TEAMS THAT FINISH EARLY ACCELERATE FASTER

..the *Development Team* may work together well but struggles every *Sprint* to attain the *Sprint Goal*. In the worst case, the team is feeling demoralized and velocity is low (see Notes on Velocity).

✣ ✣ ✣

Teams often take too much work into a Sprint and cannot finish it. Failure to attain the Sprint Goal prevents the team from improving.

Development Teams can be optimistic about their ability to finish *Product Backlog Items*. But in doing so, they fail to give themselves time to reduce technical debt and sharpen their saws. Thus they are doomed to a persistently slow pace.

Market pressure and low velocity may make the *Product Owner* desperate, and the *Development Team* will accept larger and larger *Sprint Goals.* This compounds the problem and they decelerate.

Individuals on a team may be working on other things subversively. *Development Team* members hide such work on teams that persistently overreach their ability or overestimate their capacity.

If the team takes on too much work, it feels overburdened and under pressure. When team members cannot get the work Done (see *Definition of Done*) by the end of the *Sprint*, everyone is unhappy—the *Development Team*, the *ScrumMaster*, the *Product Owner*, and management. Furthermore, the team does not have time to think clearly about how to improve its work and will probably enter the next *Sprint* with work that is only partially Done.

A lack of basic understanding of lean practices is often a cause of management pressure on teams. Taiichi Ohno's taxonomy of waste has the category of "Absurdity"—stress due to excessive scope. This can cause massive slowdown.

Muri *(無理) is a Japanese word meaning "unreasonableness; impossible; beyond one's power; too difficult; by force; perforce; forcibly; compulsorily; excessiveness; immoderation," and is a key concept in the Toyota Production System (TPS) as one of the three types of waste (muda, mura, muri) (see articles for respective terms in Wikipedia).*

Therefore:

Take less work into a Sprint (than the previous Sprint) and aim for a less ambitious Sprint Goal.

Note that the level of ambition of the *Sprint Goal* (outcome) may not be proportional to the volume, cost, or duration of the work.

Yesterday's Weather will maximize your probability of success. Then you can implement *Illegitimus Non Interruptus* (aka *Interrupt Buffer*), which will systematically deal with any interruptions that would prevent you from finishing early. On early completion of the *Sprint Goal*, pull forward from the next

Sprint's backlog, which will increase *Yesterday's Weather* for future *Sprints*. To increase the probability of acceleration, apply *Scrumming the Scrum* to identify your kaizen (see *Kaizen and Kaikaku*) in the *Sprint Retrospective*. Put the kaizen in the *Sprint Backlog* with acceptance tests (see *Testable Improvements*) for the next *Sprint* as top priority.

✣ ✣ ✣

Openview Venture Partners noticed this pattern after analyzing data from dozens of sprints with multiple teams. The teams that finished their *Sprint* early accelerated faster. The common denominator was finishing early. It allowed teams to think more clearly about what they were doing—to remove impediments, to pull forward backlog from the next Sprint, to develop a winning attitude; and to increase *Yesterday's Weather*.

An earlier version of this pattern was published at HICCS 2014.

Adapted from: Scrum Patterns. Unknown Publish Date. Teams That Finish Early Accelerate Faster. [ONLINE] Available at: https://sites.google.com/a/scrumplop.org/published-patterns/retrospective-pattern-language/teams-that-finish-early-accelerate-faster [Accessed 7 December 2019].

ILLEGITIMUS NON INTERRUPTUS

..the *Scrum Team* is serving many stakeholders, all of whom are competing for attention from the team. Requests and demands come to the team from management, from Customer A through Customer Z, and from sales and marketing. In addition, work in progress may uncover surprise shortcomings in the product itself that require attention. The frequency and importance of these requests varies over time, and occasionally their volume and urgency are overwhelming.

✣ ✣ ✣

Changing priorities or problems in the field often interrupt the work of Scrum Teams during a Sprint. Sales and marketing demands, combined with management interference, can cause chronic dysfunction in a team, repeated failure of Sprints, failure to meet release dates, and even company failure.

