Small Scale Scrum

LEIGH GRIFFIN

LEIGH IS AN ENGINEERING MANAGER and Agile Coach working for Red Hat. This role allows him realise his passion for Coaching and helping individuals and teams improve how they work on a day to day basis. Leigh has a PhD in Group Communication and helped co-supervise a research dissertation on Small Scale Scrum.

AGNIESZKA GANCARCZYK

AGNIESZKA IS AN ASSOCIATE CONSULTANT working for Red Hat Consulting and developing software solutions for customers in small 1-3 person Agile teams. She spent a year researching Small Scale Scrum for her final thesis and has recently graduated with MSc in Computing (Communications Software). Apart from software development, her interests lay in project and people management.

ABOUT OPENSOURCE.COM

What is Opensource.com?

OPENSOURCE.COM publishes stories about creating, adopting, and sharing open source solutions. Visit Opensource.com to learn more about how the open source way is improving technologies, education, business, government, health, law, entertainment, humanitarian efforts, and more.

Submit a story idea: https://opensource.com/story

Email us: open@opensource.com

Chat with us in Freenode IRC: #opensource.com

SUPPORTED BY RED HAT

INTRODUCTION

CHAPTERS

GET INVOLVED | ADDITIONAL RESOURCES

Introduction

Here's how the scrum agile methodology can help teams of three or fewer work more efficiently.

AGILE IS FAST becoming a mainstream way industries act, behave, and work as they look to improve efficiency, minimize costs, and empower staff. Most software developers naturally think, act, and work this way, and alignment towards Agile software methodologies has gathered pace in recent years.

VersionOne's 2018 State of Agile report [1] shows that Scrum and its variants remain the most popular implementation of Agile. This is in part due to changes made to the Scrum Guide's [2] wording in recent years that make it more amenable to non-software industries.

The guide also changed its stance towards the team size needed to implement Scrum: The previous recommended team size of 3-9 was changed to a range of one to three participants.

Like in Agile, scale is a hot topic in Scrum, with a battleground of competing frameworks vying for supremacy. Scaling frameworks have focused on allowing multiple teams to coordinate in a seamless manner and, in essence, promote Scrum at large scale. However, scaling down (below the recommended minimum team size) is not gaining traction, even though it is the way many sectors operate.

For example, paid-for consultancy work typically involves one or two people working on a short-term project; since consultant costs are often charged by the hour or day, a smaller team provides maximum value for the customer. Open source contributors very often work alone, albeit part of a much larger community. And college students completing final-year projects or research assignments usually work in solo mode, with small teams forming in some cases.

All of these formats can follow an Agile way of work. Scrum's principles and execution have been applied to small-scale teams; however, they're often applied in a way that leads to something slipping. In our experience, that is quality. We set out to investigate whether we could maintain a high-level of quality and output with a reduced team size.

Small Scale Scrum

The result of our research is Small Scale Scrum, a long-awaited and novel concept in Agile "supporting planning, developing, and delivering production-quality software solutions." Small, in this case, is a maximum of three people. According to an extensive survey, this is something the industry, customers, and small development teams have been asking for, as this team size is more realistic for their needs. Organizations that engage in consultancy projects are particularly asking for ways to run Scrum projects with one or two developers, due to the industry-wide acknowledgement of the approach's efficiency. Given that Scrum has amended its guidebook to include non-software industries, this approach could benefit many different sectors.

In this guide, we'll explain Small Scale Scrum—what it is, how it works, and how it can help small teams work better. In future articles, we'll cover the Small Scale Agile Manifesto, framework, and principles, describe our survey results, and where to go next.

Links

[1] https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report

[2] https://www.scrumguides.org/scrum-guide.html

Small Scale Agile Manifesto

These six values enhance agile methodologies to help smaller teams work more efficiently.

THE "AGILE MANIFESTO" is an umbrella term that describes and governs several lightweight and fuller Agile methodologies for handling IT teams and projects. Scrum, Kanban, Lean Development, Crystal, and Extreme Programming (XP) are among the most popular and lightweight Agile approaches.

