Extracted from:

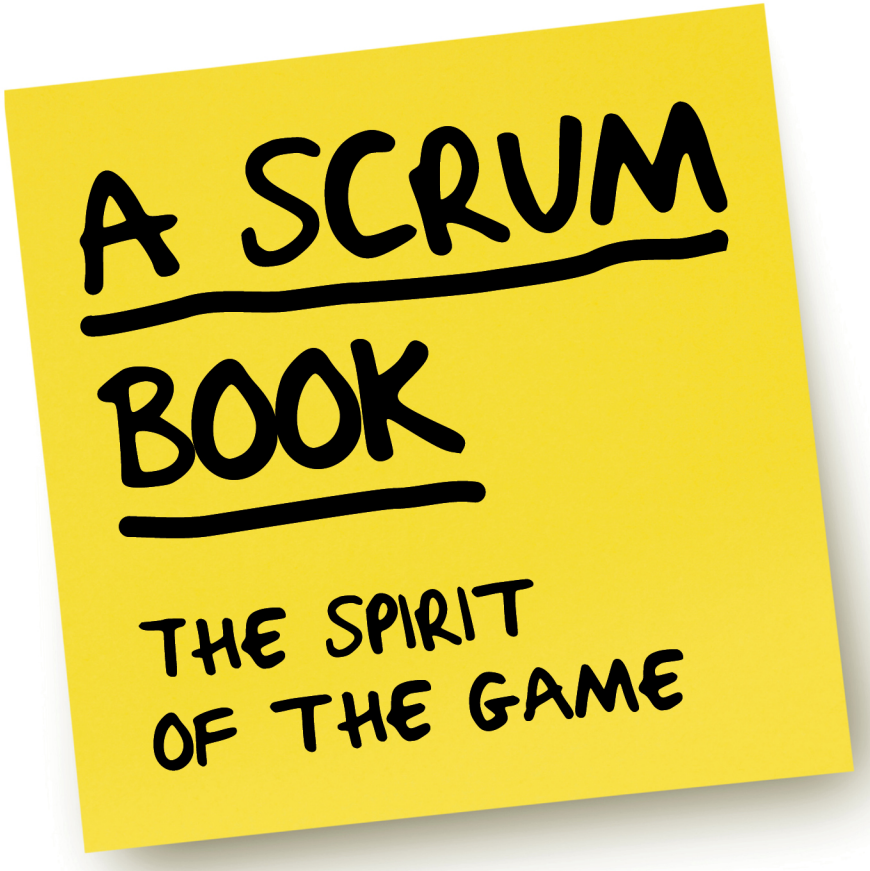
A Scrum Book

The Spirit of the Game

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers



Jeff Sutherland
James O. Coplien
The Scrum Patterns Group

edited by Adaobi Obi Tulton

A Scrum Book

The Spirit of the Game

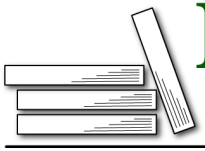
Jeff Sutherland
James O. Coplien

Lachlan Heasman
Mark den Hollander
Cesário Oliveira Ramos

and The Scrum Patterns Group:

Esther Vervloed, Neil Harrison, Kiro Harada, Joseph Yoder,
June Kim, Alan O'Callaghan, Mike Beedle, Gertrud Bjørnvig,
Dina Friis, Ville Reijonen, Gabrielle Benefield, Jens Østergaard,
Veli-Pekka Eloranta, Evan Leonard, and Ademar Aguiar

The Pragmatic Bookshelf
Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Adaobi Obi Tulton
Copy Editor: Sean Dennis
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-671-6
Book version: P1.0—August 2019

¶10 Cross-Functional Team

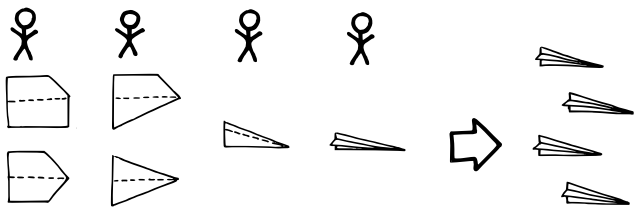
Confidence stars: **

The team as a whole should embody all the talent necessary to deliver product.

