AGILE PROJECT MANAGEMENT

SCRUM FRAMEWORK

Author

ALESSANDRO INGROSSO

Created by WikiCourseBuilder

1. Stand-up meeting

1.1.

A stand-up meeting (or simply "stand-up") is a meeting in which attendees typically participate while standing. The discomfort of standing for long periods is intended to keep the meetings short.

1.2. Notable examples

By tradition, the Privy Council of the United Kingdom meets standing.

1.3. Software development

A stand-up in the computing room
Some software development methodologies envision daily team-meetings to make commitments to team members. The daily commitments allow participants to know about potential challenges as well as to coordinate efforts to resolve difficult and/or time-consuming issues. The stand-up has particular value in Agile software development processes, such as Scrum or Kanban, but can be utilized in context of any software-development methodology.

The meetings are usually timeboxed to between 5 and 15 minutes, and take place with participants standing up to remind people to keep the meeting short and to-the-point. The stand-up meeting is sometimes also referred to as the "stand-up", "morning rollcall" or "daily scrum".

The meeting should take place at the same time and place every working day. All team members are encouraged to attend, but the meetings are not postponed if some of the team members are not present. One of the crucial features is that the meeting is a communication opportunity among team members and not a status update to management or stakeholders.TEMPLATE[cite_journal, last Stray, first Viktoria, last2 Sjøberg, first2 Dag, last3 Dybå, first3 Tore, date 2016-01-11, title The daily stand-up meeting: A grounded theory study, url http://www.sciencedirect.com/science/article/pii/S0164121216000066, journal Journal of Systems and Software, volume 114, issue 20, pages 101–124, doi 10.1016/j.jss.2016.01.004, pmid , access-date 2016-03-11] Although it is sometimes referred to as a type of status meeting, the structure of the meeting is meant to promote follow-up conversation, as well as to identify issues before they become too problematic. The practice also promotes closer working relationships in its

frequency, need for follow-up conversations and short structure, which in turn result in a higher rate of knowledge transfer – a much more active intention than the typical status meeting. Team members take turns speaking, sometimes passing along a token to indicate the current person allowed to speak. Each member talks about progress since the last stand-up, the anticipated work until the next stand-up and any impediments, taking the opportunity to ask for help or collaborate.

Team members may sometimes ask for short clarifications and make brief statements, such as "Let's talk about this more after the meeting", but the stand-up does not usually consist of full-fledged discussions.

1.4. Three Questions

Scrum-style stand-ups convene daily to re-plan in-progress development. Though it may not be practical to limit all discussion to these three questions, the objective is to create a new sprint plan within the time box (less than 15 minutes), while deferring discussions about impediments until after the event is complete. Team members briefly (a maximum of one minute per team member) address three questions as input to this planning:

Whereas Kanban-style daily stand-ups focus more on:

1. What did I do yesterday that helped the development team meet the sprint goal?
2. What will I do today to help the development team meet the sprint goal?
3. Do I see any impediment that prevents me or the development team from meeting the sprint goal?

1. What obstacles are impeding my progress?
2. (looking at the board from right to left) What has progressed?

2. software development process

2.1.

In software engineering, a software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle. The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.

Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming.

Some people consider a life-cycle "model" a more general term for a category of methodologies and a software development "process" a more specific term to refer to a specific process chosen by a specific organization. For example, there are many specific software development processes that fit the spiral life-cycle model. The field is often considered a subset of the systems development life cycle

2.2. History

The software development methodology (also known as SDM) framework didn't emerge until the 1960s. According to Elliott (2004) the systems development life cycle (SDLC) can be considered to be the oldest formalized methodology framework for building information systems. The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle—from inception of the idea to delivery of the final system—to be carried out rigidly and sequentially"Geoffrey Elliott (2004) Global Business Information Technology: an integrated systems approach. Pearson Education. p.87. within the context of the framework being applied. The main target of this methodology framework in the 1960s was "to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines".

Methodologies, processes, and frameworks range from specific proscriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a "sponsor" or "maintenance" organization distributes an official set of documents that

describe the process. Specific examples include:

2010s

It is notable that since DSDM in 1994, all of the methodologies on the above list except RUP have been agile methodologies - yet many organisations, especially governments, still use pre-agile processes (often waterfall or similar). Software process and software quality are closely interrelated; some unexpected facets and effects have been observed in practice

Since the early 2000s scaling agile delivery processes has become the biggest challenge for teams using agile processes.

Among these another software development process has been established in open source. The adoption of these best practices known and established processes within the confines of a company is called inner source.

1. Structured programming since 1969
2. Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology

1. Structured systems analysis and design method (SSADM) from 1980 onwards
2. Information Requirement Analysis/Soft systems methodology

1. Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s
2. Rapid application development (RAD), since 1991
3. Dynamic systems development method (DSDM), since 1994
4. Scrum, since 1995
5. Team software process, since 1998
6. Rational Unified Process (RUP), maintained by IBM since 1998
7. Extreme programming, since 1999

1. Agile Unified Process (AUP) maintained since 2005 by Scott Ambler
2. Disciplined agile delivery (DAD) Supersedes AUP

1. Scaled Agile Framework (SAFe)
2. Large-Scale Scrum (LeSS)

2.3. Practices

Several software development approaches have been used since the origin of information technology, in two main categories . Typically an approach or a combination of approaches is chosen by management or a development team .

"Traditional" methodologies such as waterfall that have distinct phases are sometimes known as software development life cycle (SDLC) methodologies, though this term could also be used more generally to refer to any methodology. A "life cycle" approach with distinct phases is in contrast to Agile approaches which define a process of iteration, but where design, construction, and deployment of different pieces can occur simultaneously .

Continuous integration is the practice of merging all developer working copies to a shared mainline several times a day. Grady Booch first named and proposed CI in his 1991 method, although he did not advocate integrating several times a day. Extreme programming (XP) adopted the concept of CI and did advocate integrating more than once per day – perhaps as many as tens of times per day.

Software prototyping is about creating prototypes, i.e. incomplete versions of the software program being developed.

The basic principles are:

A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problems, but this is true for all software methodologies.

Various methods are acceptable for combining linear and iterative systems development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

There are three main variants of incremental development:

Rapid Application Development (RAD) Model

Rapid application development (RAD) is a software development methodology, which favors iterative development and the rapid construction of prototypes instead of large amounts of up-front

planning. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

The rapid development process starts with the development of preliminary data models and business process models using structured techniques. In the next stage, requirements are verified using prototyping, eventually to refine the data and process models. These stages are repeated iteratively; further development results in "a combined business requirements and technical design statement to be used for constructing new systems".

The term was first used to describe a software development process introduced by James Martin in 1991. According to Whitten (2003), it is a merger of various structured techniques, especially data-driven Information Engineering, with prototyping techniques to accelerate software systems development.Whitten, Jeffrey L.; Lonnie D. Bentley, Kevin C. Dittman. (2003). Systems Analysis and Design Methods. 6th edition. .

The basic principles of rapid application development are:

1. Prototyping is not a standalone, complete development methodology, but rather an approach to try out particular features in the context of a full methodology (such as incremental, spiral, or rapid application development (RAD)).
2. Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
3. The client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.
4. While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

1. A series of mini-Waterfalls are performed, where all phases of the Waterfall are completed for a small part of a system, before proceeding to the next increment, or
2. Overall requirements are defined before proceeding to evolutionary, mini-Waterfall development of individual increments of a system, or
3. The initial software concept, requirements analysis, and design of architecture and system core are defined via Waterfall, followed by incremental implementation, which culminates in installing the final version, a working system.

1. Key objective is for fast development and delivery of a high quality system at a relatively low

investment cost.
2. Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
3. Aims to produce high quality systems quickly, primarily via iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include Graphical User Interface (GUI) builders, Computer Aided Software Engineering (CASE) tools, Database Management Systems (DBMS), fourth-generation programming languages, code generators, and object-oriented techniques.
4. Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.
5. Project control involves prioritizing development and defining delivery deadlines or “timeboxes”. If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not in increasing the deadline.
6. Generally includes joint application design (JAD), where users are intensely involved in system design, via consensus building in either structured workshops, or electronically facilitated interaction.
7. Active user involvement is imperative.
8. Iteratively produces production software, as opposed to a throwaway prototype.
9. Produces documentation necessary to facilitate future development and maintenance.
10. Standard systems analysis and design methods can be fitted into this framework.