While Small Scale Scrum fits into the Agile Manifesto, six of its additional values, described below, should complement and enhance Agile for smaller teams.

Wide communication over narrow communication

Maintaining narrow communication with a project's manager is essential, but wider communication offers more value. Wider communication includes all of a team's stakeholders, including the product owner, ScrumMaster, and all team members. Excellence through application of best principles in every team-to-team or team-to-customer communication is very important. Therefore, team members are encouraged to take time to prepare before meetings to best accommodate changes and ensure productive outcomes. Regularly using the team's preferred communication channels quickly and effectively creates a welcoming environment.

Team feature delivery over individual responsibility

An individual team member's responsibility for delivering single features is considered standard practice in software projects. Members of small teams, however, are expected to have much broader involvement in the project lifecycle, so they need to take mutual responsibility for delivering work items. In this approach, teamwork plays an important role, with team members working together as a single unit to ensure the project's success from the bottom up. This can be achieved with techniques including support for remote team members, fair workload, pair programming, code review, and shadowing. Remote team members work towards the common goal, but are not geographically co-located; fair workload is to ensure that the team's workload is divided fairly; pair programming is focused on writing and reviewing source code together in real time; code review, also known as peer review is focused on viewing and reading source code after or as an interruption of implementation; shadowing is an on-the-job learning and it's commonly used for in-house training.

Quality delivery over speed of development

Rapid development and high-quality delivery are expectations in every customer engagement. While speed of development is important, quality delivery has much greater impact on the project's continuous success. Investing time in quality development and testing by using quality-assurance techniques, tools, mentoring, and training can help the team continuously excel.

Multiple project responsibilities over fixed assignment

Fixed project responsibility where team members typically have one role, with the overall team containing enough

skills to be self sufficient for the roles to be filled, is a standard practice in Agile, but the real value for small-scale teams comes from members taking additional roles (within reason). For example, an engineer may become the frontend developer, backend developer, quality engineer, or user interface (UI) designer. The idea behind this approach is to ensure that the small team is as self-sufficient as possible. For small teams to adopt multiple responsibilities, the workload must be fair and the project's processes must be streamlined. This can be achieved by continuously reviewing and improving work conditions and simplifying processes to help the team focus on delivery. Coaching should be offered to give team members guidelines to help them improve their skills.

Accelerating innovation over marginal request-driven thinking

Request-driven thinking where specific business requests are followed strictly, is important in enterprise engagements, but innovation is what customers value the most. In such engagements the customer is the only stakeholder and their view and voice are followed to the letter. Planting innovation is critical to get the team and customers to think outside the project's defined requirements so they can envision the final solution with the most creative and optimal architectures, requirements, and designs.

Customer growth over customer engagement

A successful customer engagement is very important to a project, but it's not sufficient for building and maintaining a strong and successful relationship with the customer. For small teams, it's important to approach business engagement from the customer's business-success perspective. Growth or creation of business is the customer's highest priority for a software solution, therefore it should be the team's highest priority.

A version of this article was originally published on Medium and is republished with permission.

Small Scale Scrum Framework

This agile approach is designed for small teams whose members play multiple roles.

SCRUM IS A LEADING candidate for the implementation of Small Scale Agile for many reasons including its popularity, developers' preference, high success rates for Scrum adoption and project deliveries, and strong principles and values including focus, courage, openness, commitment, and respect.

Small Scale Scrum [1] can be best described as "a people-first framework defined by and for small teams (a maximum of three people) and supporting planning, developing, and delivering production-quality software solutions." The proposed framework centers around the concept of team members occupying multiple roles on any project.

Small Scale Scrum is valuable due to its strong support for the small, distributed teams found in organizations all over the world. Small teams need new ways to meet customers' continuously growing expectations for rapid delivery and high quality, and Small Scale Scrum's guidelines and principles help address this challenge.

The **Project Backlog** is a list of everything known to be required in the project. It is created before any development work begins and is maintained by a development team. The Project Backlog contains development tasks and software requirements. It replaces the traditional Product and Sprint backlogs.