In many ways, the *Scrum Team* is a community resource that meets the needs of many stakeholders. The tragedy of the commons is a dilemma arising from the situation in which multiple individuals, acting independently and rationally consulting their own self-interest, will ultimately deplete a shared limited resource even when it is clear that it is not in anyone's long-term interest for this to happen. American ecologist and philosopher Garret Hardin first described this dilemma in an influential article titled "The Tragedy of the Commons," which was first published in the journal Science in 1968. [1]

The *Scrum Team* is a critical resource for creating new software and maintaining old software. This makes it a central resource for solving problems that arise during both development and product use, for technical communications with customers, for marketing demos, and for special projects to serve the needs of everyone in the organization. See *Work Flows Inward*.

Often poor product ownership allows competing priorities in a company to reach a *Scrum Team*. Some teams have even been bribed to work on features not in the *Product Backlog*.

In almost all cases, it is desirable to have the *Scrum Team* "eat their own dog food." If they produce a defect that gets into the field, they need to fix it as soon as possible. Setting up special maintenance teams to fix defects incentivizes the *Scrum Team* to not be attentive to latent defects.

For these, and many other reasons, a *Scrum Team* is always exposed to interrupts that disrupt production.

Therefore:

Explicitly allot time for interrupts and do not allow more work than fits within the allotment. If work exceeds the allotment, abort the Sprint.

Set up three simple rules that will cause the organization to self-organize to avoid disrupting production.

This strategy will help the team replan during the *Sprint* to raise the chances of delivering the complete *Product Increment.*

1. The team creates a buffer for unexpected items based on historical data. For example, let's say that a third of the team's work on average comes from unplanned work coming into the *Sprint* unexpectedly. If the team *velocity* averages 60 points, the team reserves 20 points for the interrupt buffer.
2. All non-trivial requests must go through the *Product Owner* for triage. (Web page spelling errors and compilation errors are examples of trivial errors where the fix is so obvious that there is no benefit from additional business insight. Developers may spend some small, time-boxed amount of time addressing even non-trivial defects before escalating to the *Product Owner*.) The *Product Owner* will give some items low priority if there is no perceived value relative to the business plan. The *Product Owner* will push many other items to subsequent *Sprints* even if they have immediate value. A few items are critical and the team must complete them in the current *Sprint*, so the *Product Owner* puts them into the interrupt buffer.
3. If the buffer starts to overflow, that is, the *Product Owner* puts one point more than 20 points into the *Sprint*, the *Scrum Team* must automatically abort, the *Sprint* must be replanned, and the *Product Owner* notifies management that dates will slip. It is essential to get management agreement on these rules and to enforce them. The *Product Owner* must always be available to the team and other stakeholders. In the *Product Owner's* absence, the *Scrum Team* should designate one of its own to temporarily fill that role.

The *Product Owner* balances the buffer size to balance short-term customer satisfaction with future revenue generation. Often, a *Product Owner* has third-party metrics on customer satisfaction that he or she can adjust up or down with buffer size.

This strategy is independent of the focus on fixing all defects that arise in the *Sprint* from backlog items worked on during the *Sprint* (see Goo*d Housekeeping*). It is also independent of *PBIs* assigned to a *Sprint* by the *Product Owner* as part of *Sprint Planning* to reduce technical debt. Low defect tolerance increases velocity in general, but exceeding the buffer typically generates at least a 50 percent reduction in velocity. The *Product Owner* must use common sense to balance these forces. See *Whack the Mole*.

✣ ✣ ✣

These rules will invariably cause individuals to self-organize to avoid blowing up a *Sprint*, as no individual wants to be seen as the direct cause of *Sprint* failure.

Even better, the buffer will tend to never be full, allowing the team to finish early and pull forward from the backlog and/or work on removing impediments. This is important because *Teams That Finish Early Accelerate Faster*. Furthermore, if the team uses *Yesterday's Weather* to size the buffer and the buffer almost never fills up, the buffer size continuously gets smaller, making the interrupt problem go away.

Counterintuitively, this does not cause critical problems to be hidden or unresolved. The *Product Owner* will put any critical items on the *Product Backlog*. This helps the team increase its velocity and increase the output of future *Sprints*. This typically allows more than enough time to address critical items and often with spare capacity.

A team exhibits a high degree of *Product Pride* to pause in its work for the good of the product quality and reputation. Other related patterns include: *Product Owner*, *Product Backlog*, *Teams That Finish Early Accelerate Faster*, *Work Flows Inward*, and *Completion Headroom*.

