events & Other Meetings

Finally, ensure that everyone on the Scrum Team understands that your events are more important than theirs:

- As a manager, claim that there are too many Scrum events that require too much time. Instead, suggest skipping some of them.
- Require to be present at every Scrum event.
- Exclude Scrum Master from important meetings outside the Scrum Team's events.
- Constantly pull Scrum team members into long unnecessary meetings during their Scrum team events.
- Be very understanding of the needs of the Scrum Team; for example, that stakeholders shall participate in the Sprint Review. However, never join any Scrum event yourself.

Conclusion

Every middle manager has ample opportunity to sabotage Scrum Masters. Moreover, most of the time, it will be unlikely that others will identify these practices as deliberate obstruction. Providing the benefit of the doubt and assuming positive intent, they probably get away with it. Therefore, reading these signals early in the process is vital to us agile practitioners.
What practices to sabotage Scrum Masters have you observed? Please share your learnings with us in the comments.

How to Sabotage A Product Owner

53 Anti-Patterns from the Trenches to Avoid

One of my favorite exercises from my Professional Scrum Product Owner classes is how to best sabotage a Product Owner as a member of the middle management. The exercise rules are simple: You're not allowed to use any form of illegal activity. So, outsourcing the task to a bunch of outlaws is out of the question. Instead, you are only allowed to use practices that are culturally acceptable within your organization.

Read on and learn more on how to best sabotage a Product Owner from the exercise results of more than twenty PSPO classes. (I edited the suggestions for better readability.)

How to Sabotage a Product Owner
53 PRACTICES FROM THE TRENCHES
You won't spend my budget w/o my approval!
I know what I need. Game on!
Welcome!
It's good to have you!
BOSS
I am the new Product Owner.
How can I help?
HI!
PO
© Stefan Wolpers, 2022 · Berlin Product People GmbH

The Exercise: Consider How to Best Sabotage A Product Owner

The exercise is based on a Liberating Structure microstructure called TRIZ.

Here is the briefing for the participants; typically, they have a five-minute timebox to come up with suggestions and observations in a workgroup of three to five people:

- You are a middle manager in the IT organization, and you believe this Scrum thingy is a fad and will go away—with a little help from your side.
- Your task: As a team, come up with ideas on how to best sabotage the new Product Owner of the first Scrum Team in your organization.

- Please note that only legal and culturally accepted practices qualify as a suitable outcome.

Once we accomplish the first part of the exercise—aggregating sabotaging practices—the workgroup then identifies those practices to sabotage Product Owners they have already witnessed, followed by identifying possible ways to counter these practices.

I categorized the results from the before-mentioned classes into eight groups:

1. Messing with the Scrum Framework in General

The first category of how to best sabotage a Product Owner is generally about disqualifying Scrum itself as a helpful framework or introducing changes to conflict with the first principles of Scrum:

- Ignore the autonomy of the Product Owner in general.
- Bypass the Product Owner and talk directly to the Developers.
- Product Owner has to act as Scrum Master at the same time.
- Impose traditional project management tasks on the Product Owner.
- Enforce traditional or waterfall practices within the Scrum framework; for example, insist on approving Increments.
- Isolate the Scrum team by applying “old” processes at the interface between the Scrum team and the rest of the organization.
- Organize important stakeholder meetings in parallel to Sprint Reviews.
- Ensure that critical stakeholders are never available for the Product Owner.
- Set hard timelines or milestones.
- Insist on all projects being accomplished within a fixed time, scope, and budget setting.

2. Scrum Team Building & Line Management

If the direct approach does not yield the expected results, why not use the backdoor and help make the Scrum team as dysfunctional as possible? Some suggestions worth considering are:

- Ignore the self-management of Scrum Teams; instead, micromanage them.
- Assign Developers to multiple projects to increase the overhead. Then, blame the Product Owner for lack of productivity.
- Change the Scrum Team structure often to keep it in a state of constant team building. For example, introduce new team members during a Sprint or shuffling Developers between multiple Scrum teams.
- Deliberately pick new Scrum team members from those who openly despise Scrum.
- Assign ad-hoc tasks outside the Scrum team’s focus to team members.
- Put constant pressure on the Developers. Tell them they are too slow and they don’t understand the business.