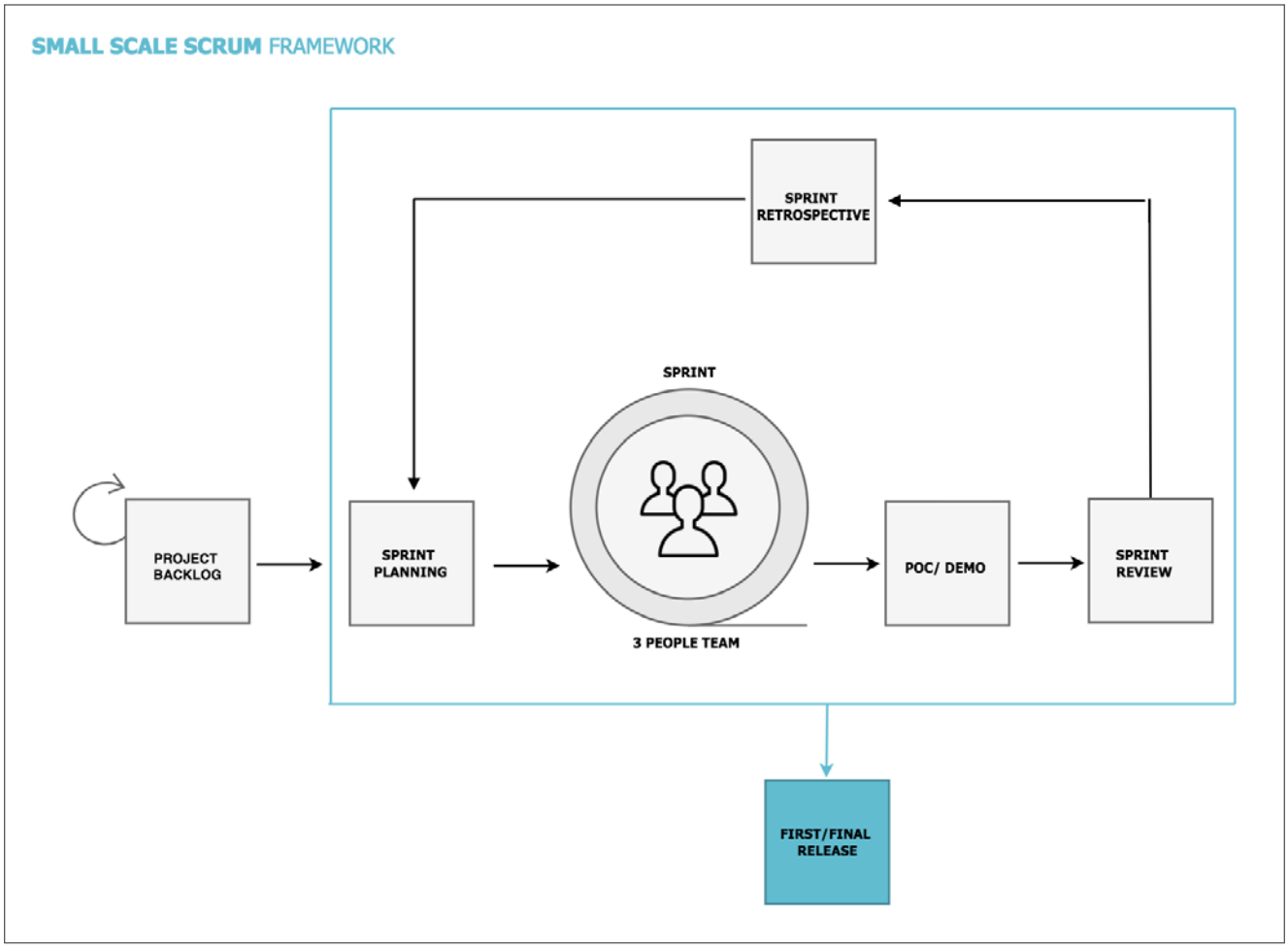
The **Sprint Planning** meeting is time-boxed (i.e., set in advance for a specified amount of time) and focused on planning work for the upcoming Sprint. The meeting is run by the development team and advance planning is required to keep it concise. Sprint Planning is value-based. At this point, requirements that will be worked on in the upcoming Sprint should be clear and contain approved acceptance criteria. Capacity, typically measured as velocity, is not used; instead, the team has to take an educated guess. Velocity only emerges after three or more Sprints, which is usually after the Small Scale Scrum projects are finished.