2.4. Methodologies

"Agile software development" refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve via collaboration between self-organizing cross-functional teams. The term was coined in the year 2001 when the Agile Manifesto was formulated.

Agile software development uses iterative development as a basis but advocates a lighter and more people-centric viewpoint than traditional approaches. Agile processes fundamentally incorporate iteration and the continuous feedback that it provides to successively refine and deliver a software system.

There are many agile methodologies, including:

The activities of the software development process represented in the waterfall model. There are several other models to represent this process.

The waterfall model is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through several phases, typically:

The first formal description of the method is often cited as an article published by Winston W. RoyceWasserfallmodell > Entstehungskontext, Markus Rerych, Institut für Gestaltungs- und Wirkungsforschung, TU-Wien. Accessed on line November 28, 2007. in 1970 although Royce did not use the term "waterfall" in this article. Royce presented this model as an example of a flawed, non-working model.Conrad Weisert, Waterfall methodology: there's no such thing!

The basic principles are:

The waterfall model is a traditional engineering approach applied to software engineering. A strict waterfall approach discourages revisiting and revising any prior phase once it is complete. This "inflexibility" in a pure waterfall model has been a source of criticism by supporters of other more "flexible" models. It has been widely blamed for several large-scale government projects running over budget, over time and sometimes failing to deliver on requirements due to the Big Design Up Front approach. Except when contractually required, the waterfall model has been largely superseded by more flexible and versatile methodologies developed specifically for software development. See Criticism of Waterfall model.

Spiral model (Boehm, 1988)

In 1988, Barry Boehm published a formal software system development "spiral model," which combines some key aspect of the waterfall model and rapid prototyping methodologies, in an effort to combine advantages of top-down and bottom-up concepts. It provided emphasis in a key area many felt had been neglected by other methodologies: deliberate iterative risk analysis, particularly suited to large-scale complex systems.

The basic principles are:

Other high-level software project methodologies include:

1. Dynamic systems development method (DSDM)
2. Kanban
3. Scrum

1. Requirements analysis resulting in a software requirements specification
2. Software design
3. Implementation
4. Testing

5. Integration, if there are multiple subsystems
6. Deployment (or Installation)
7. Maintenance

1. Project is divided into sequential phases, with some overlap and splashback acceptable between phases.
2. Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
3. Tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase. Written documentation is an explicit deliverable of each phase.

1. Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle.
2. "Each cycle involves a progression through the same sequence of steps, for each part of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program."Barry Boehm (1996., "A Spiral Model of Software Development and Enhancement". In: ACM SIGSOFT Software Engineering Notes (ACM) 11(4):14-24, August 1986
3. Each trip around the spiral traverses four basic quadrants: (1) determine objectives, alternatives, and constraints of the iteration; (2) evaluate alternatives; Identify and resolve risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration.Richard H. Thayer, Barry W. Boehm (1986). Tutorial: software engineering project management. Computer Society Press of the IEEE. p.130
4. Begin each cycle with an identification of stakeholders and their "win conditions", and end each cycle with review and commitment.Barry W. Boehm (2000). Software cost estimation with Cocomo II: Volume 1.

1. Behavior-driven development and business process managementTEMPLATE[Cite_journal, url http://ieeexplore.ieee.org/document/7548916/, doi 10.1109/MS.2016.117, title Modeling Test Cases in BPMN for Behavior-Driven Development, journal IEEE Software, volume 33, issue 5, pages 15-21, year 2016, author1 Lübke, Daniel, author2 van Lessen, Tammo]
2. Chaos model - The main rule is always resolve the most important issue first.
3. Incremental funding methodology - an iterative approach
4. Lightweight methodology - a general term for methods that only have a few rules and practices

5. Structured systems analysis and design method - a specific version of waterfall
6. Slow programming, as part of the larger Slow Movement, emphasizes careful and gradual work without (or minimal) time pressures. Slow programming aims to avoid bugs and overly quick release schedules.
7. V-Model (software development) - an extension of the waterfall model
8. Unified Process (UP) is an iterative software development methodology framework, based on Unified Modeling Language (UML). UP organizes the development of software into four phases, each consisting of one or more executable iterations of the software at that stage of development: inception, elaboration, construction, and guidelines. Many tools and products exist to facilitate UP implementation. One of the more popular versions of UP is the Rational Unified Process (RUP).

2.5. Process meta-models

Some "process models" are abstract descriptions for evaluating, comparing, and improving the specific process adopted by an organization.

1. ISO/IEC 12207 is the international standard describing the method to select, implement, and monitor the life cycle for software.
2. The Capability Maturity Model Integration (CMMI) is one of the leading models and based on best practice. Independent assessments grade organizations on how well they follow their defined processes, not on the quality of those processes or the software produced. CMMI has replaced CMM.
3. ISO 9000 describes standards for a formally organized process to manufacture a product and the methods of managing and monitoring progress. Although the standard was originally created for the manufacturing sector, ISO 9000 standards have been applied to software development as well. Like CMMI, certification with ISO 9000 does not guarantee the quality of the end result, only that formalized business processes have been followed.
4. ISO/IEC 15504 Information technology — Process assessment also known as Software Process Improvement Capability Determination (SPICE), is a "framework for the assessment of software processes". This standard is aimed at setting out a clear model for process comparison. SPICE is used much like CMMI. It models processes to manage, control, guide and monitor software development. This model is then used to measure what a development organization or project team actually does during software development. This information is analyzed to identify weaknesses and drive improvement. It also identifies strengths that can be continued or integrated into common practice for that organization or team.
5. SPEM 2.0 by the Object Management Group
6. Soft systems methodology - a general method for improving management processes

7. Method engineering - a general method for improving information system processes

2.6. In practice

The three basic approaches applied to software development methodology frameworks.

A variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. One software development methodology framework is not necessarily suitable for use by all projects. Each of the available methodology frameworks are best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.Centers for Medicare & Medicaid Services (CMS) Office of Information Service (2008). Selecting a development approach. Webarticle. United States Department of Health and Human Services (HHS). Re-validated: March 27, 2008. Retrieved 27 Oct 2008.

Software development organizations implement process methodologies to ease the process of development. Sometimes, contractors may require methodologies employed, an example is the U.S. defense industry, which requires a rating based on process models to obtain contracts. The international standard for describing the method of selecting, implementing and monitoring the life cycle for software is ISO/IEC 12207.

A decades-long goal has been to find repeatable, predictable processes that improve productivity and quality. Some try to systematize or formalize the seemingly unruly task of designing software. Others apply project management techniques to designing software. Large numbers of software projects do not meet their expectations in terms of functionality, cost, or delivery schedule - see List of failed and overbudget custom software projects for some notable examples.

Organizations may create a Software Engineering Process Group (SEPG), which is the focal point for process improvement. Composed of line practitioners who have varied skills, the group is at the center of the collaborative effort of everyone in the organization who is involved with software engineering process improvement.

A particular development team may also agree to programming environment details, such as which integrated development environment is used, and one or more dominant programming paradigms, programming style rules, or choice of specific software libraries or software frameworks. These details are generally not dictated by the choice of model or general methodology.
Software development life cycle (SDLC)

3. Scrum (software development)

3.1.

Scrum is an agile framework for managing knowledge work, with an emphasis on software development. It is designed for teams of three to nine members, who break their work into actions that can be completed within timeboxed iterations, called "sprints", no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute stand-up meetings, called daily scrums.

Approaches to coordinating the work of multiple scrum teams in larger organizations include Large-scale Scrum (LeSS), Scaled agile framework (SAFe), scrum of scrums, and Scrum@Scale, among others.

3.2. Key ideas

Scrum is a lightweight, iterative and incremental framework for managing product development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved.

A key principle of Scrum is the dual recognition that customers will change their minds about what they want or need (often called requirements volatilityJ. Henry and S. Henry. Quantitative assessment of the software maintenance process and requirements volatility. In Proc. of the ACM Conference on Computer Science, pages 346–351, 1993.) and that there will be unpredictable challenges—for which a predictive or planned approach is not suited. As such, Scrum adopts an evidence-based empirical approach—accepting that the problem cannot be fully understood or defined up front, and instead focusing on how to maximize the team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in market conditions.

Many of the terms used in Scrum (e.g., scrum master) are typically written with leading capitals (e.g., Scrum Master) or as conjoint words written in camel case (e.g., ScrumMaster). To maintain an encyclopedic tone, however, this article uses normal sentence case for these terms—unless they are recognized marks (such as Certified Scrum Master). This is occasionally seen written in all-capitals, as SCRUM. The word is not an acronym, so this is not correct; however, it likely arose due to an early paper by Ken Schwaber which capitalized SCRUM in its title.