- Award contradicting individual incentives to Developers to cause internal competition within the Scrum team.
- Use the Daily Scrum to check on progress and hold the Product Owner accountable for not meeting your targets.
- Make the availability of tools and other resources difficult for the Scrum team.

3. Flow of Information

If you are already messing with the Scrum Team team-building process, why not place a few obstacles into the Scrum Team's way of working? Sabotage your Product Owner by keeping them in the dark while holding them accountable for not being up-to-date:

- Don't provide feedback.
- Otherwise, respond irregularly and with a long delay to inquiries of the Product Owner.
- Don't share new market insights or intelligence with the Product Owner, particularly during the Sprint Review. Later hold the Product Owner accountable for not knowing this information.
- Don't attend the Sprint Review in the first place.
- Generally, don't accept meeting requests by the Product Owner.
- Restrict the Scrum team's access to the customers.
- Ensure that the salespeople relay feedback from customers exclusively.
- Never communicate organizational changes that could affect the Scrum team's work in time.

4. Metrics & Reporting

The next bucket of useful sabotage practices are metrics, OKRs, KPIs — you name it. Just turn the members of your Scrum team into glorified data-entry clerks with a challenging reporting burden:

- Keep the team busy with unnecessary reports, documentation, or administrative tasks.
- Enforce proprietary and obscure metrics useless to the Scrum team.
- Demand evidence of progress. At the same time, reject all metrics from the Scrum team as unsuited.
- Request detailed accounts of what will be done when. Insist on defining milestones with accurate breakdowns of work items.

5. Vision, Product Discovery & Product Backlog Management

All sabotage is—at least partly—based on deception. Therefore, obfuscate whatever might help the Product Owner to get an understanding of what strategic direction they are supposed to support with their Scrum team:

- Constantly change the organization's focus or priorities.
- Create an unclear vision and keep changing it regularly.

- Break down the vision into unclear requirements and refuse to elaborate, claiming they are apparent.
- Overwhelm the Product Owner with a never-ending flow of unclear requirements and requests.
- Ask for requirements specifically outside of the expertise of the Scrum team, deny them the necessary resources, yet hold the Product Owner accountable for the subsequent failure.
- Overrule the Product Owner's decisions by pulling rank.
- Spread the Product Owner thin by assigning them to multiple products simultaneously only to complain about the resulting lack of attention to detail.
- Add items to the Product Backlog at your will.
- Micromanage the Product Backlog and its refinement; for example, insist on being present but rarely commit to attendance.
- Insist on detailed specifications for each Product Backlog item.

6. Messing w/ the Sprint, Increment, or Release

Does the Product Owner still lead the Scrum team towards creating Increments? Time for some rearguard action by introducing new approval stages and security measures—for the sake of your customers, of course:

- Reject Increments as not meeting specifications and blame the Product Owner.
- Add tasks for the Developers to the Sprint Backlog that are not aligned with the Sprint Goal.
- Keep striving for perfection, never release.
- Request a separate, multi-step approval process outside the Scrum team before any release.
- Install a separate governance board dedicated to product quality and security issues.
- Define new security protocols that overburden the Scrum team, claiming legal requirements.

7. Politics

Use your career equity, the network you have been building for years, to your advantage. Maybe, it is time to call in some favors:

- Continuously seed doubt within the organization's hierarchy regarding the capabilities of the Product Owner.
- Challenge Scrum's suitability as a framework in your organization's case by pointing at failures in other organizations.
- Play 'blame-game' with the Product Owner for any failure or shortcoming while claiming successes of the Scrum team as yours.

8. Budget

If everything fails, don't waste time reading the Scrum Guide. Instead, use the budgeting process to your advantage:

- Slash the budget but keep the scope.
- Keep the budget but increase the scope.
- In general, make the budget process longer and more complex by adding useless requirements, for example, regarding the paperwork.

Conclusion

Every middle manager has ample opportunity to sabotage Product Owners. Moreover, most of the time, it will be unlikely that others will identify these practices as deliberate obstruction. Providing the benefit of the doubt and assuming positive intent, they probably get away with it. Therefore, reading these signals early in the process is vital to us agile practitioners.

Please keep in mind: not everyone will be excited when the organization succeeds in becoming a learning organization. Self-management and shifting decision-making to those closest to the problems will likely flatten the hierarchy, reducing the need for managers. Typically, scaling "agility" equals "descaling the organization."