The **Three-People Team** is comprised of the development team and the project manager (most of the time). The development team is expected to be involved in gathering and clarifying business requirements, doing development work, performing quality testing, and releasing software to the customer.

The **Proof of Concept/Demo** is a realization of the small work completed in the Sprint to showcase progress in development. Any deviations or incomplete work are discussed during the POC/demo.

The **Sprint Review** meeting is organized and run by the development team and project manager (if involved in the project) and attended by the customer or customer team. The Sprint Review is time-boxed and contains demonstrations of Sprint work and a short outline of work completed/not completed, bugs raised, and known and/or descoped issues (if any). Customer feedback is gathered at the end of the demonstration and incorporated into future Sprints. Very little (if any) preparation is required to keep the meeting short and structured.

The **Sprint Retrospective** meeting is organized and run by the development team and project manager (if involved)



and may be attended by the customer and/or customer team. The Sprint Retrospective is time-boxed and requires no advance preparation. Due to the small size of the development team and the Sprint builds (with a smaller number of features delivered), this meeting is relatively short. The Sprint Retrospective takes place after the Sprint Review and before the next Sprint Planning meeting.

The Sprint Review, Sprint Retrospective, and Sprint Planning may be combined into a single meeting which we term the Sprint Termination, lasting approximately 90 minutes. These meetings are concise, structured (thanks to advance preparation), and must not include any unnecessary/unrelated discussions.

The **First/Final Release** is the project's end result. The development team runs the tests, verifies the completed work, confirms fixes, and signs off on the release build.

The Scrum Guide's [2] recommended time-box guidelines per ceremony should be considered valid for Small Scale Scrum ceremonies.

Links

[1] https://github.com/agagancarczyk/small-scale-scrum/blob/master/Agnieszka-Gancarczyk-20060828-Final-Dissertation.pdf

[2] https://www.scrumguides.org/scrum-guide.html

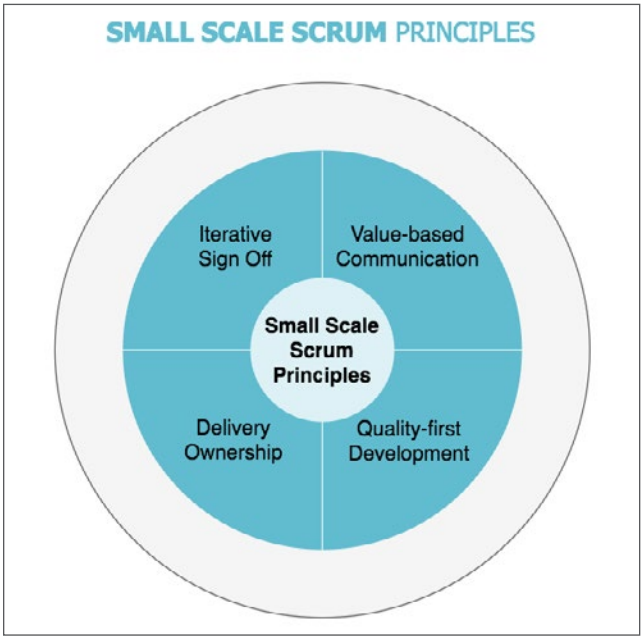
A related version of this article was originally published on Medium and is republished with permission.

Small Scale Scrum Principles

These four principles address communication, processes, and benefits of scrum methodologies for smaller workgroups.

SMALL SCALE SCRUM PRINCIPLES address: the preferred approach towards communication; the processes introduced to ensure the highest quality of delivery; and the benefits behind implementing Small Scale Scrum for the business.

Small Scale Scrum principles [1] are non-negotiable.