...the *¶7 Scrum Team* is organizing its development effort, and are choosing team members, or are assessing how to grow the team skill set.

✣ ✣ ✣

The *Scrum Team* is not able to work autonomously because it does not have all the skills required to complete a complex network of tasks. By depending on skills from people outside the team, the team cannot take ownership for finishing their tasks. It reduces the team's influence on the time it takes to finish and can tarnish the quality of the end result. The core lean principles of consistency and reduced rework depend on short feedback loops. Most complex development requires people with numerous talents from areas as diverse as human factors, engineering excellence, and quality assurance. It is rare that one finds all these talents in members of a single team, let alone in any given individual. Teams often organize around areas of competence: birds of a feather flock together. This is sometimes called a functional organization. Yet, it is costly to coordinate these functions across team boundaries because efficient communications take place between those who share the context of the current work—and that is usually the team members.



A complex product might require that the team has mastered numerous skills to develop *Done* functionality (see *¶82 Definition of Done*). When work calls

for an additional individual for each required skill, the team will become too big to be effective. You might be tempted to not extend the skill set in the team and to instead introduce external dependencies. On the other hand, you might choose to give the work to the team so they can develop and learn the required skill. But learning takes time.

Local learning can become local optimization, where a group of specialists develop practices and processes that optimize their work. Specialization, local practices and processes can all be sources of efficiency in an organization, but can also create problems at the group boundary. To attack these problems, an organization can define "contracts" outlining how to work with each other (e.g., service requests). Such contracts might specify the nature of the work that an organization is willing to do along with expected durations for responses to requests. Anyone needing the group's specialization would have to use these contracts. However, this can slow development of the product as a whole even though it increases the efficiency of the local department. Again, there may be a need for additional coordination groups within the organization to manage these boundary contracts, to negotiate exceptions or ensure all parties understand what is required, and to make sure each team meets its obligations to other teams—and to the customer, according to the obligations of these contracts.

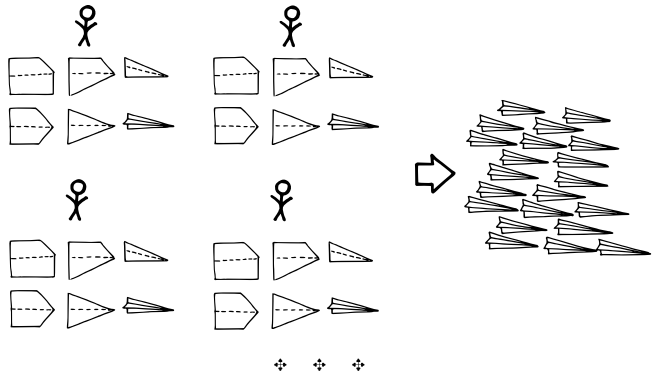
New products—or new versions of existing products—each create a new world for their customers. Because you cannot know in advance what this new world will be like, you must focus on learning and experimentation as the product evolves. The team must find lessons in its experience with actual customer use of each product increment rather than according to some pre-arranged plan. And the team must integrate these lessons across the product development process. Everyone recognizes the advantages of local flow, autonomy, and control that come from working as an individual within a step of the process or within a functional area. However, such a work structure moves everyone (except the person doing the last step) further from the end user and the broad insights that come from interactions at that boundary. This may result in suboptimal local functions but greater optimization across the entire product development process.

Therefore:

Each *Scrum Team* should include all talent necessary to deliver *Done* functionality.

It's good to pay attention to skill set coverage when initially creating the team, but it's more important that the charter team members share enthusiasm for

the ¶39 Vision and that they have a track record of learning new things. Because things change over time, it is unlikely that the team will be able to foresee all its long-term skill needs from the beginning.