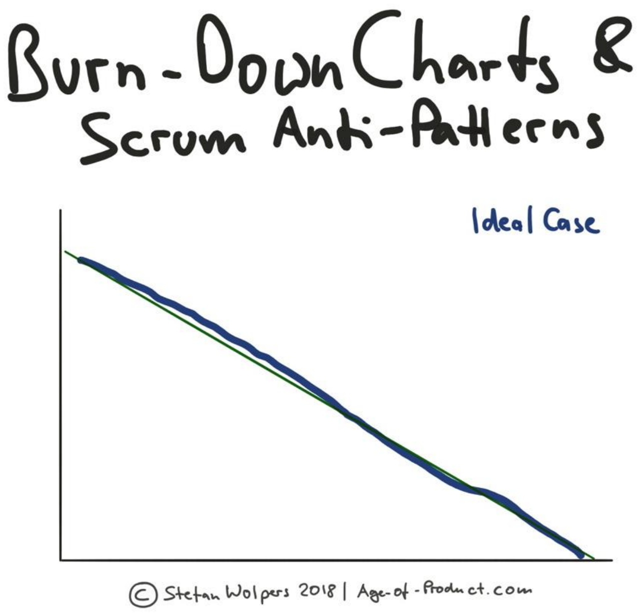
How to Detect Scrum Anti-Patterns

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

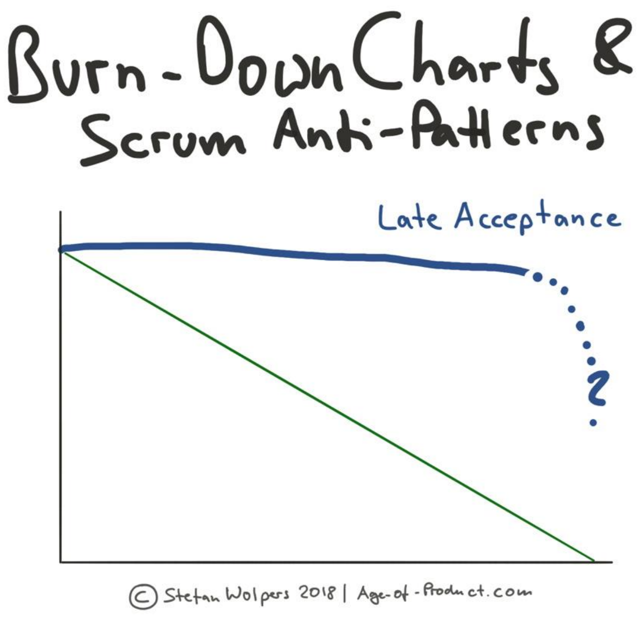
Burn-down charts have become popular to provide team members as well as stakeholders with an easy to understand status whether a sprint goal will be accomplished. (Critics of the burn-down chart may note, though, that a scrum team should have a gut feeling anyway whether the sprint goal is achievable.)

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:

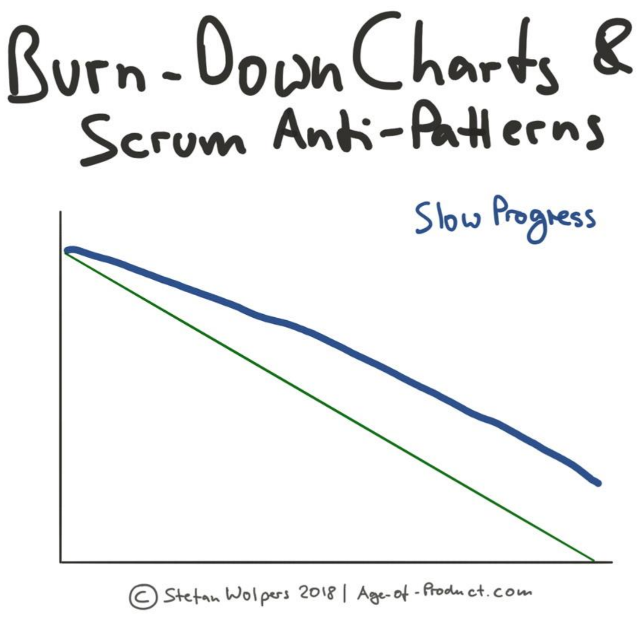
- **The absent product owner**: The product owner is rarely available for the team to clarify matters and accept work. This is creating an artificial queue that has a diminishing effect on the team's ability to deliver value by delaying the necessary clarification of tasks or the shipment of tasks themselves. (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.