Value-based Communication includes inner (within the development team) and outer (with the customer) communication. It is focused on delivering a value. Understanding the solution, its purpose, and its desired functionality depends upon effective communication. Initiating and maintaining communication between parties requires openness and dedication to look for the best solution to a given functionality request or fix. It may also lead to further discussions around progress made in software delivered, which can, in turn, uncover omissions in the requirements. With finite time, the focus should be put on valuable and urgent communication as opposed to communication deviating from importance and usefulness. This aligns with the Eisenhower Matrix [2] of making urgent vs important decisions to help prioritise communication flows.

Quality-first Development focuses on taking a quality approach to software development during each Sprint. This means ensuring that features are delivered according to acceptance criteria, the solution is bug-free (or at least free from any evident bugs), any inconsistencies in the software are removed, the solution is tested, and any edge-case bugs and omitted features are reported, logged, and considered by the customer.

Delivery Ownership is about the development team taking initiative in driving software delivery on a Sprint-to-Sprint basis. Enabling the team to consult the customer directly positively impacts the team's overall performance and the customer's satisfaction. Elimination of any micromanagement-related barriers is critical to allow the team to take ownership in delivering software solutions.

Iterative Sign Off focuses on reducing technical debt and identifying gaps in requirements through an iterative sign-off approach. As the solution evolves and matures, business requirements change. With iterative development, functionality delivered within a Sprint should be signed off upon the Sprint's completion. Similarly, requirements for the upcoming Sprint should be signed off on before it begins.

Links

[1] https://github.com/agagancarczyk/small-scale-scrum/blob/master/Agnieszka-Gancarczyk-20060828-Final-Dissertation.pdf

[2] https://www.eisenhower.me/eisenhower-matrix/

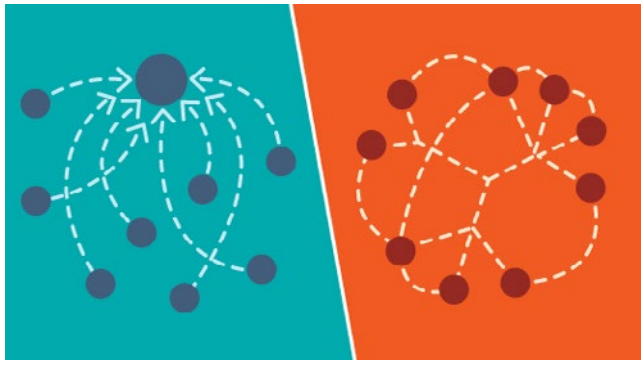
A related version of this article was originally published on Medium and is republished with permission.

Small Scale Scrum VS. Large Scale Scrum

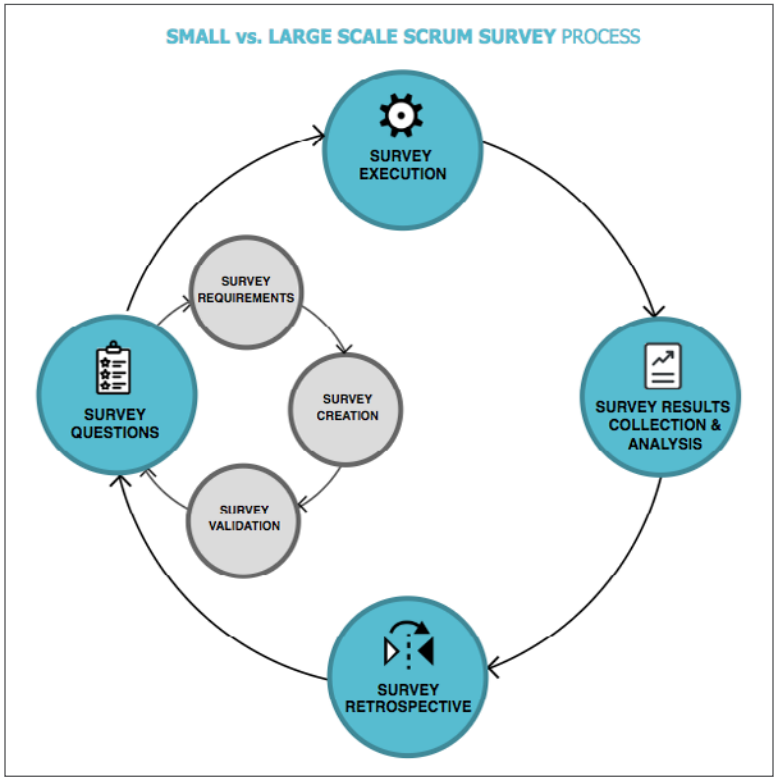
We surveyed individual members of small and large scrum teams. Here are some key findings.