While the trademark on the term Scrum itself has been allowed to lapse, so that it is deemed as owned by the wider community rather than an individual, the leading capital is retained—except when used with other words (as in daily scrum or scrum team).

3.3. History

Hirotaka Takeuchi and Ikujiro Nonaka introduced the term scrum in the context of product development in their 1986 Harvard Business Review article, "The New New Product Development Game". Takeuchi and Nonaka later argued in The Knowledge Creating Company that it is a form of "organizational knowledge creation, especially good at bringing about innovation continuously, incrementally and spirally".

The authors described a new approach to commercial product development that would increase speed and flexibility, based on case studies from manufacturing firms in the automotive, photocopier and printer industries. They called this the holistic or rugby approach, as the whole process is performed by one cross-functional team across multiple overlapping phases, where the team "tries to go the distance as a unit, passing the ball back and forth". (In rugby football, a scrum is used to restart play, as the forwards of each team interlock with their heads down and attempt to gain possession of the ball.TEMPLATE[Oxford Dictionaries, Scrum])

In the early 1990s, Ken Schwaber used what would become Scrum at his company, Advanced Development Methods; while Jeff Sutherland, John Scumniotales and Jeff McKenna, developed a similar approach at Easel Corporation, referring to it using the single word Scrum.

In 1995, Sutherland and Schwaber jointly presented a paper describing the Scrum framework at the Business Object Design and Implementation Workshop held as part of Object-Oriented Programming, Systems, Languages & Applications '95 (OOPSLA '95) in Austin, Texas. Over the following years, Schwaber and Sutherland collaborated to combine this material—with their experience and evolving good practice—to develop what became known as Scrum.

In 2001, Schwaber worked with Mike Beedle to describe the method in the book, Agile Software Development with Scrum. Scrum's approach to planning and managing product development involves bringing decision-making authority to the level of operation properties and certainties.

In 2002, Schwaber with others founded the Scrum Alliance
and set up the Certified Scrum accreditation series. Schwaber left the Scrum Alliance in late 2009 and founded Scrum.org which oversees the parallel Professional Scrum accreditation series.

Since 2009, there is a public document called The Scrum Guide that defines a sort of official

version of Scrum and is occasionally revised. In 2018 it was expanded upon with the publication of The Kanban Guide for Scrum Teams.

3.4. Roles

There are three core roles in the Scrum framework. These are ideally co-located to deliver potentially shippable product increments every sprint. Together these three roles form the scrum team. While many organizations have other roles involved with defining and delivering the product, Scrum defines only these three.

The product owner represents the product's stakeholders and the voice of the customer, is responsible for the backlog and accountable for maximising the value that the team delivers. The product owner defines the product in customer-centric terms (typically user stories), adds them to the product backlog, and prioritizes them based on importance and dependencies. A scrum team should have only one product owner (although a product owner could support more than one team). This role should not be combined with that of the scrum master. The product owner should focus on the business side of product development and spend the majority of their time liaising with stakeholders and should not dictate how the team reaches a technical solution. This role is equivalent to the customer representative role in some other agile frameworks such as extreme programming (XP).

Communication is a core responsibility of the product owner. The ability to convey priorities and empathize with team members and stakeholders is vital to steer product development in the right direction. The product owner role bridges the communication gap between the team and its stakeholders, serving as a proxy for stakeholders to the team and as a team representative to the overall stakeholder community.Pichler, Roman. Agile Product Management with Scrum: Creating Products that Customers Love. Upper Saddle River, NJ: Addison-Wesley, 2010.

As the face of the team to the stakeholders, the following are some of the communication tasks of the product owner to the stakeholders:

Empathy is a key attribute for a product owner to have—the ability to put one's self in another's shoes. A product owner converses with different stakeholders, who have a variety of backgrounds, job roles, and objectives. A product owner must be able to see from these different points of view. To be effective, it is wise for a product owner to know the level of detail the audience needs. The development team needs thorough feedback and specifications so they can build a product up to expectation, while an executive sponsor may just need summaries of progress. Providing more information than necessary may lose stakeholder interest and waste time. A direct means of communication is the most preferred by seasoned agile product owners.Cohn, Mike. Succeeding

with Agile: Software Development Using Scrum. Upper Saddle River, NJ: Addison-Wesley, 2010.

A product owner's ability to communicate effectively is also enhanced by being skilled in techniques that identify stakeholder needs, negotiate priorities between stakeholder interests, and collaborate with developers to ensure effective implementation of requirements.

The development team is responsible for delivering potentially shippable product increments every sprint (the sprint goal).

The team has from three to nine members who carry out all tasks required to build the product increments (analysis, design, development, testing, technical writing, etc.). Although there will be several disciplines represented in the team, its members are referred to generically as developers. To avoid potential confusion that this only refers to programmers, some organizations call this a delivery team and its members just team members.

The development team in Scrum is self-organizing, even though there may be interaction with other roles outside the team, such as a project management office (PMO).

Scrum is facilitated by a scrum master, who is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The scrum master is not a traditional team lead or project manager but acts as a buffer between the team and any distracting influences. The scrum master ensures that the Scrum framework is followed. The scrum master helps to ensure the team follows the agreed processes in the Scrum framework, often facilitates key sessions, and encourages the team to improve. The role has also been referred to as a team facilitatorLeybourn, E. (2013). Directing the Agile Organisation: A Lean Approach to Business Management. London: IT Governance Publishing: 117–120. or servant-leader to reinforce these dual perspectives.

The core responsibilities of a scrum master include (but are not limited to):

One of the ways the scrum master role differs from a project manager is that the latter may have people management responsibilities and the scrum master does not. Scrum does not formally recognise the role of project manager, as traditional command and control tendencies would cause difficulties.

1. demonstrates the solution to key stakeholders who were not present at a sprint review;
2. defines and announces releases;
3. communicates team status;
4. organizes milestone reviews;

5. educates stakeholders in the development process;
6. negotiates priorities, scope, funding, and schedule;
7. ensures that the product backlog is visible, transparent and clear.

1. Helping the product owner maintain the product backlog in a way that ensures the needed work is well understood so the team can continually make forward progress
2. Helping the team to determine the definition of done for the product, with input from key stakeholders
3. Coaching the team, within the Scrum principles, in order to deliver high-quality features for its product
4. Promoting self-organization within the team
5. Helping the scrum team to avoid or remove impediments to its progress, whether internal or external to the team
6. Facilitating team events to ensure regular progress
7. Educating key stakeholders in the product on Scrum principles
8. Coaching the development team in self-organization and cross-functionality

3.5. Workflow

Scrum frameworkThe Scrum process

A sprint (or iteration) is the basic unit of development in Scrum. The sprint is a timeboxed effort; that is, it is restricted to a specific duration. The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common.

Each sprint starts with a sprint planning event that aims to define a sprint backlog, identify the work for the sprint, and make an estimated forecast for the sprint goal. Each sprint ends with a sprint review and sprint retrospective, that reviews progress to show to stakeholders and identify lessons and improvements for the next sprints.

Scrum emphasizes working product at the end of the sprint that is really done. In the case of software, this likely includes that the software has been fully integrated, tested and documented, and is potentially shippable.

At the beginning of a sprint, the scrum team holds a sprint planning event to:

A daily scrum in the computing room. This centralized location helps the team start on time.

Each day during a sprint, the team holds a daily scrum (or stand-up) with specific guidelines:

Any impediment (e.g., stumbling block, risk, issue, delayed dependency, assumption proved unfounded) identified in the daily scrum should be captured by the scrum master and displayed on the team's scrum board or on a shared risk board, with an agreed person designated to working toward a resolution (outside of the daily scrum). No detailed discussions should happen during the daily scrum.

At the end of a sprint, the team holds two events: the sprint review and the sprint retrospective.

At the sprint review, the team:

Guidelines for sprint reviews:

At the sprint retrospective, the team:

Guidelines for sprint retrospectives:

The following activities are commonly done, although not considered by all as a core part of Scrum:

Backlog refinement (once called backlog grooming) is the ongoing process of reviewing product backlog items and checking that they are appropriately prioritised and prepared in a way that makes them clear and executable for teams once they enter sprints via the sprint planning activity. Product backlog items may be broken into multiple smaller ones; acceptance criteria may be clarified; and dependencies, investigation, and preparatory work may be identified and agreed as technical spikes.

