White Paper

Moving Your Scrum Downfield

The Six Essential Traits of the Game

by Gunther Verheyen
independent Scrum Caretaker

April 2020 - September 2023

Gunther Verheyen
Author of
- 97 Things Every Scrum Practitioner Should Know
- Scrum - A Pocket Guide
Founder of Ullizee Inc.
“I call myself a Scrum Caretaker.

Scrum is used to manage an ever-increasing variety of work—in and beyond software and (new) product development. Despite this wide spread, many seem to get stuck at interpreting and debating the formal rules of Scrum. *As if a language can be learned by studying and discussing the dictionary.* This publication shines a more holistic light on Scrum and focuses on the *why* of things—agnostic of the naming of things. It describes the six essential traits that make Scrum work. These rather implicit traits, too often disregarded, are crucial to get unstuck, evolve to a more unconsidered use of Scrum and regain focus on the purpose of the game. These traits are indicative of your Scrum coming to life and the first step to start moving your Scrum downfield.

1. Scrum Is Simple, Yet Sufficient

Scrum supports people in addressing *complex* challenges and derive value from them, with *value* being a very different purpose than *volume* is.

Complex challenges are highly unpredictable and cannot be tackled with predefined or copy-paste solutions. Scrum is simple in the sense that it defines no more than a limited set of rules. That set is sufficient to devise a way of working specific and fitting to time and context and continually optimize toward creating the most valuable outcomes. This does imply revising, adding, and improving work, management, people, and organizational practices.

In a nutshell, Scrum requires a Scrum Master to foster an environment where (repeatedly):

1. A Product Owner orders functions and solutions for value against an overarching product vision.
2. A team of Developers creates valuable Increments against an overarching Sprint Goal.
3. All players figure out what to work on next and how to best organize for that.

Through its minimalistic design, every element in Scrum serves a purpose. Leaving out parts or not playing by the rules covers up problems (rather than expose them) and limits—and eventually eliminates—substantial benefits.

Many tactics to apply the rules exist. Scrum is a skeleton process that can wrap new practices and render existing practices superfluous. Devise, apply, and tune patterns, practices, and techniques to better fit, not to twist or overburden your Scrum.

The potential of Scrum unfolds when players play by the rules that apply and explore how tactics, interactions, behaviors, and the six essential traits—of which *simplicity* is the first—make it work.

Many struggle with the deliberate incompleteness of Scrum and demand exact instructions that universally apply—regardless of the people involved, environment, tools, business, and markets. This desire for precision contradicts the complexity of reality and the reality of complexity.

2. Scrum's DNA

Scrum is grounded in the management principles of *Self-organization* and *Empiricism*. They are entwined and form Scrum's DNA.

Self-organization asserts that the people undertaking complex work know best how to organize for that work. No external forces can do that better for them. Scrum sets the boundaries within which they are invited to use their intelligence and creativity to act with agility and collaboratively optimize for valuable outcomes.

Empiricism (or *empirical process control*) asserts that forward-looking decisions in complex work are best based on experience, observed results and proven outcomes of experimentation. Scrum implements empiricism via its methodical approach of inspection and adaptation upon transparency of the work being undertaken and results being produced.

Scrum engages people in sharing or acquiring the insights and skills to collectively perform complex work (self-organization), while employing an iterative-incremental approach to make the best possible progress (empiricism). It requires players to regularly stop, reflect, gather feedback, and learn from observation and inspection in order to continue or change course as needed, in order to re-organize, improve, adapt.

Self-organization

Self-organization is the process of people forming organized groups around problems or challenges without external work plans or instructions being imposed on them. No single person can know better how to organize for complex work than a group of skilled people accountable for that work. Self-organization is employed when a group of people collectively organizes, manages, and performs its work.

Regarding the use of "(Team of) Developers"

There is no single term that covers all types of complex work for which Scrum is employed.

In this publication "Team of" is added where Scrum officially mentions only "Developers."

Although 'developer' applies to all involved in creating and sustaining a product, it is too easily misunderstood as (software) programmer.

Although the official "Developers" is plural, the notion of 'team' is explicitly added to emphasize the need to move as a unit up the field.

A term for the combined Scrum roles, for which Scrum uses "Scrum Team," is not used here.

Self-organization *is*, it happens. As any human individual has the inherent quality to self-organize, self-organization doesn't need to be instructed nor empowered. Rather, it requires removal of barriers that prevent people to apply this natural ability. Impediments to self-organization are not in people, but in processes, procedures, and organizational constructs. This is how external authorities are most effective: by removing organizational barriers to self-organization.