FOLLOWING the publication of the scrum framework, we wanted to collect feedback on how teams in our target demographic (consultants, open source developers, and students) work and what they value. With this first opportunity to inspect, adapt, and help shape the next stage of Small Scale Scrum, we decided to create a survey to capture some data points and begin to validate some of our assumptions and hypotheses.

Our reasons for using the survey were multifold, but chief among them were the global distribution of teams, the small local data sample available in our office, access to customers, and the industry's utilization of surveys (e.g., the Stack Overflow Developer Survey 2018, HackerRank 2018 Developer Skills Report [1], and GitLab 2018 Global Developer Report [2]).



The scrum's iterative process was used to facilitate the creation of the survey shown to the right.



The survey [3], which we invite you to complete, consisted of 59 questions and was distributed at a local college (Waterford Institute of Technology [4]) and to Red Hat's consultancy and engineering teams. Our initial data was gathered from the responses of 54 individuals spread across small and large scrum teams, who were asked about their experiences with agile within their teams.

Here are the main results and initial findings of the survey:

- A full 96% of survey participants practice a form of agile, work in distributed teams, think scrum principles help them reduce development complexity, and believe agile contributes to the success of their projects.
- Only 8% of survey participants belong to small (one- to three-person) teams, and 10 out of 51 describe their typical project as short-lived (three months or less).
- The majority of survey participants were software engineers, but quality engineers (QE), project managers (PM), product owners (PO), and scrum masters were also represented.
- Scrum master, PO, and team member are typical roles in projects.

- Nearly half of survey respondents work on, or are assigned to, more than one project at the same time.
- Almost half of projects are customer/value-generating vs. improvement/not directly value-generating or unclassified.
- Almost half of survey participants said that their work is clarified sometimes or most of the time and estimated before development with extensions available sometimes or most of the time. They said asking for clarification of work items is the team's responsibility.
- Almost half of survey respondents said they write tests for their code, and they adhere to best coding practices, document their code, and get their code reviewed before merging.
- Almost all survey participants introduce bugs to the codebase, which are prioritized by them, the team, PM, PO, team lead, or the scrum master.
- Participants ask for help and mentoring when a task is complex. They also take on additional roles on their projects when needed, including business analyst, PM, QE, and architect, and they sometimes find changing roles difficult.
- When changing roles on a daily basis, individuals feel they lose one to two hours on average, but they still complete their work on time most of the time.
- Most survey participants use scrum (65%), followed by hybrid (18%) and Kanban (12%). This is consistent with results of VersionOne's State of Agile Report [5].
- The daily standup, sprint, sprint planning and estimating, backlog grooming, and sprint retrospective are among the top scrum ceremonies and principles followed, and team members do preparation work before meetings.
- The majority of sprints (62%) are three weeks long, followed by two-week sprints (26%), one-week sprints (6%), and four-week sprints (4%). Two percent of participants are not using sprints due to strict release and update timings, with all activities organized and planned around those dates.
- Teams use planning poker [6] to estimate (storypoint) user stories. User stories contain acceptance criteria.
- Teams create and use a Definition of Done [7] mainly in respect to features and determining completion of user stories.
- The majority of teams don't have or use a Definition of Ready [8] to ensure that user stories are actionable, testable, and clear.
- Unit, integration, functional, automated, performance/load, and acceptance tests are commonly used.
- Overall collaboration between team members is considered high, and team members use various communication channels.
- The majority of survey participants spend more than four hours weekly in meetings, including face-to-face meetings, web conferences, and email communication.
- The majority of customers are considered large, and half of them understand and follow scrum principles.
- Customers respect "no deadlines" most of the time and sometimes help create user stories and participate in sprint planning, sprint review and demonstration, sprint retrospective, and backlog review and refinement.
- Only 27% of survey participants know their customers have a high level of satisfaction with the adoption of agile, while the majority (58%) don't know this information at all.