Instead of changing team membership as the need for new skills emerges, grow the people internally and strive for ¶9 Small Teams and ¶15 Stable Teams. Over time, cross-train team members so they grow their skill sets to accommodate more and more competency areas (see ¶123 Moderate Truck Number on page ?). This will increase the ability of the team to work as an ¶16 Autonomous Team. With *Cross-Functional Team*s it becomes easier to ¶131 Distribute Work Evenly on page ?.

The team members now have all the opportunities to learn secondary skills. They can swarm (see ¶25 Swarming: One-Piece Continuous Flow) on ¶55 Product Backlog Items (*PBI*s), which increases learning opportunities and optimizes flow to help get functionality to *Done* fast. The development of secondary skills makes the team more flexible so any member can stand in for another that has become unavailable. The team always makes progress and is autonomous.

Scrum is silent on how to handle a missing competency. Let common sense prevail; for example, ask for help from another team, or subcontract large work increments that might surprise the team. It is understandable if the team needs such help now and then. But if the team finds they frequently depend on external help, then they should view this as an impediment and take measures (such as training, reorganization, or hiring) to remedy the situation.

For example, a team of software programmers may find themselves building a product in an area outside their native expertise, such as pharmaceuticals or aerospace. It is tempting to appoint a person on the team for each of the underrepresented competencies, perhaps by consulting with an external domain expert. However, the team representative may not know how much

they don't know, and may not even know what questions to ask the domain experts. Most domain experts carry domain expertise as tacit knowledge, so they are not in a position to recover the right insights to support the software person in a proper implementation. It is crucial that team members understand the implications of domain considerations on the implementation and have a thorough knowledge of both the business and solution space. In a recent article, Jesse Watson of Amazon noted that it's crucial that both of these factors co-exist "within one skull."[13] It is better to bring the expert on board to the team and to broaden the knowledge with cross-training. But remember *Small Teams*: adding specialists may grow the team to a point where teamwork diminishes to almost nothing.

These teams naturally act like "feature teams" (see *¶4 Conway's Law*) because most *PBI*s are feature-shaped: marketable elements of revenue-generating functional product increment. If *Cross-Functional Team*s develop the product, then handoffs naturally disappear from the *¶41 Value Stream*: the team itself can develop any feature without outside support or intervention. Involving multiple teams introduces delays in feedback loops, increases the waste (*muda*) of rework, and creates inconsistency (*mura*) between development stages in the *Value Stream*.

A study published in the Harvard Business Review of two corporations, one organized functionally and the other by product, suggests that cross-functional teams offer the best features of both organizational structures (see "Organizational Choice: Product vs. Function" in *Harvard Business Review 46 [WL68]*).

¶42 Set-Based Design is a technique that keeps developers engaged in many disciplines and domains that may be relevant to the business, even if they ultimately don't make it through to the current product. Such practice broadens the expertise base of the team and enterprise and reduces the probability that the team will be surprised by the need to master some new discipline.

As the team integrates new lessons there will be new product ideas. Change will proceed quickly (and must be allowed to proceed quickly). Change will be the norm rather than the exception. This requires small organizations where everyone knows what is happening: organizations that can embrace change, work across specializations, regularly deliver value and are, for want of another term: agile.

13. Jesse Watson. "The Hard Thing about Software Development." LinkedIn.com, https://www.linkedin.com/pulse/hard-thing-software-development-jesse-watson, 12 July 2017 (accessed 4 July 2018).

A Game

Assemble several small teams who will compete in a game to make and fly paper airplanes. Each team member may make only one fold at a time, and then must switch to work on another plane. No plane may have more than 15 folds. It must be at least 15 centimeters long and 8 centimeters wide. It must have a blunt tip at least 2 centimeters wide. To qualify as a quality product the plane must fly 3 meters horizontally when the tester tests it. The tester may test each plane only once.

Try the game, and apply Scrum patterns (hint: *Swarming: One-Piece Continuous Flow*) to optimize the number of quality planes produced in a one-minute Sprint.