A shared (visual) workspace is an important enabler for self-organization. It forms a bounded environment that allows people to optimize focus, collaboration, and shared-ness and benefit from the fastest information exchange possible. Self-organization is most effective in such workspace.

Empiricism

Empirical process control implies closed-loop feedback so that actual outcomes are regularly inspected and validated against desired outcomes. This is a self-correcting process as unwanted variances or results are eliminated or corrected through adaptation in the next or in future system runs.

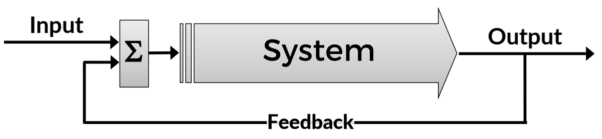

Exhibit 1: A closed-loop feedback system

Empiricism requires and creates transparency. Reality is exposed for inspection purposes in order to allow sensible adaptations. The player-inspectors take forward-looking decisions based on information that reflects their actual situation. Inspections of incomplete or twisted versions of reality lead to pointless adaptations–and even detrimental deviations. Transparency serves the process of inspection and adaptation and implies that all required information is available for the player-inspectors. It does not mean that any piece of information should be available for anyone.

From the need for transparency follows the need for standards and agreements to work against and inspect upon. They are transparent in the sense that they are agreed, followed, visible, accessible, and comprehensible. In Scrum, the definition of Done is particularly important in providing transparency over work to be done and work actually done.

The frequency at which inspections and adaptations are performed should be such that sufficient work can be performed for meaningful inspection, while not impeding the opportunity to adapt to important new insights or evolutions.

3. Players Demonstrate Accountability

A sustainable complex adaptive system does not spring from individual heroism or from hierarchical power. It requires people from different backgrounds and domains combining their skills, talents, experiences, insights, and personalities to work, learn, and improve together.

The increased emphasis on peer collaboration makes leadership more subtle, somewhat dispersed, and even transient in time and place. Leadership becomes a distributed quality instead of an expression of the assertiveness and dominance of some individual(s).

The complementarity of the accountabilities of *Product Owner*, *team of Developers*, and *Scrum Master* instantiates such collaborative spirit. As they collaborate, they additionally develop new relationships with consumers, stakeholders, and others. Tensions are a natural part of this process.

Accountability is not in titles or in functions. It cannot be demanded or instructed. Rather than installing measures and means to 'hold people accountable' Scrum fosters an environment where self-organizing people *demonstrate* accountability.

Product Owner

'Product Owner' is a one-person player role to connect consumers, stakeholders, and teams of Developers. This is very different from dominating all communication or preventing direct interactions. The core accountability is to optimize value for the people receiving the work, for the organization funding it, and for the people performing it. This view on value can be extended to value to society, the environment, the planet.

'Product' is the vehicle to create value. A product is a tangible or non-tangible good or service, a value stream, or is something more abstract like the outcome of specific processes or actions. Without a clearly identified 'product'–and its consumers and stakeholders–a Product Owner is hardly effective in optimizing the value delivered. As a result, Scrum is hardly used effectively.

A Product Owner, self-evidently minding the long-term viability of the product, leads through a product vision. A product vision captures why the product exists. It encapsulates what makes the product worthwhile buying, consuming, and investing in. A product vision helps emerge specific product goals, uncover product functions and solutions, and validate whether value is actually being created via product Increments.

A Product Owner orders envisioned functions and solutions in a Product Backlog and assures that they are known and understood for how they potentially deliver value. A Product Owner represents the needs of many and is the sole person deciding over ordering the Product Backlog and spending the game budget.

The Product Owner is accountable, whether doing the above or having others do it.

Team of Developers

'Team of Developers' is a multi-person player role consisting of a group of people self-organizing around the challenge of turning functions and solutions from the Product Backlog into observable, Done output. The core accountability is to create such usable and valuable Increments of product no later than by the end of a Sprint, and thereby sustain the resultant product.

Teams of Developers work with the Product Owner in identifying the most important functions and solutions for a Sprint. They perform and manage all activities involved in delivering such *forecast* of Product Backlog in a Sprint. They work with the Product Owner as needed to optimize the Sprint's outcome captured in a Sprint Goal.

Beyond managing their own work, teams of Developers also self-organize for that work in terms of size, skills, expertise, and availabilities—by using the process of inspection and adaptation.