These survey results will inform the next stage of our data-gathering exercise. We will apply Small Scale Scrum to real-world projects, and the guidance obtained from the survey will help us gather key data points as we move toward version 2.0 of Small Scale Scrum. If you want to help, take our survey [3]. If you have a project to which you'd like to apply Small Scale Scrum, please get in touch.

Links

[1] https://research.hackerrank.com/developer-skills/2018/
[2] https://about.gitlab.com/developer-survey/2018/
[3] https://docs.google.com/forms/d/e/1FAIpQLScAXf52KMEiEzS68OOIsjLtwZJto_XT7A3b9aB0RhasnE_dEw/viewform?c=0&w=1
[4] https://www.wit.ie/
[5] https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report
[6] https://en.wikipedia.org/wiki/Planning_poker
[7] https://www.scruminc.com/definition-of-done/
[8] https://www.scruminc.com/definition-of-ready/

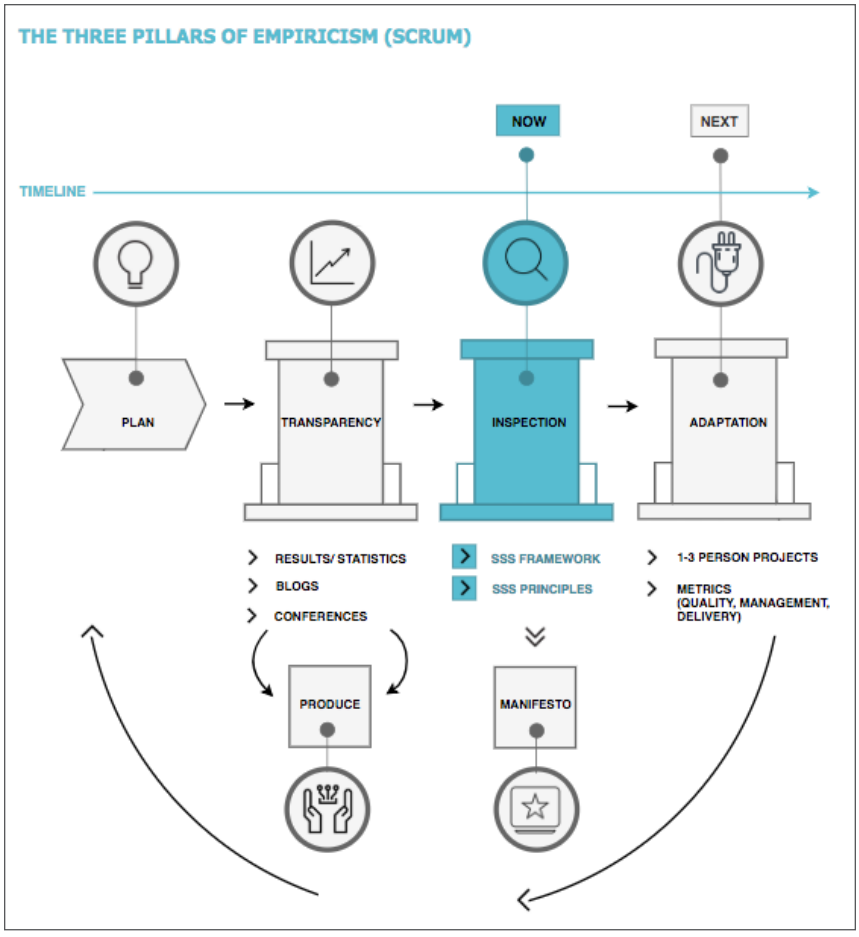
The next steps for Small Scale Scrum

We plan to test the Small Scale Scrum framework in real-world projects involving small teams.

SCRUM IS BUILT on the three pillars of Inspection, Adaptation, and Transparency. Our empirical research is really the starting point in bringing Scrum, one of the most popular Agile implementations, to smaller teams. As presented in the diagram below, we are now taking time to inspect this framework and principles by testing them in real-world projects.