Adapted from: Scrum Patterns. Unknown Publish Date. Illegitimus Non Interruptus. [ONLINE] Available at: https://sites.google.com/a/scrumplop.org/published-patterns/product-organization-pattern-language/illegitimus-non-interruptus [Accessed 7 December 2019].

QUANTUM ENTANGLEMENT

(Spooky action at a distance)

Entanglement is a strange feature of quantum physics, the science of the very small. It's possible to link together two quantum particles -- photons of light or atoms for example – in a special way that makes them effectively two parts of the same entity. You can then separate them as far as you like and a change in one is instantly reflected in the other. (The Strange World of Quantum Entaglement. Paul Comstock, California Literary Review, 30 Mar 2010)

… you have teams working in distributed locations, often across timezones, countries and cultures

✣ ✣ ✣

Distributed Teams are unable to work at the same level of productivity as a collocated team. They produce lower quality work and have reduced morale. Even a team that starts off well will see degradation over time.

When teams are separated they lose their collective identity and conflicts emerge. This may be displayed in areas such as communication breakdowns, increase in defects, velocity decreases and overall product incoherence. "The number one factor in a team's identity is local allegiance (which is a function of geography)." Peter Burgi

Timezone differences exacerbate the problem: meetings become difficult to schedule. In addition, time zone differences can mean that some people meet in the morning when they are fresh, but those meeting in their late afternoons are running out of energy.

There is a tendency amongst distributed teams to exhibit the "Us and them" syndrome; it is easier to blame the remote people when anything goes wrong.

Different locations have different cultures with respect to management; e.g. teams fear being transparent or being the bearer of bad news and will hide information to the detriment of the whole team.

The greatest bonds are formed when people meet personally; these bonds form automatically through face-to-face meetings. However, these bonds break down over time.

When people have a personal bond, you can give constructive feedback without fear that it could be taken in a negative way, and they will understand your good intent. This also breaks down over time.

The "tribe" way of thinking is very powerful. We want to establish a single "tribe" among all locations, but the local influences undermine the cross-location tribe culture.

Therefore:

Establish social, cultural and technical connections between the locations of the team at the time the team is formed, establish measures of normal accomplishment, and monitor these measurements. If degradation is detected, re-establish the connections.

✣ ✣ ✣

There is a three step process – initialize (establish a bond), inspect (monitor for continued unity), and adapt (refresh the bond)

Initialization requires getting the team to work together locally (Face to Face Before Working Remotely). Beyond that the team needs to work together so we can establish a new culture and provide baseline measurement of productivity. We believe there is a minimum time or minimum number of sprints to accomplish this (approximately one month or several sprints).

We know this can work well with two locations. More locations make this difficult and no published data demonstrate sustainable performance yet at more than two locations.

The team needs to bond and social as well as work activities are required. Establish good remote pair relationships that will continue once the team is distributed.

A critical part of initialization is setting up the technical environment to be location-transparent.

Inspecting involves collection of key metrics on team performance and watching for any degradation. Important measures include: velocity, open bugs, broken builds, commits, and team happiness. Don't forget value.

We see symptoms of team degradation among the four quadrants of measurement (velocity, quality, sustainable pace, and team morale). However, the most significant one related to distribution degradation are related to velocity. In particular, look at commits: if they drop, it could be that the remote teams are no longer trusting each other.

Adapting is repairing the team to reestablish shared culture with local performance. This is difficult and often not accomplished without another face to face experience.

For daily meetings video works best. Phone calls don't transmit as much information. We pick up a few nonverbal cues through vocal intonation, etc., but it is so much stronger with faces. Regular visual meetings are crucial, but are still inferior to physical presence.

Photos of everyone in prominent places are quite powerful e.g. on a conference phone hub or beside the Scrum board.

Adapted from: Scrum Patterns. Unknown Publish Date. Quantum Entanglement [ONLINE] Available at: https://sites.google.com/a/scrumplop.org/published-patterns/distributed-scrum-pattern-language/quantum-entanglement [Accessed 7 December 2019].

STABLE TEAMS

…teams have been producing product for some time. An ever-changing business landscape raises questions about staff adjustment, growth, and optimal team composition.