Team size and work organization are such that teams of Developers work at a sustainable pace, a pace that can be maintained indefinitely. Working in Sprints serves rhythm and cadence while improving focus, but is not for burning out people.

To assure alignment and consistency of the collaborative work, a team of Developers has an agreed set of work practices and standards to be applied to the collective work. In particular, the qualities and criteria that must be met for an Increment to be declared "Done" are captured in a *definition of Done*. This ensures a shared understanding of the state of an Increment when inspected. What a team of Developers requires in terms of skills, tools, and practices is a function of what is defined as Done—not the other way round.

A team of Developers is always accountable as a whole, regardless of the type of work needed or performed, or of specialized skills and focus areas of individual players. No sub-teams, titles, or hierarchy exist within a team of Developers.

Scrum Master

'Scrum Master' is a one-person player role acting as a game master to assure that the rules of the game are known and understood and to support players to uncover better ways to play. The core accountability is to guide self-organization toward the creation of valuable outcomes.

This requires certain management skills, traits, and insights, but it is very different from being a traditional manager. A Scrum Master has no formal power over people, careers, or incentives. A Scrum Master does not control progress, budget, quality, or tasks. A Scrum Master does support fellow players figure out how to manage these in Scrum.

A Scrum Master leads through a vision of what can be achieved with Scrum in terms of engagement, creativity, and a humanized workplace. A Scrum Master induces the continual desire to become better players. Scrum Masters understand that embracing Scrum doesn't occur overnight, but is a journey. They are patiently impatient.

A Scrum Master facilitates players by making sure that *impediments* are removed, elements that hinder or block the work but are beyond self-organizational control. Think organizational expectations, directives, processes, procedures, or structures that are at odds with the rules, values, principles, or purpose of Scrum. This may require coaching for behavioral change at any level of organizations. Forming alliances with fellow Scrum Masters comes naturally when having to challenge the status quo, organizational or otherwise.

Removing impediments, facilitating events, teaching techniques, supporting teams, educating the organization, keeping the road open to perform, to work, to innovate, to be creative are some of the services that a Scrum Master provides. How interventionist the services are is a mirror of the state of Scrum within an organization.

Scrum Master accountability cannot be eliminated. Complex work, turbulent circumstances, and changing environments inevitably give rise to situations and challenges for which players need support, guidance, observations, and coaching. No sports team has no coach. Scrum Masters can only strive to become invisibly present.

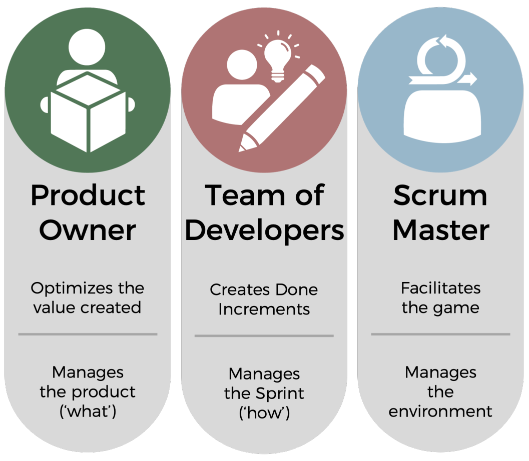

Exhibit 2: Accountability in Scrum

4. Transparency for a Flow of Value

Complex challenges are highly unpredictable. Deriving value from them requires more than harnessing or 'managing' change. It requires capitalizing on new insights, accumulated experience, and unforeseen opportunities. This is why Scrum implements empiricism.

Transparency, as the foundation of empiricism, holds that work done and work to be done can be fully understood at any point in time—regardless of past hopes, dreams, and plans. The extent to which the Scrum artifacts of *Product Backlog*, *Sprint Backlog*, and *Increment* reflect reality will impact the results and outcomes, down to the risk of results and outcomes becoming useless and even harmful when including unaccountable deviations as a result of complete or partial absence of transparency. The definition of Done is particularly important in making the reality of observed work fully transparent.

The Scrum artifacts support maintaining a *flow of value* at a macro level by uncovering, ordering, and delivering functions and solutions. A clearly identified 'product,' as the vehicle to deliver value, provides focus and purpose to the use of the Scrum artifacts. Without such clear identification, optimizing for value is hardly possible and Scrum is hardly used effectively.

Product Backlog

Product Backlog is an emergent, ordered list of functions and solutions that the Product Owner deems as potentially valuable, and exposes factors that enhance or obstruct the flow of value—like goals, dependencies, and constraints.