Although not originally a core Scrum practice, backlog refinement has been added to the Scrum Guide and adopted as a way of managing the quality of product backlog items entering a sprint, with a recommended investment of up to 10% of a team's sprint capacity.

The backlog can also include technical debt (also known as design debt or code debt). This is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer.

The product owner can cancel a sprint if necessary. The product owner may do so with input from the team, scrum master or management. For instance, management may wish the product owner to cancel a sprint if external circumstances negate the value of the sprint goal. If a sprint is abnormally terminated, the next step is to conduct a new sprint planning, where the reason for the termination is reviewed.

1. Mutually discuss and agree on the scope of work that is intended to be done during that sprint
2. Select product backlog items that can be completed in one sprint
3. Prepare a sprint backlog that includes the work needed to complete the selected product backlog items
4. The recommended duration is four hours for a two-week sprint (pro-rata for other sprint durations)
5. During the first half, the whole scrum team (development team, scrum master, and product owner) selects the product backlog items they believe could be completed in that sprint During the second half, the development team identifies the detailed work (tasks) required to complete those product backlog items; resulting in a confirmed sprint backlog As the detailed work is elaborated, some product backlog items may be split or put back into the product backlog if the team no longer believes they can complete the required work in a single sprint
6. Once the development team has prepared their sprint backlog, they forecast (usually by voting) which tasks will be delivered within the sprint.

1. All members of the development team come prepared. The daily scrum:
2. starts precisely on time even if some development team members are missing should happen at the same time and place every day is limited (timeboxed) to fifteen minutes
3. Anyone is welcome, though only development team members should contribute.
4. During the daily scrum, each team member typically answers three questions:
5. What did I complete yesterday that contributed to the team meeting our sprint goal? What do I plan to complete today to contribute to the team meeting our sprint goal? Do I see any impediment that could prevent me or the team from meeting our sprint goal?

1. reviews the work that was completed and the planned work that was not completed
2. presents the completed work to the stakeholders (a.k.a. the demo)
3. collaborates with the stakeholders on what to work on next

1. Incomplete work cannot be demonstrated.
2. The recommended duration is two hours for a two-week sprint (proportional for other sprint-durations).

1. Reflects on the past sprint
2. Identifies and agrees on continuous process improvement actions

1. Three main questions are asked in the sprint retrospective: What went well during the sprint? What did not go well? What could be improved for better productivity in the next sprint?
2. The recommended duration is one-and-a-half hours for a two-week sprint (proportional for other sprint duration(s))
3. This event is facilitated by the scrum master

3.6. Artifacts

The product backlog comprises an ordered list of product requirements that a scrum team maintains for a product. The format of product backlog items varies, common formats include user stories, use cases, or any other requirements format the team finds useful. These will define features, bug fixes, non-functional requirements, etc.—whatever must be done to successfully deliver a viable product. The product owner prioritizes product backlog items (PBIs) based on considerations such as risk, business value, dependencies, size, and date needed.

The product backlog is what will be delivered, ordered into the sequence in which it should be delivered. It is visible to everyone but may only be changed with the consent of the product owner, who is ultimately responsible for ordering product backlog items for the development team to choose.

The product backlog contains the product owner's assessment of business value and the development team's assessment of development effort, which are often, but not always, stated in story points using the rounded Fibonacci scale. These estimates help the product owner to gauge the timeline and may influence the ordering of product backlog items; for example, if two features have the same business value, the product owner may schedule earlier delivery of the one with the lower development effort (because the return on investment is higher) or the one with higher development effort (because it is more complex or riskier, and they want to retire that risk earlier).

The product backlog and the business value of each product backlog item is the responsibility of the product owner. The size (i.e. estimated complexity or effort) of each item is, however, determined by the development team, who contributes by sizing in story points or in estimated hours.

Every team should have a product owner, although in many instances they will work with more than one team. The product owner is responsible for maximizing the value of the product. The product owner gathers input and takes feedback from, and is lobbied by, many people, but ultimately makes the call on what gets built.

The product backlog:

Typically, the product owner and the scrum team come together and write down everything that must be prioritized, and this becomes content for the first sprint—which is a block of time meant for focused work on selected items that can be accommodated within a timeframe. The product backlog can evolve as new information surfaces about the product and about its customers, and so later sprints may address new work.

The following items typically comprise a product backlog: features, bugs, technical work, and knowledge acquisition. A feature is wanted, while a bug is unintended or unwanted (but may not be necessarily something defective). An example of technical work could be to run a virus check on all developers' workstations. An example of knowledge acquisition could be to research Wordpress plugin libraries and making a selection.

A product backlog, in its simplest form, is merely a list of items to work on. Having well-established rules about how work is added, removed and ordered helps the whole team make better decisions about how to change the product.

The product owner prioritizes product backlog items based on which are needed soonest. The team then chooses which items they can complete in the coming sprint. On the scrum board, the team moves items from the product backlog to the sprint backlog, which is the list of items they will build. Conceptually, it is ideal for the team to only select what they think they can accomplish from the top of the list, but it is not unusual to see in practice that teams are able to take lower-priority items from the list along with the top ones selected. This normally happens because there is time left within the sprint to accommodate more work. Items at the top of the backlog, the items to work on first, should be broken down into stories that are suitable for the development team to work on. The further down the backlog goes, the less refined the items should be. As Schwaber and Beedle put it "The lower the priority, the less detail until you can barely make out the backlog item."

As the team works through the backlog, it must be assumed that change happens outside their environment—the team can learn about new market opportunities to take advantage of, competitor threats that arise, and feedback from customers that can change the way the product was meant to work. All of these new ideas tend to trigger the team to adapt the backlog to incorporate new knowledge. This is part of the fundamental mindset of an agile team. The world changes, the backlog is never finished.

A Scrum task board

The sprint backlog is the list of work the development team must address during the next sprint. The list is derived by the scrum team progressively selecting product backlog items in priority order from the top of the product backlog until they feel they have enough work to fill the sprint. The development team should keep in mind its past performance assessing its capacity for the

new-sprint, and use this as a guideline of how much 'effort' they can complete.

The product backlog items may be broken down into tasks by the development team. Tasks on the sprint backlog are never assigned (or pushed) to team members by someone else; rather team members sign up for (or pull) tasks as needed according to the backlog priority and their own skills and capacity. This promotes self-organization of the development team and developer buy-in.

The sprint backlog is the property of the development team, and all included estimates are provided by the development team. Often an accompanying task board is used to see and change the state of the tasks of the current sprint, like to do, in progress and done.

Once a sprint backlog is committed, no additional work can be added to the sprint backlog except by the team. Once a sprint has been delivered, the product backlog is analyzed and reprioritized if necessary, and the next set of functionality is selected for the next sprint.

The increment (or potentially shippable increment, PSI) is the sum of all the product backlog items completed during a sprint, integrated with the work of all previous sprints. At the end of a sprint, the increment must be complete, according to the scrum team's definition of done (DoD), fully functioning, and in a usable condition regardless of whether the product owner decides to actually release it.

The following artifacts are commonly used, although not considered by all as a core part of Scrum:

A sample burn-down chart for a completed sprint, showing remaining effort at the end of each day.

The sprint burn-down chart is a publicly displayed chart showing remaining work in the sprint backlog. Updated every day, it gives a simple view of the sprint progress. It also provides quick visualizations for reference. The horizontal axis of the sprint burn-down chart shows the days in a sprint, while the vertical axis shows the amount of work remaining each day (typically representing the estimate of hours of work remaining).

During sprint planning, the ideal burndown chart is plotted. Then, during the sprint, each member picks up tasks from the sprint backlog and works on them. At the end of the day, they update the remaining hours for tasks to be completed. In such a way, the actual burndown chart is updated day by day.

It should not be confused with an earned value chart.

A sample burn-up chart for a release, showing scope completed each sprint

The release burn-up chart is a way for the team to provide visibility and track progress toward a release. Updated at the end of each sprint, it shows progress toward delivering a forecast scope. The horizontal axis of the release burn-up chart shows the sprints in a release, while the vertical axis shows the amount of work completed at the end of each sprint (typically representing cumulative story points of work completed). Progress is plotted as a line that grows up to meet a horizontal line that represents the forecast scope; often shown with a forecast, based on progress to date, that indicates how much scope might be completed by a given release date or how many sprints it will take to complete the given scope.