✣ ✣ ✣

Stakeholders are happiest with teams who can meet their expectations in a timely fashion, so the team wants to do what is necessary to reduce variance in its predictions.

In project management there is a tendency to confuse human beings with human resources. It leads to "resource management," lining up demand with each team's capacity (or, sometimes, each team member's capacity) to contribute to the deliverable.

It often results in moving people from team to team when starting projects, or crisis to crisis during delivery, and leads to an unstable environment with added costs of:

- administration to keep track of what people are working on,
- reduced efficiency, as teams need to integrate a new member (effectively creating a new team) and the new member needs to learn about the team and its product,
- exposure to Brooks' Law ("adding manpower to a late software project makes it later" ([1], p. 25)).

When teams are formed from a resource pool, the resource utilization often leads to multitasking with people distributed across several teams and often on several products. Add in changing requirements and the instability becomes unbearable. So we need to fix something—the stable and unstable parts must be balanced. For years the response has been to freeze the requirements so our ever-changing resources are able to deliver at a (hopefully) predictable time. Locking down the requirements of our products prevents us from learning, and ignores product changes that could create the *Greatest Value*, so this is not a good solution.

Therefore:

Keep teams stable and avoid shuffling people around between teams. Stable Teams tend to get to know their capacity, which makes it possible for the business to have some predictability. Dedicate team members to a single team whenever possible.

✣ ✣ ✣

Members of *Stable Teams* get to know each other. The team members experience each other's work style and learn how much work they can do together. A Stable Team grows in familiarity and consistency of meeting mutual expectations and starts developing a community of trust (see *Community of Trust*).

Research has shown that fatigued NASA crews that had worked extensively together made about half as many mistakes as teams of rested pilots with no prior joint work experience. This is because many of the factors that relate to effective teamwork (such as shared mental models, cohesion, climate, and psychological safety) come about only in teams that have consistently worked together. People that work together for only a short period of time will probably not invest much energy in improving their work processes or social interactions with each other—in a couple of months (or less) they will be working with other people. On the other hand, if people understand that they will stay with the same colleagues for longer periods of time, they are more likely to invest energy in creating an enjoyable team environment and improving their work processes.

A team can use its *velocity* to measure improvement in its capacity for work. Many *Scrum Teams* use velocity as a key to predictability. Velocity is currently the most-proven praxis to increase the level of predictability. But what is often forgotten, when measuring a team's velocity, is that the only way a team can get to know its velocity is by having the same team members over a longer period of time.

Cross-Functional Teams will benefit the most from long-term stability because there are more opportunities to share knowledge and experience. Specialized teams (e.g., that provide services for specialized skill sets or technologies) will also yield some benefits but have broader negative consequences for the organization—for example, creating queues and requiring extra administrative overhead for work.

Stable Teams will create pressure on the work pipeline. The workflow now must structure work to fit with the teams, rather than restructuring teams to fit with work—or to handle crises. This will, in turn, highlight capability issues that the organization must address. For example, constantly reshuffling a specialist between teams will make it visible that some skill set is missing across several teams. As you transition to Stable Teams, temporarily assign specialists where they are needed with the stipulation that they cross-train team members in their area of expertise, to reduce risk in the long term.

Start using *Stable Teams* by committing to keeping teams together, potentially using self-selecting teams (see *Self-Selecting Teams*). Then follow up by managing the work to fit this structure, and make an effort to accommodate gaps in capability in the teams. Let the teams self-organize to spread competencies and to explore the best team compositions; you will find that a structure converges quickly.

The organization replaces the "flexibility" of shifting people from crisis to crisis with flexible work assignment. Such flexibility lets teams take on work in accordance within their current capability and capacity, which in turn leads to more accurate forecasting. Moving people between teams to be "flexible" instead leads to higher cost and uncertainty.

This pattern helps the team grow and share its expertise over time to reduce the risk of losing a team member; see *Moderate Truck Number*.

A stable team that uses *Estimation Points* and lets the Pig*s Estimate* can get a realistic number for their velocity after a few *Sprints*; see *Running Average Velocity*.

Having Stable Teams allows an individual to master the work of the team. When there are multiple *Development Teams* or multiple *Scrum Teams*, using *Birds of a Feather* can support team members in furthering their mastery.