Product Backlog is the primary source of work and progress in Scrum. While the Product Owner is accountable for its ordering, teams of Developers are responsible for its sizing. Product Backlog is the single source of work for teams of Developers. A Product Owner keeps everyone with a vetted interest updated on Product Backlog progress.

A Product Backlog reminds all players that right-time conversations are required to elaborate on what the work entangles. This is very different from exhaustive lists holding exhaustively detailed specifications. The primary value of Product Backlog is in providing direction and uncovering opportunities to create value—not in completeness, precision, or detail. Product Backlog is not a tool for trying to predict the inherently unpredictable.

The Product Owner is accountable that the Product Backlog exists and is ordered. To maximize transparency and assure fast and consistent decisions, one product has one Product Backlog ordered by one Product Owner, regardless the number of teams of Developers involved.

Sprint Backlog

Sprint Backlog is the emergent plan for a Sprint. Teams of Developers use it to manage the improvements and the work done or anticipated to most effectively turn selected functions and solutions from the Product Backlog into Done Increments, therein guided by the Sprint Goal. Only a team of Developers decides what is in their Sprint Backlog and how to manage it.

Sprint Backlog is a living artifact that is kept accurate and realistic. Throughout a Sprint new insights on how to achieve the Sprint Goal might surface. Work that becomes obsolete or was unanticipated is removed from or added to the Sprint Backlog, no later than on a daily basis.

Increments of integrated, Done output emerge from the collaborative work of teams of Developers. Sprint Backlog is used to keep track of the progress of a Sprint and not miss out on useful adaptations. If the actual progress impacts the forecast of Product Backlog, the Product Owner is consulted. If no movement is radiated, the empirical process might be in danger and the team of Developers might be in need of help.

Increment

Quality and progress are assured by repeatedly producing Increments upon defined, agreed work practices and standards. An Increment is a solid, meaningful body of work that is available for inspection no later than by the end of a Sprint, at which point it already has been released or is in a releasable state. An Increment has no hidden or "undone" work. It is guaranteed to comply with the definition of Done. Adaptations to the definition of Done may reveal work that must be done for previous Increments first. Product is the integrated resultant of all Increments.

Increments incorporate additions, expansions, improvements, eliminations, and modifications. Regardless the time of their availability, Increments can be released when they are Done *and* are deemed useful. An Increment assures that one or more Done functions and solutions become available for the consumers of the product.

It is good—if not essential—practice for a Product Owner to know about usage, satisfaction, or other indicators of the impact and value of Increments. This must be validated against the product vision, goals, or other ambitions. Otherwise decisions to optimize value remain doused with opaqueness.

The definition of Done is particularly important in assuring that an Increment is usable, stable, and of high quality. Quality best encompasses more, however. A Done product should exhibit the qualities an organization envisions and wants to be known for by its user base. A Done product Increment should exhibit the qualities needed to deliver or result in value. Ideally, the definition of Done would echo *valuable* and exceed *releasable*.

The state of an Increment described in the definition of Done is not a function of the skills available in a team of Developers, or the tools and practices applied, but the other way round.

5. Closing the Loops

Effective use of Scrum entails closing loops—more loops than meet the eye.

In Scrum all work is organized in Sprints. Sprints have a fixed length which is never more than four weeks. Sprint length allows weighing progress in terms of tangible output against the ability to adapt upon feedback gathered at the macro level. Sprint length is a factor of minimal stability and reflects the contextual need for the frequency at which to inspect and adapt at a strategic and tactical level.

For the external world, Sprint is the only unit of work and time—not days, hours, or work estimates. A Sprint is a shielded playfield that allows creating Done Increments of work. What happens in a Sprint, stays in the Sprint. External authorities are most effective by preventing distractions or interventions during a Sprint, and not being one.

Complex work innately incorporates divergent options. Players are required to converge regularly *within a Sprint*, not just toward its end. Full closure by the end of a Sprint is needed to preserve unburdened adaptability at the macro level. Open work is not a valuable outcome of a Sprint. It blocks creativity and openness and is a liability for adaptability and future value creation.

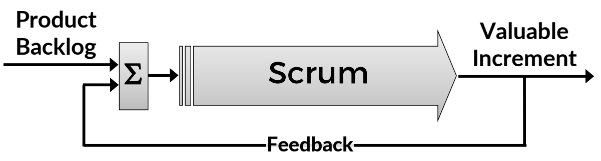

Exhibit 3: Closed-loop feedback with Scrum

A dramatic way of preventing proper closure is the termination of an on-going Sprint. It occurs only when the work of a Sprint becomes fundamentally and totally invalid and cannot be replaced.