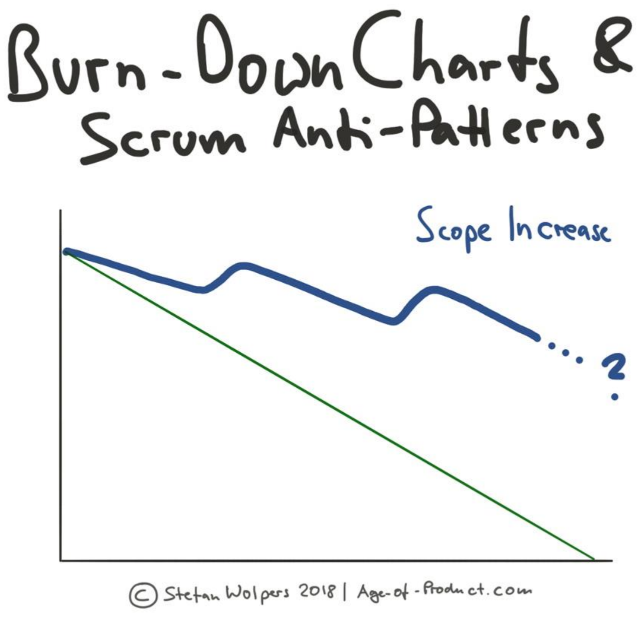
- **Capacity issues**: The capacity of the team changes after the sprint starts, for example, team members get sick, or they give notice and leave the team. (Note: Admittedly, this is hardly plannable anyway.)

- **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:



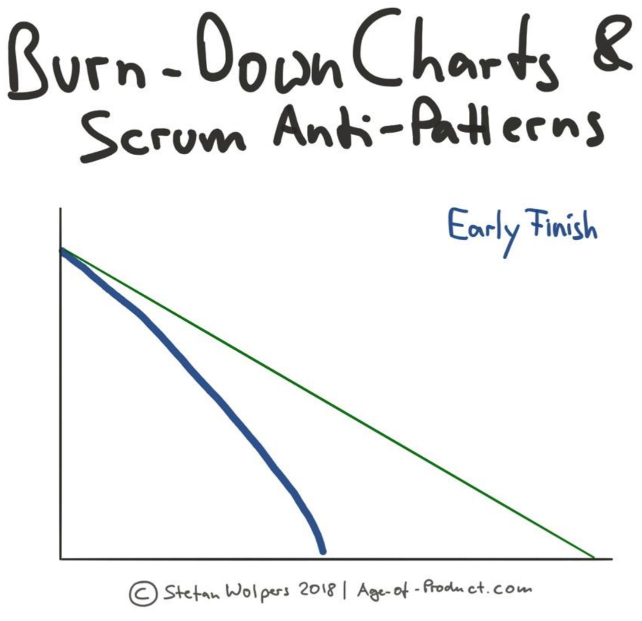
Most of the time, this pattern can be attributed to inadequate preparation:

- **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?

About the Author

Stefan is a Professional Scrum Trainer with Scrum.org, an Agile Coach, and Scrum Master.

PST Professional Scrum Trainer Scrum.org

He is specializing in coaching agile practices for change, for example, agile software development with Scrum, LeSS, Kanban, and Lean Startup, as well as product management.

He also serves as one of the XSCALE Alliance stewards and coaches organizations in business agility. Additionally, he is a licensed facilitator of the Agile Fluency™ Team Diagnostic.

He has served in senior leadership positions several times throughout his career. His agile coaching expertise focuses on scaling product delivery organizations of fast-growing, venture-capital funded startups, and transitioning existing product teams in established enterprise organizations.

Stefan is also curating the popular 'Food for Agile Thought' newsletter for the global Agile community with 35,000-plus subscribers. He blogs about his experiences on Age-of-Product.com and hosts the most significant global Slack community of agile practitioners with more than 12,000 members.

His ebooks on agile topics have been downloaded more than 85,000 times. Lastly, Stefan is the organizer of the Agile Camp Berlin, a Barcamp for 200-plus agile practitioners.

Read more about Stefan at Scrum.org, and connect with him via LinkedIn, or Twitter, or privately via email.