A *Stable Team* builds a collective identity that can be a foundation of a shared sense of pride, both in the product and in belonging to the team (see *Product Pride* and *Team Pride*).

Adapted from: Scrum Patterns. Unknown Publish Date. Stable Teams [ONLINE] Available at: https://sites.google.com/a/scrumplop.org/published-patterns/product-organization-pattern-language/development-team/stable-teams [Accessed 7 December 2019].

EMERGENCY PROCEDURE

Alias: Stop the Line

...companies, teams, and individuals often find their efforts are failing to deliver on time and the *Sprint Burndown Chart* shows failure is virtually certain. Rapid identification of problems and quick response is fundamental to the spirit of agility.

✣ ✣ ✣

Problems arise in the middle of a Sprint due to emergent requirements or unanticipated changes. By mid-Sprint, it may be obvious that the Development Team cannot complete the Sprint Backlog successfully. The team is high on the Sprint Burndown Chart and sees that it cannot achieve the Sprint Goal at the current rate of getting things done.

Causes of *Sprint* dysfunction are legion and this pattern focuses primarily on the top three of these common problems:

- Emergent requirements
- Technical problems
- Loss of critical people or capabilities
- Overestimated capacity (use *Yesterday's Weather*)
- Unplanned interruptions (use *Illegitimus Non Interruptus*)
- Previous work not Done (use *Definition of Done*)
- Product Owner changes backlog (use *Product Owner*)
- Management interference (use *Involve the Managers* and *MetaScrum*)

Agility requires rapid response to change, and that means making problems visible as early as possible. Unfortunately, new teams and average teams often do not want problems to become visible. In particular, they do not want to stop work, fix problems, and risk criticism. At the first *NUMMI* (New United Motor Manufacturing, Inc) Toyota plant in America, Japanese management visited the plant after six months and saw that employees were afraid to pull the *andon* (lamp) cord—the cord that causes a trouble lamp to turn on and that starts a countdown timer to stop the production line. Workers had not stopped the line enough to fix their impediments. The management pulled the *andon* cord to stop the production line then and there, to communicate to the workers that their biggest impediment was their reluctance to stop the line. Stopping the line makes problems visible so they are fixed properly. “No problem is a problem” is the Japanese management mantra.

The team must consult the *Product Owner* when things are not going well. Not only that, the *Development Team* should agree with the *Product Owner* on how to quickly address major problems that affect reaching the *Sprint Goal*.

Therefore:

When high on the burndown, try a technique used routinely by pilots. When bad things happen, execute the Emergency Procedure designed specifically for the problem.

Do not delay execution while trying to figure out what is wrong or what to do. In a fighter aircraft, you could be dead in less time than it takes to figure out what is going on. It is the responsibility of the *ScrumMaster* to make sure the team immediately executes the Scrum *Emergency Procedure*, preferably by mid-*Sprint*, when things are going off-track. This will require careful coordination with the *Product Owner*, yet kaizen mind requires execution of this pattern even when the *Product Owner* is not available. Great teams act without permission and ask for forgiveness later (see *Community of Trust*).

Scrum Emergency Procedure: (do only as much as necessary)

1. Change the way the team does the work. Do something different.
2. Get help, usually by offloading backlog to someone else.
3. Reduce scope.
4. Abort the *Sprint* and replan.
5. Inform management how the emergency affects release dates.

Teams often want to reduce scope when they encounter difficulty. Great teams find a way to instead execute a different strategy to achieve the *Sprint Goal*. In the 2005–6 football (soccer) season, John Terry, Chelsea's captain and center back, had to take over as goalkeeper in a game against Reading after Petr Cech suffered a fractured skull, and then the substitute goalkeeper who replaced him, Carlo Cuddicini, was carried off unconscious before halftime. Terry made two fine saves and Chelsea won the game 2-0. Similarly in software, adopting new practices that remove waste can multiply performance while drastically cutting effort.

When multiple teams are working on the same products, one team can often pass backlog to another team who has slack. The company PatientKeeper, a pioneer in agile development in the medical sector, automated this strategy ([2]). If a team was behind, it could assign *Sprint Backlog* Items to another team. If the second team could not take them, they passed it to a third team. If the third team could not take them, all three met to decide what to do. This automatically leveled the loading of backlog across teams so they could all finish together.