Planning a Sprint

Planning a Sprint is the opening act of a Sprint. The players consider and collaborate on *what* the most valuable functions and solutions are, *why* they are worthwhile, and *how* to turn them into cohesive and observable output.

Scrum defines the *Sprint Planning* event for this purpose. The event takes as long as needed to meet its goal, but never more than eight hours. That goal is thus to choose direction and allow embarkment–rather than predicting exact output.

The players choose how to organize Sprint Planning. Experience shows how some select multiple functions and solutions first, identify the work involved, and update the selection as they find they have more or less capacity. Others iterate dynamically between the *what* and the *how* of individual functions or solutions. Others do something in between. The players decide but assure alignment with one Sprint Goal.

The Product Owner shares and clarifies Product Backlog so the team of Developers can anticipate and map out work and activities to be performed. Only the team of Developers determines how much it reasonably can achieve within the Sprint. A forecast of work for a Sprint is tuned to the Sprint length–not the other way round.

The event meets its goal if Sprint Backlog holds enough work to embark and the team of Developers has decided *how* to start turning selected functions and solutions into observable output. Additional work and new insights are assessed and managed via Sprint Backlog in due time during the Sprint. The Sprint Goal expresses what makes the Sprint worth the energy and the investment–an envisioned state of the product or some other meaningful outcome.

Managing for Progress

Protected from external distractions, the players continue engaging in collaborative work to move as a unit up the field toward achieving the next game level, the Sprint Goal.

At any time Sprint Backlog reflects what is needed to achieve the Sprint Goal. Problems and progress toward the Sprint Goal can be shared and discussed at any time, but no later than on a daily basis. Scrum defines the *Daily Scrum* event for this purpose, with a time-box of 15 minutes. It serves to identify and agree over the upcoming Sprint work, in particular until the next Daily Scrum.

The team of Developers applies its agreed work practices to incrementally create Done output, satisfy the Sprint Goal, and generate valuable outcomes. Feedback loops of development are closed regularly and repeatedly within the Sprint to assure alignment and consistency as well as to catch problems and errors early. Consider the propagation of errors that, if not detected early, potentially endanger proper closure of the Sprint.

If during the Sprint it is discovered that substantially different, more, or less work is needed or possible than planned for, the forecast is renegotiated with the Product Owner. When a body of work complies with the definition of Done, it can be shared as an Increment with the product's consumers–upon the Product Owner's consent.

Sprint Reflections

Sprint reflections serve closure of a Sprint. The players and invited guests collaborate on *why* the Sprint was undertaken, *what* functions and solutions are delivered in Increment(s) against this Sprint Goal, and what Product Backlog currently holds. They reflect on *how* the Sprint went to define their working process for the next game round. Inspection without adaptation is pointless.

The definition of Done is particularly important in assuring that all have a shared understanding of the qualities and state of the work being observed. It is imperative that an Increment of product has all the characteristics of the final…product. Paper reports or presentations don't meet that demand.

Scrum defines two events for these closing activities: *Sprint Review* and *Sprint Retrospective*.

Sprint Review starts with a focus on *why* a Sprint was undertaken and *what* was achieved. The Product Owner connects the team(s) of Developers with invited stakeholders and consumers for this purpose. Product Backlog is used to assess progress along with major changes that impacted it. All attendees collaborate and share ideas on how to further improve the value of the product in the next Sprint(s), which is captured in an updated Product Backlog. The event takes the time needed to meet this goal, but never more than four hours.

Sprint Retrospective serves a deep dive on *how* the Sprint went. Many aspects of the work are covered, including (but not limited to): team engagement, Done-ness and the potential value of the Increment(s), the use of Scrum, practices and techniques, social aspects, collaboration, team values, team agreements. Although improvements may be implemented at any time, they are identified no later than at the Sprint Retrospective. A Sprint Retrospective provides a formal opportunity to reflect on and define the actual work process. The event takes as long as needed to meet this goal, but never more than three hours.

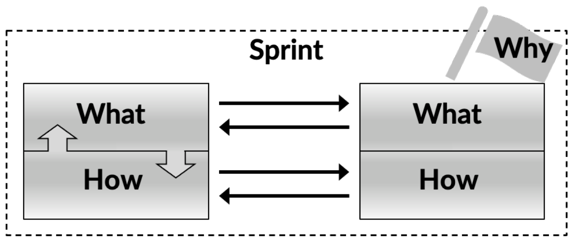

Exhibit 4: Information circulation at the macro level

6. The Scrum Values

Scrum is more effective through spirited collaboration, for which it provides a frame. Scrum, actually, is more about behavior than it is about process. Values drive behavior. The Scrum Values of *Commitment*, *Focus*, *Openness*, *Respect*, and *Courage* give direction to working in Scrum, as a compass. All interactions and decisions, the steps taken, the tactics chosen to apply the rules of Scrum should re-enforce these values, not diminish or undermine them.