The release burn-up chart makes it easy to see how much work has been completed, how much work has been added or removed (if the horizontal scope line moves), and how much work is left to be done.

The exit-criteria to determine whether a product backlog item is complete. In many cases, the DoD requires that all regression tests be successful. The definition of done may vary from one scrum team to another but must be consistent within one team.Ken Schwaber, Agile Project Management with Scrum, p.55

The total effort a team is capable of in a sprint. The number is derived by evaluating the work (typically in user story points) completed in the last sprint. The collection of historical velocity data is a guideline for assisting the team in understanding how much work they can likely achieve in a future sprint.

A time-boxed period used to research a concept or create a simple prototype. Spikes can either be planned to take place in between sprints or, for larger teams, a spike might be accepted as one of many sprint delivery objectives. Spikes are often introduced before the delivery of large or complex product backlog items in order to secure budget, expand knowledge, or produce a proof of concept. The duration and objective(s) of a spike is agreed between product owner and development team before the start. Unlike sprint commitments, spikes may or may not deliver tangible, shippable, valuable functionality. For example, the objective of a spike might be to successfully reach a decision on a course of action. The spike is over when the time is up, not necessarily when the objective has been delivered.

Also called a drone spike, a tracer bullet is a spike with the current architecture, current technology set, current set of best practices that result in production quality code. It might just be a very narrow implementation of the functionality but is not throwaway code. It is of production quality, and the rest of the iterations can build on this code. The name has military origins as ammunition that makes the path of the bullet visible, allowing for corrections. Often these implementations are a 'quick shot' through all layers of an application, such as connecting a single form's input field to the back-end, to prove the layers connect as expected.

1. Captures requests to modify a product—including new features, replacing old features, removing features, and fixing issues
2. Ensures the development team has work that maximizes business benefit to the product owner

3.7. Limitations

Scrum works less well in the following circumstances:

From a business perspective, Scrum has many virtues, one of which is that it is designed to yield the best business solutions. However, the efficiency by which it does so in any given organization can vary widely and is largely dependent on the ability of the organization to adhere to the implementation guidelines. Every company has its own distinct organizational structure, culture, and set of business practices, and some are more naturally amenable to this methodology than others.

1. Teams whose members are geographically dispersed or part-time: In Scrum, developers should have close and ongoing interaction, ideally working together in the same space most of the time. While recent improvements in technology have reduced the impact of these barriers (e.g., being able to collaborate on a digital whiteboard), the Agile manifesto asserts that the best communication is face to face.
2. Teams whose members have very specialized skills: In Scrum, developers should be able to work on any task or pick up work that another developer has started. This can be managed by good Scrum leadership. While team members with very specific skills can and do contribute well, they should be encouraged to learn more about and collaborate with other disciplines.
3. Products with many external dependencies: In Scrum, dividing product development into short sprints requires careful planning; external dependencies, such as deliveries of software from other teams, can lead to delays and the failure of individual sprints.
4. Products that are mature or legacy or with regulated quality control: In Scrum, product increments should be fully developed and tested in a single sprint; products that need large amounts of regression testing or safety testing (e.g., medical devices or vehicle control) for each release are less suited to short sprints than to longer waterfall releases.

3.8. Tools for implementation

Like other agile methods, effective adoption of Scrum can be supported through a wide range of tools.

Many companies use universal tools, such as spreadsheets to build and maintain artifacts such as the sprint backlog. There are also open-source and proprietary software packages for Scrum—which are either dedicated to product development using the Scrum framework or support multiple product development approaches including Scrum.

Other organizations implement Scrum without software tools and maintain their artifacts in hard-copy forms such as paper, whiteboards, and sticky notes.

3.9. Scrum values

Scrum is a feedback-driven empirical approach which is, like all empirical process control, underpinned by the three pillars of transparency, inspection, and adaptation. All work within the Scrum framework should be visible to those responsible for the outcome: the process, the workflow, progress, etc. In order to make these things visible, scrum teams need to frequently inspect the product being developed and how well the team is working. With frequent inspection, the team can spot when their work deviates outside of acceptable limits and adapt their process or the product under development.TEMPLATE[Cite_book, url https://www.worldcat.org/oclc/951453155, title Scrum: an ideal framework for agile projects, last Morris, first David, publisher In Easy Steps, year 2017, isbn 9781840787313, location , pages 178–179, oclc 951453155]

These three pillars require trust and openness in the team, which the following five values of Scrum enable:

1. Commitment: Team members individually commit to achieving their team goals, each and every sprint.
2. Courage: Team members know they have the courage to work through conflict and challenges together so that they can do the right thing.
3. Focus: Team members focus exclusively on their team goals and the sprint backlog; there should be no work done other than through their backlog.
4. Openness: Team members and their stakeholders agree to be transparent about their work and any challenges they face.

5. Respect: Team members respect each other to be technically capable and to work with good intent.

3.10. Adaptations

The hybridization of Scrum with other software development methodologies is common as Scrum does not cover the whole product development lifecycle; therefore, organizations find the need to add in additional processes to create a more comprehensive implementation. For example, at the start of product development, organizations commonly add process guidance on the business case, requirements gathering and prioritization, initial high-level design, and budget and schedule forecasting.

Various authors and communities of people who use Scrum have also suggested more detailed techniques for how to apply or adapt Scrum to particular problems or organizations. Many refer to these methodological techniques as 'patterns' - by analogy with design patterns in architecture and software.TEMPLATE[cite_web, url https://sites.google.com/a/scrumorgpatterns.com/www/, title Scrum as Organizational Patterns, last Bjørnvig, first Gertrud, last2 Coplien, first2 Jim, date June 21, 2008, website , publisher Gertrude & Cope] Such patterns have extended Scrum outside of the software development domain into Manufacturing,TEMPLATE[cite_web, url http://agilebusinessmanagement.org/content/wikispeed-%E2%80%93-applying-agile-software-principles-and-practices-fast-automotive-development, title WIKISPEED – Applying Agile software principles and practices for fast automotive development, date December 3, 2013, website , publisher Agile Business Management Consortium, accessdate September 11, 2015] Finance and Human Resources.

Scrumban is a software production model based on Scrum and Kanban. Scrumban is especially suited for product maintenance with frequent and unexpected work items, such as production defects or programming errors. In such cases the time-limited sprints of the Scrum framework may be perceived to be of less benefit, although Scrum's daily events and other practices can still be applied, depending on the team and the situation at hand. Visualization of the work stages and limitations for simultaneous unfinished work and defects are familiar from the Kanban model. Using these methods, the team's workflow is directed in a way that allows for minimum completion time for each work item or programming error, and on the other hand ensures each team member is constantly employed.

To illustrate each stage of work, teams working in the same space often use post-it notes or a large whiteboard. In the case of decentralized teams, stage-illustration software such as Assembla, JIRA or Agilo.

The major differences between Scrum and Kanban is that in Scrum work is divided into sprints

that last a fixed amount of time, whereas in Kanban the flow of work is continuous. This is visible in work stage tables, which in Scrum are emptied after each sprint, whereas in Kanban all tasks are marked on the same table. Scrum focuses on teams with multifaceted know-how, whereas Kanban makes specialized, functional teams possible.

The scrum of scrums is a technique to operate Scrum at scale, for multiple teams working on the same product, allowing them to discuss progress on their interdependencies, focusing on how to coordinate delivering software, especially on areas of overlap and integration. Depending on the cadence (timing) of the scrum of scrums, the relevant daily scrum for each scrum team ends by designating one member as an ambassador to participate in the scrum of scrums with ambassadors from other teams. Depending on the context, the ambassadors may be technical contributors or each team's scrum master.

Rather than simply a progress update, the scrum of scrums should focus on how teams are collectively working to resolve, mitigate, or accept any risks, impediments, dependencies, and assumptions (RIDAs) that have been identified. The scrum of scrums tracks these RIDAs via a backlog of its own, such as a risk board (sometimes known as a ROAM board after the initials of resolved, owned, accepted, and mitigated),TEMPLATE[cite_web, url http://www.allaboutagile.com/risk-management-how-to-stop-risks-from-screwing-up-your-projects/, title Risk Management – How to Stop Risks from Screwing Up Your Projects!, publisher Kelly Waters] which typically leads to greater coordination and collaboration between teams.