Reducing scope early so the team can finish planned work is better than coasting into failure. The organization can inspect and adapt to problems rather than be surprised. See *Teams That Finish Early Accelerate Faster*.

Aborting the *Sprint* (stop the line) may be the best option, particularly if the team consistently fails to deliver. Only the *Product Owner* may decide whether to abort the *Sprint*: as bad as things may be, the Product Owner may judge that the business payoff may not be worth it, or that aborting the Sprint may otherwise have long-term negative consequences in the market or the business.

After terminating the Sprint the team typically convenes a brief Sprint Planning for an abbreviated Sprint (to stay on cadence as per *Organizational Sprint Pulse*; see also *Follow the Moon*) to achieve the *Sprint Goal*, if possible, and to deliver as much value as possible. Alternatively, the team may convene a more protracted *Retrospective* to explore and rectify problems in the environment and the team's Scrum implementation, and then replan and move on to the next *Sprint*. But, again, much of the value in *Sprint* termination comes in making it publicly visible that there are fundamental impediments that keep the team from doing its job. A visible problem is one that the team can fix.

Sprint termination sends a strong message throughout the organization that something is wrong and increases the capability of removing impediments that cause failure. One playful Scrum tradition (or at least metaphor) is the “Abnormal Termination of Sprint Ceremony” which is ostensibly carried out in the lobby of corporate headquarters, where the *Development Team* members gather to lay on their backs, scream, and flail their arms and legs in the air to let off steam. The intent is to make it visible that the *Product Owner* has abrogated the team’s commitment.

The *Scrum Team* executes this pattern, and it is particularly useful for high-performing teams. For teams that are serious about kaizen (see *Kaizen and Kaikaku*), Scrum is an extreme sport and they enter into a *Sprint* with some risk in order to go faster. Their primary risk is emergent requirements or unexpected technical problems as the team has addressed most other causes of failure. The team may need to use this pattern every third or fourth Sprint, particularly when implementing new technologies and pushing the state of the art. However, for most emergencies, great teams will recover and meet *Sprint Goal*s. And if they stop the line (abort the *Sprint*) they will poka-yoke ([3]) their process so the same problem does not recur.

***Poka-yoke** (ポカヨケ) [poka joke] is a Japanese term that means “fail-safing” or “mistake-proofing.” A poka-yoke is any mechanism in a lean manufacturing process that helps an equipment operator avoid (yokeru) mistakes (poka). Its purpose is to eliminate product defects by preventing, correcting, or drawing attention to human errors as they occur.*

Making problems visible is part of kaizen mind; see *Kaizen and Kaikaku*.

✣ ✣ ✣

The team will learn to rapidly respond to change in a disciplined way and overcome challenges. In many organizations, when things are not going well, teams are not thinking clearly and are frustrated and demotivated. They fail to understand the cause of their problems and the way to fix them. Executing the *Emergency Procedure* will train the team to focus on success and systematically remove impediments. Great teams will surprise themselves with their ability to overcome adversity and move from strength to strength. It increases chances for successfully delivering a *Potentially Shippable Product Increment* both in the short term and long term. The team will feel it is doing all it can when using Emergency Procedure to get back on the right track, out of both professional pride (see *Team Pride*) and *Product Pride*.

You can use *Emergency Procedure* in a more disciplined way to raise transparency into unmanaged requirements with the pattern *Illegitimus Non Interruptus*.

See also *Take No Small Slips*.

This pattern anticipates use by highly disciplined teams. If a team is using this too often (e.g., more often than once every four *Sprints*) and is not improving its value, quality, and rate of delivery, then the team might reflect about whether something is fundamentally wrong in the environment or in the team’s use of Scrum. It is usually better for young teams to do their best to deliver, to take the *Sprint* to the end, then fail the *Sprint*. Away from the heat of battle in the *Sprint Retrospective*, the team can explore the drivers for failure and plan kaizen. Some process improvements may help the team resort to *Emergency Procedure in* analogous future situations.

Adapted from: Scrum Patterns. Unknown Publish Date. Emergency Procedure [ONLINE] Available at: https://sites.google.com/a/scrumplop.org/published-patterns/product-organization-pattern-language/emergency-procedure [Accessed 7 December 2019].