We plan to implement Small Scale Scrum in several upcoming projects. Our test candidates are customers with real projects where teams of one to three people will undertake short-lived projects (ranging from a few weeks to three months) with an emphasis on quality and outputs. Individual projects, such as final-year projects (over 24 weeks) that are a capstone project after four years in a degree program, are almost exclusively completed by a single person. In projects of this nature, there is an emphasis on the project plan and structure and maximizing the outputs that a single person can achieve.

We plan to metricize and publish the results of these projects and hold several retrospectives with the teams involved. We are particularly interested in metrics centered around quality, with a particular emphasis on quality in a software engineering context and management, both project management through the lifecycle with a customer and management of the day-to-day team activities and the delivery, release, handover, and sign-off process. Ultimately, we will retrospectively analyze the overall framework and principles and see if the Manifesto we envisioned holds up to the reality of executing a project with small numbers. From this data, we will produce the second version of Small Scale Scrum and begin a cyclic pattern of inspecting the model in new projects and adapting it again.



We want to do all of this transparently. This series of articles is one window into the data, the insights, the experiences, and the reality of running Scrum for small teams whose everyday challenges include context switching, communication, and the need for a quality delivery. A follow-up series of articles is planned to examine the outputs and help develop the second edition of Small Scale Scrum entirely in the community.

We also plan to attend conferences and share our knowledge with the Agile community. We are advising colleges and sharing our structure and approach to managing and executing final-year projects. All our outputs will be freely available in the open source way.

Given the changes to recommended team sizes in the Scrum Guide, our long-term goal and vision is to have the Scrum Guide reflect that teams of one or more people occupying one or more roles within a project are capable of following Scrum.

GET INVOLVED

If you find these articles useful, get involved! Your feedback helps improve the status quo for all things DevOps.

Contribute to the Opensource.com DevOps resource collection, and join the team of DevOps practitioners and enthusiasts who want to share the open source stories happening in the world of IT.

The Open Source DevOps team is looking for writers, curators, and others who can help us explore the intersection of open source and DevOps. We're especially interested in stories on the following topics:

- DevOps practical how to's
- DevOps and open source
- DevOps and talent
- DevOps and culture
- DevSecOps/rugged software

Learn more about the Opensource.com DevOps team: https://opensource.com/devops-team

ADDITIONAL RESOURCES

The open source guide to DevOps monitoring tools

This free download for sysadmin observability tools includes analysis of open source monitoring, log aggregation, alerting/visualizations, and distributed tracing tools.

Download it now: The open source guide to DevOps monitoring tools

The ultimate DevOps hiring guide

This free download provides advice, tactics, and information about the state of DevOps hiring for both job seekers and hiring managers.

Download it now: The ultimate DevOps hiring guide

The Open Organization Guide to IT Culture Change

In The Open Organization Guide to IT Culture Change, more than 25 contributors from open communities, companies, and projects offer hard-won lessons and practical advice on how to create an open IT department that can deliver better, faster results and unparalleled business value.

Download it now: The Open Organization Guide to IT Culture Change

WRITE FOR US

Would you like to write for Opensource.com? Our editorial calendar includes upcoming themes, community columns, and topic suggestions: https://opensource.com/calendar

Learn more about writing for Opensource.com at: https://opensource.com/writers

We're always looking for open source-related articles on the following topics:

Big data: Open source big data tools, stories, communities, and news.

Command-line tips: Tricks and tips for the Linux command-line.

Containers and Kubernetes: Getting started with containers, best practices, security, news, projects, and case studies.

Education: Open source projects, tools, solutions, and resources for educators, students, and the classroom.

Geek culture: Open source-related geek culture stories.

Hardware: Open source hardware projects, maker culture, new products, howtos, and tutorials.

Machine learning and AI: Open source tools, programs, projects and howtos for machine learning and artificial intelligence.

Programming: Share your favorite scripts, tips for getting started, tricks for developers, tutorials, and tell us about your favorite programming languages and communities.

Security: Tips and tricks for securing your systems, best practices, checklists, tutorials and tools, case studies, and security-related project updates.