This should run similar to a daily scrum, with each ambassador answering the following four questions:

As Jeff Sutherland commented,

Since I originally defined the Scrum of Scrums (Ken Schwaber was at IDX working with me), I can definitively say the Scrum of Scrums is not a "meta Scrum". The Scrum of Scrums as I have used it is responsible for delivering the working software of all teams to the Definition of Done at the end of the sprint, or for releases during the sprint. PatientKeeper delivered to production four times per Sprint. Ancestry.com delivers to production 220 times per two-week Sprint. Hubspot delivers live software 100-300 times a day. The Scrum of Scrums Master is held accountable for making this work. So the Scrum of Scrums is an operational delivery mechanism.

Large-scale Scrum (LeSS) is a product development framework that extends Scrum with scaling rules and guidelines without losing the original purposes of Scrum.

There are two levels to the framework: the first LeSS level is designed for up to 8 teams; the second level, known as "LeSS Huge", introduces additional scaling elements for development with up to hundreds of developers. "Scaling Scrum starts with understanding and being able to adopt

standard real one-team Scrum. Large-scale Scrum requires examining the purpose of single-team Scrum elements and figuring out how to reach the same purpose while staying within the constraints of the standard Scrum rules."

Bas Vodde and Craig Larman evolved the LeSS framework from their experiences working with large-scale product development, especially in the telecoms and finance industries. It evolved by taking Scrum and trying many different experiments to discover what works. In 2013, the experiments were solidified into the LeSS framework rules. The intention of LeSS is to "descale" organization complexity, dissolving unnecessary complex organizational solutions, and solving them in simpler ways. Less roles, less management, less organizational structures.

Many courses in higher education are adapting the scrum framework to give students in both IT and non-IT environments new tools and better insight for dealing with project management. The scrum framework helps students grasp the concepts of the coursework in new ways, while also fostering better teamwork, better communication, breaking tasks down into smaller parts, and it helps students become more self-driven and self-organized. Scrum has been used to help students become more self-aware and it encourages self-directed learning. Many of the aforementioned skills that are gained by classroom adapted scrum are sought after by many companies and help students become more prepared for the workplace, post-graduation. It is worth noting that classroom adopted scrum is not pure scrum, as it has to be adapted to fit a 16-week period and often many members of the team must play many roles—this does however help reinforce the Agile philosophy of always being ready for change, constant adaption, and being ready for new requirements and constraints.

1. What risks, impediments, dependencies, or assumptions has your team resolved since we last met?
2. What risks, impediments, dependencies, or assumptions will your team resolve before we meet again?
3. Are there any new risks, impediments, dependencies, or assumptions slowing your team down or getting in their way?
4. Are you about to introduce a new risk, impediment, dependency, or assumption that will get in another team's way?

4. Scrum (software development)#Large-scale scrum

4.1.

Scrum is an agile framework for managing knowledge work, with an emphasis on software development. It is designed for teams of three to nine members, who break their work into actions that can be completed within timeboxed iterations, called "sprints", no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute stand-up meetings, called daily scrums.

Approaches to coordinating the work of multiple scrum teams in larger organizations include Large-scale Scrum (LeSS), Scaled agile framework (SAFe), scrum of scrums, and Scrum@Scale, among others.

4.2. Key ideas

Scrum is a lightweight, iterative and incremental framework for managing product development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved.

A key principle of Scrum is the dual recognition that customers will change their minds about what they want or need (often called requirements volatilityJ. Henry and S. Henry. Quantitative assessment of the software maintenance process and requirements volatility. In Proc. of the ACM Conference on Computer Science, pages 346–351, 1993.) and that there will be unpredictable challenges—for which a predictive or planned approach is not suited. As such, Scrum adopts an evidence-based empirical approach—accepting that the problem cannot be fully understood or defined up front, and instead focusing on how to maximize the team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in market conditions.

Many of the terms used in Scrum (e.g., scrum master) are typically written with leading capitals (e.g., Scrum Master) or as conjoint words written in camel case (e.g., ScrumMaster). To maintain an encyclopedic tone, however, this article uses normal sentence case for these terms—unless they are recognized marks (such as Certified Scrum Master). This is occasionally seen written in all-capitals, as SCRUM. The word is not an acronym, so this is not correct; however, it likely arose due to an early paper by Ken Schwaber which capitalized SCRUM in its title.

While the trademark on the term Scrum itself has been allowed to lapse, so that it is deemed as owned by the wider community rather than an individual, the leading capital is retained—except when used with other words (as in daily scrum or scrum team).

4.3. History

Hirotaka Takeuchi and Ikujiro Nonaka introduced the term scrum in the context of product development in their 1986 Harvard Business Review article, "The New New Product Development Game". Takeuchi and Nonaka later argued in The Knowledge Creating Company that it is a form of "organizational knowledge creation, especially good at bringing about innovation continuously, incrementally and spirally".

The authors described a new approach to commercial product development that would increase speed and flexibility, based on case studies from manufacturing firms in the automotive, photocopier and printer industries. They called this the holistic or rugby approach, as the whole process is performed by one cross-functional team across multiple overlapping phases, where the team "tries to go the distance as a unit, passing the ball back and forth". (In rugby football, a scrum is used to restart play, as the forwards of each team interlock with their heads down and attempt to gain possession of the ball.TEMPLATE[Oxford Dictionaries, Scrum])

In the early 1990s, Ken Schwaber used what would become Scrum at his company, Advanced Development Methods; while Jeff Sutherland, John Scumniotales and Jeff McKenna, developed a similar approach at Easel Corporation, referring to it using the single word Scrum.

In 1995, Sutherland and Schwaber jointly presented a paper describing the Scrum framework at the Business Object Design and Implementation Workshop held as part of Object-Oriented Programming, Systems, Languages & Applications '95 (OOPSLA '95) in Austin, Texas. Over the following years, Schwaber and Sutherland collaborated to combine this material—with their experience and evolving good practice—to develop what became known as Scrum.

In 2001, Schwaber worked with Mike Beedle to describe the method in the book, Agile Software Development with Scrum. Scrum's approach to planning and managing product development involves bringing decision-making authority to the level of operation properties and certainties.

In 2002, Schwaber with others founded the Scrum Alliance
and set up the Certified Scrum accreditation series. Schwaber left the Scrum Alliance in late 2009 and founded Scrum.org which oversees the parallel Professional Scrum accreditation series.

Since 2009, there is a public document called The Scrum Guide that defines a sort of official

version of Scrum and is occasionally revised. In 2018 it was expanded upon with the publication of The Kanban Guide for Scrum Teams.

4.4. Roles

There are three core roles in the Scrum framework. These are ideally co-located to deliver potentially shippable product increments every sprint. Together these three roles form the scrum team. While many organizations have other roles involved with defining and delivering the product, Scrum defines only these three.

The product owner represents the product's stakeholders and the voice of the customer, is responsible for the backlog and accountable for maximising the value that the team delivers. The product owner defines the product in customer-centric terms (typically user stories), adds them to the product backlog, and prioritizes them based on importance and dependencies. A scrum team should have only one product owner (although a product owner could support more than one team). This role should not be combined with that of the scrum master. The product owner should focus on the business side of product development and spend the majority of their time liaising with stakeholders and should not dictate how the team reaches a technical solution. This role is equivalent to the customer representative role in some other agile frameworks such as extreme programming (XP).

Communication is a core responsibility of the product owner. The ability to convey priorities and empathize with team members and stakeholders is vital to steer product development in the right direction. The product owner role bridges the communication gap between the team and its stakeholders, serving as a proxy for stakeholders to the team and as a team representative to the overall stakeholder community.Pichler, Roman. Agile Product Management with Scrum: Creating Products that Customers Love. Upper Saddle River, NJ: Addison-Wesley, 2010.

As the face of the team to the stakeholders, the following are some of the communication tasks of the product owner to the stakeholders:

Empathy is a key attribute for a product owner to have—the ability to put one's self in another's shoes. A product owner converses with different stakeholders, who have a variety of backgrounds, job roles, and objectives. A product owner must be able to see from these different points of view. To be effective, it is wise for a product owner to know the level of detail the audience needs. The development team needs thorough feedback and specifications so they can build a product up to expectation, while an executive sponsor may just need summaries of progress. Providing more information than necessary may lose stakeholder interest and waste time. A direct means of communication is the most preferred by seasoned agile product owners.Cohn, Mike. Succeeding

with Agile: Software Development Using Scrum. Upper Saddle River, NJ: Addison-Wesley, 2010.