- ✦ *Commitment* shows in the working spirit of the players in terms of motivation, dedication, and engagement. It is about the actions and the intensity of the efforts, not about exact and predicted content and volume of output.
- ✦ *Focus* is increased through the balanced but distinct accountabilities of Scrum. Time-boxing all work encourages players to focus on what is imminent now as the future in complex environments is highly uncertain.
- ✦ *Openness* shows in the attitude of all players. It is reflected in their collaboration, interactions, and relationships, in how they deal with change and with differences in skills, personalities, and opinions.
- ✦ *Respect* is essential for self-organizing groups to navigate complexity. It requires respecting the people aspects of the work, rules, agreements, skills, practices, ideas, experience, and viewpoints, while at times respectfully challenging them.
- ✦ *Courage* is much needed in complex work. Confronted with uncertainty, it takes courage to act, to self-organize, to uphold quality, to embrace imperfection and ambiguity, to apply empiricism, to turn change into a source of inspiration and innovation, to enact Scrum and to live…the Scrum Values.

How the six essential traits of the game are indicative of Scrum coming to life:

1. Scrum Is Simple, Yet Sufficient. The players unfold the potential of Scrum by using the *simple* rules that apply and explore how tactics, interactions, behaviors, and the six essential traits make Scrum work.

2. Scrum's DNA. The players form a *self-organizing* unit around the challenge of collectively creating observable, Done Increments of work, while employing *empiricism* to manage all work and progress.

3. Players Demonstrate Accountability. The players contribute to high-quality, valuable system outcomes through spirited collaboration, and sharing and challenging rules, agreements, skills, practices, ideas, and viewpoints.

4. Transparency for a Flow of Value. The players use the Scrum artifacts to uphold transparency over all the work done and the work to be done, manage for a flow of value, and preserve the ability to capitalize on unforeseen opportunities.

5. Closing the Loops. The players regularly and repeatedly close the many intertwined loops within a Sprint toward full closure by the end of a Sprint, thereby preserving unburdened adaptability at the macro level.

6. The Scrum Values. The Scrum Values of *Commitment*, *Focus*, *Openness*, *Respect*, and *Courage* take prominence in the behaviors, relationships, actions, and decisions of the players and their ecosystem.

About the author

Gunther Verheyen calls himself an *independent Scrum Caretaker* on a journey of humanizing the workplace with Scrum. He is a longtime Scrum practitioner who started applying Scrum in 2003 and worked with various teams and organizations in various industries since then. He has published two acclaimed books about Scrum, was the partner of Ken Schwaber (co-creator of Scrum) and Director of the "Professional Scrum" series at Scrum.org.

Gunther Verheyen ventured into IT and software development after graduating as an Industrial Engineer in Electronics in 1992. His Agile journey started with eXtreme Programming and Scrum in 2003. Years of dedication and employing Scrum in diverse circumstances followed. In 2010, Gunther became the inspiring force behind some large-scale enterprise transformations. In 2011, he became a Professional Scrum Trainer.

Gunther founded Ullizee-Inc in 2013 to partner exclusively with Ken Schwaber. He represented Ken and his organization Scrum.org in Europe, while maintaining its Professional Scrum series and shepherding its global network of trainer/coaches. Gunther is co-creator of Agility Path, EBM (Evidence-Based Management), and the Nexus framework for Scaled Professional Scrum at Scrum.org.

Since 2016, Gunther is continuing his journey of humanizing the workplace as an independent Scrum Caretaker—a connector, writer, trainer, and speaker. He helps organizations re-imagine their Scrum and re-emerge the organization around their Scrum to create a more humane and thereby more productive workplace.

Gunther created his book *Scrum – A Pocket Guide* in 2013, with a 2nd edition published in 2019 and a 3rd edition in 2021. He was the editor of the book *97 Things Every Scrum Practitioner Should Know* (2020); a collection of essays from field experts across the world. Several translations of his work are available.

When not traveling for Scrum and humanizing the workplace, Gunther lives and works in Antwerp, Belgium.

Check out his website for more information: https://guntherverheyen.com/