A product owner's ability to communicate effectively is also enhanced by being skilled in techniques that identify stakeholder needs, negotiate priorities between stakeholder interests, and collaborate with developers to ensure effective implementation of requirements.

The development team is responsible for delivering potentially shippable product increments every sprint (the sprint goal).

The team has from three to nine members who carry out all tasks required to build the product increments (analysis, design, development, testing, technical writing, etc.). Although there will be several disciplines represented in the team, its members are referred to generically as developers. To avoid potential confusion that this only refers to programmers, some organizations call this a delivery team and its members just team members.

The development team in Scrum is self-organizing, even though there may be interaction with other roles outside the team, such as a project management office (PMO).

Scrum is facilitated by a scrum master, who is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The scrum master is not a traditional team lead or project manager but acts as a buffer between the team and any distracting influences. The scrum master ensures that the Scrum framework is followed. The scrum master helps to ensure the team follows the agreed processes in the Scrum framework, often facilitates key sessions, and encourages the team to improve. The role has also been referred to as a team facilitatorLeybourn, E. (2013). Directing the Agile Organisation: A Lean Approach to Business Management. London: IT Governance Publishing: 117–120. or servant-leader to reinforce these dual perspectives.

The core responsibilities of a scrum master include (but are not limited to):

One of the ways the scrum master role differs from a project manager is that the latter may have people management responsibilities and the scrum master does not. Scrum does not formally recognise the role of project manager, as traditional command and control tendencies would cause difficulties.

1. demonstrates the solution to key stakeholders who were not present at a sprint review;
2. defines and announces releases;
3. communicates team status;
4. organizes milestone reviews;

5. educates stakeholders in the development process;
6. negotiates priorities, scope, funding, and schedule;
7. ensures that the product backlog is visible, transparent and clear.

1. Helping the product owner maintain the product backlog in a way that ensures the needed work is well understood so the team can continually make forward progress
2. Helping the team to determine the definition of done for the product, with input from key stakeholders
3. Coaching the team, within the Scrum principles, in order to deliver high-quality features for its product
4. Promoting self-organization within the team
5. Helping the scrum team to avoid or remove impediments to its progress, whether internal or external to the team
6. Facilitating team events to ensure regular progress
7. Educating key stakeholders in the product on Scrum principles
8. Coaching the development team in self-organization and cross-functionality

4.5. Workflow

Scrum frameworkThe Scrum process

A sprint (or iteration) is the basic unit of development in Scrum. The sprint is a timeboxed effort; that is, it is restricted to a specific duration. The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common.

Each sprint starts with a sprint planning event that aims to define a sprint backlog, identify the work for the sprint, and make an estimated forecast for the sprint goal. Each sprint ends with a sprint review and sprint retrospective, that reviews progress to show to stakeholders and identify lessons and improvements for the next sprints.

Scrum emphasizes working product at the end of the sprint that is really done. In the case of software, this likely includes that the software has been fully integrated, tested and documented, and is potentially shippable.

At the beginning of a sprint, the scrum team holds a sprint planning event to:

A daily scrum in the computing room. This centralized location helps the team start on time.

Each day during a sprint, the team holds a daily scrum (or stand-up) with specific guidelines:

Any impediment (e.g., stumbling block, risk, issue, delayed dependency, assumption proved unfounded) identified in the daily scrum should be captured by the scrum master and displayed on the team's scrum board or on a shared risk board, with an agreed person designated to working toward a resolution (outside of the daily scrum). No detailed discussions should happen during the daily scrum.

At the end of a sprint, the team holds two events: the sprint review and the sprint retrospective.

At the sprint review, the team:

Guidelines for sprint reviews:

At the sprint retrospective, the team:

Guidelines for sprint retrospectives:

The following activities are commonly done, although not considered by all as a core part of Scrum:

Backlog refinement (once called backlog grooming) is the ongoing process of reviewing product backlog items and checking that they are appropriately prioritised and prepared in a way that makes them clear and executable for teams once they enter sprints via the sprint planning activity. Product backlog items may be broken into multiple smaller ones; acceptance criteria may be clarified; and dependencies, investigation, and preparatory work may be identified and agreed as technical spikes.

Although not originally a core Scrum practice, backlog refinement has been added to the Scrum Guide and adopted as a way of managing the quality of product backlog items entering a sprint, with a recommended investment of up to 10% of a team's sprint capacity.

The backlog can also include technical debt (also known as design debt or code debt). This is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer.

The product owner can cancel a sprint if necessary. The product owner may do so with input from the team, scrum master or management. For instance, management may wish the product owner to cancel a sprint if external circumstances negate the value of the sprint goal. If a sprint is abnormally terminated, the next step is to conduct a new sprint planning, where the reason for the termination is reviewed.

1. Mutually discuss and agree on the scope of work that is intended to be done during that sprint
2. Select product backlog items that can be completed in one sprint
3. Prepare a sprint backlog that includes the work needed to complete the selected product backlog items
4. The recommended duration is four hours for a two-week sprint (pro-rata for other sprint durations)
5. During the first half, the whole scrum team (development team, scrum master, and product owner) selects the product backlog items they believe could be completed in that sprint During the second half, the development team identifies the detailed work (tasks) required to complete those product backlog items; resulting in a confirmed sprint backlog As the detailed work is elaborated, some product backlog items may be split or put back into the product backlog if the team no longer believes they can complete the required work in a single sprint
6. Once the development team has prepared their sprint backlog, they forecast (usually by voting) which tasks will be delivered within the sprint.

1. All members of the development team come prepared. The daily scrum:
2. starts precisely on time even if some development team members are missing should happen at the same time and place every day is limited (timeboxed) to fifteen minutes
3. Anyone is welcome, though only development team members should contribute.
4. During the daily scrum, each team member typically answers three questions:
5. What did I complete yesterday that contributed to the team meeting our sprint goal? What do I plan to complete today to contribute to the team meeting our sprint goal? Do I see any impediment that could prevent me or the team from meeting our sprint goal?

1. reviews the work that was completed and the planned work that was not completed
2. presents the completed work to the stakeholders (a.k.a. the demo)
3. collaborates with the stakeholders on what to work on next

1. Incomplete work cannot be demonstrated.
2. The recommended duration is two hours for a two-week sprint (proportional for other sprint-durations).

1. Reflects on the past sprint
2. Identifies and agrees on continuous process improvement actions

1. Three main questions are asked in the sprint retrospective: What went well during the sprint? What did not go well? What could be improved for better productivity in the next sprint?
2. The recommended duration is one-and-a-half hours for a two-week sprint (proportional for other sprint duration(s))
3. This event is facilitated by the scrum master

4.6. Artifacts

The product backlog comprises an ordered list of product requirements that a scrum team maintains for a product. The format of product backlog items varies, common formats include user stories, use cases, or any other requirements format the team finds useful. These will define features, bug fixes, non-functional requirements, etc.—whatever must be done to successfully deliver a viable product. The product owner prioritizes product backlog items (PBIs) based on considerations such as risk, business value, dependencies, size, and date needed.

The product backlog is what will be delivered, ordered into the sequence in which it should be delivered. It is visible to everyone but may only be changed with the consent of the product owner, who is ultimately responsible for ordering product backlog items for the development team to choose.

The product backlog contains the product owner's assessment of business value and the development team's assessment of development effort, which are often, but not always, stated in story points using the rounded Fibonacci scale. These estimates help the product owner to gauge the timeline and may influence the ordering of product backlog items; for example, if two features have the same business value, the product owner may schedule earlier delivery of the one with the lower development effort (because the return on investment is higher) or the one with higher development effort (because it is more complex or riskier, and they want to retire that risk earlier).

The product backlog and the business value of each product backlog item is the responsibility of the product owner. The size (i.e. estimated complexity or effort) of each item is, however, determined by the development team, who contributes by sizing in story points or in estimated hours.

Every team should have a product owner, although in many instances they will work with more than one team. The product owner is responsible for maximizing the value of the product. The product owner gathers input and takes feedback from, and is lobbied by, many people, but ultimately makes the call on what gets built.

The product backlog:

Typically, the product owner and the scrum team come together and write down everything that must be prioritized, and this becomes content for the first sprint—which is a block of time meant for focused work on selected items that can be accommodated within a timeframe. The product backlog can evolve as new information surfaces about the product and about its customers, and so later sprints may address new work.

The following items typically comprise a product backlog: features, bugs, technical work, and knowledge acquisition. A feature is wanted, while a bug is unintended or unwanted (but may not be necessarily something defective). An example of technical work could be to run a virus check on all developers' workstations. An example of knowledge acquisition could be to research Wordpress plugin libraries and making a selection.

A product backlog, in its simplest form, is merely a list of items to work on. Having well-established rules about how work is added, removed and ordered helps the whole team make better decisions about how to change the product.

The product owner prioritizes product backlog items based on which are needed soonest. The team then chooses which items they can complete in the coming sprint. On the scrum board, the team moves items from the product backlog to the sprint backlog, which is the list of items they will build. Conceptually, it is ideal for the team to only select what they think they can accomplish from the top of the list, but it is not unusual to see in practice that teams are able to take lower-priority items from the list along with the top ones selected. This normally happens because there is time left within the sprint to accommodate more work. Items at the top of the backlog, the items to work on first, should be broken down into stories that are suitable for the development team to work on. The further down the backlog goes, the less refined the items should be. As Schwaber and Beedle put it "The lower the priority, the less detail until you can barely make out the backlog item."

As the team works through the backlog, it must be assumed that change happens outside their environment—the team can learn about new market opportunities to take advantage of, competitor threats that arise, and feedback from customers that can change the way the product was meant to work. All of these new ideas tend to trigger the team to adapt the backlog to incorporate new knowledge. This is part of the fundamental mindset of an agile team. The world changes, the backlog is never finished.

A Scrum task board

The sprint backlog is the list of work the development team must address during the next sprint. The list is derived by the scrum team progressively selecting product backlog items in priority order from the top of the product backlog until they feel they have enough work to fill the sprint. The development team should keep in mind its past performance assessing its capacity for the

new-sprint, and use this as a guideline of how much 'effort' they can complete.

The product backlog items may be broken down into tasks by the development team. Tasks on the sprint backlog are never assigned (or pushed) to team members by someone else; rather team members sign up for (or pull) tasks as needed according to the backlog priority and their own skills and capacity. This promotes self-organization of the development team and developer buy-in.

The sprint backlog is the property of the development team, and all included estimates are provided by the development team. Often an accompanying task board is used to see and change the state of the tasks of the current sprint, like to do, in progress and done.

Once a sprint backlog is committed, no additional work can be added to the sprint backlog except by the team. Once a sprint has been delivered, the product backlog is analyzed and reprioritized if necessary, and the next set of functionality is selected for the next sprint.

The increment (or potentially shippable increment, PSI) is the sum of all the product backlog items completed during a sprint, integrated with the work of all previous sprints. At the end of a sprint, the increment must be complete, according to the scrum team's definition of done (DoD), fully functioning, and in a usable condition regardless of whether the product owner decides to actually release it.

The following artifacts are commonly used, although not considered by all as a core part of Scrum:

A sample burn-down chart for a completed sprint, showing remaining effort at the end of each day.

The sprint burn-down chart is a publicly displayed chart showing remaining work in the sprint backlog. Updated every day, it gives a simple view of the sprint progress. It also provides quick visualizations for reference. The horizontal axis of the sprint burn-down chart shows the days in a sprint, while the vertical axis shows the amount of work remaining each day (typically representing the estimate of hours of work remaining).

During sprint planning, the ideal burndown chart is plotted. Then, during the sprint, each member picks up tasks from the sprint backlog and works on them. At the end of the day, they update the remaining hours for tasks to be completed. In such a way, the actual burndown chart is updated day by day.

It should not be confused with an earned value chart.

A sample burn-up chart for a release, showing scope completed each sprint

The release burn-up chart is a way for the team to provide visibility and track progress toward a release. Updated at the end of each sprint, it shows progress toward delivering a forecast scope. The horizontal axis of the release burn-up chart shows the sprints in a release, while the vertical axis shows the amount of work completed at the end of each sprint (typically representing cumulative story points of work completed). Progress is plotted as a line that grows up to meet a horizontal line that represents the forecast scope; often shown with a forecast, based on progress to date, that indicates how much scope might be completed by a given release date or how many sprints it will take to complete the given scope.

The release burn-up chart makes it easy to see how much work has been completed, how much work has been added or removed (if the horizontal scope line moves), and how much work is left to be done.

The exit-criteria to determine whether a product backlog item is complete. In many cases, the DoD requires that all regression tests be successful. The definition of done may vary from one scrum team to another but must be consistent within one team.Ken Schwaber, Agile Project Management with Scrum, p.55

The total effort a team is capable of in a sprint. The number is derived by evaluating the work (typically in user story points) completed in the last sprint. The collection of historical velocity data is a guideline for assisting the team in understanding how much work they can likely achieve in a future sprint.

A time-boxed period used to research a concept or create a simple prototype. Spikes can either be planned to take place in between sprints or, for larger teams, a spike might be accepted as one of many sprint delivery objectives. Spikes are often introduced before the delivery of large or complex product backlog items in order to secure budget, expand knowledge, or produce a proof of concept. The duration and objective(s) of a spike is agreed between product owner and development team before the start. Unlike sprint commitments, spikes may or may not deliver tangible, shippable, valuable functionality. For example, the objective of a spike might be to successfully reach a decision on a course of action. The spike is over when the time is up, not necessarily when the objective has been delivered.

Also called a drone spike, a tracer bullet is a spike with the current architecture, current technology set, current set of best practices that result in production quality code. It might just be a very narrow implementation of the functionality but is not throwaway code. It is of production quality, and the rest of the iterations can build on this code. The name has military origins as ammunition that makes the path of the bullet visible, allowing for corrections. Often these implementations are a 'quick shot' through all layers of an application, such as connecting a single form's input field to the back-end, to prove the layers connect as expected.

1. Captures requests to modify a product—including new features, replacing old features, removing features, and fixing issues
2. Ensures the development team has work that maximizes business benefit to the product owner

4.7. Limitations

Scrum works less well in the following circumstances:

From a business perspective, Scrum has many virtues, one of which is that it is designed to yield the best business solutions. However, the efficiency by which it does so in any given organization can vary widely and is largely dependent on the ability of the organization to adhere to the implementation guidelines. Every company has its own distinct organizational structure, culture, and set of business practices, and some are more naturally amenable to this methodology than others.

1. Teams whose members are geographically dispersed or part-time: In Scrum, developers should have close and ongoing interaction, ideally working together in the same space most of the time. While recent improvements in technology have reduced the impact of these barriers (e.g., being able to collaborate on a digital whiteboard), the Agile manifesto asserts that the best communication is face to face.
2. Teams whose members have very specialized skills: In Scrum, developers should be able to work on any task or pick up work that another developer has started. This can be managed by good Scrum leadership. While team members with very specific skills can and do contribute well, they should be encouraged to learn more about and collaborate with other disciplines.
3. Products with many external dependencies: In Scrum, dividing product development into short sprints requires careful planning; external dependencies, such as deliveries of software from other teams, can lead to delays and the failure of individual sprints.
4. Products that are mature or legacy or with regulated quality control: In Scrum, product increments should be fully developed and tested in a single sprint; products that need large amounts of regression testing or safety testing (e.g., medical devices or vehicle control) for each release are less suited to short sprints than to longer waterfall releases.

4.8. Tools for implementation

Like other agile methods, effective adoption of Scrum can be supported through a wide range of tools.

Many companies use universal tools, such as spreadsheets to build and maintain artifacts such as the sprint backlog. There are also open-source and proprietary software packages for Scrum—which are either dedicated to product development using the Scrum framework or support multiple product development approaches including Scrum.

Other organizations implement Scrum without software tools and maintain their artifacts in hard-copy forms such as paper, whiteboards, and sticky notes.

4.9. Scrum values

Scrum is a feedback-driven empirical approach which is, like all empirical process control, underpinned by the three pillars of transparency, inspection, and adaptation. All work within the Scrum framework should be visible to those responsible for the outcome: the process, the workflow, progress, etc. In order to make these things visible, scrum teams need to frequently inspect the product being developed and how well the team is working. With frequent inspection, the team can spot when their work deviates outside of acceptable limits and adapt their process or the product under development.TEMPLATE[Cite_book, url https://www.worldcat.org/oclc/951453155, title Scrum: an ideal framework for agile projects, last Morris, first David, publisher In Easy Steps, year 2017, isbn 9781840787313, location , pages 178–179, oclc 951453155]

These three pillars require trust and openness in the team, which the following five values of Scrum enable:

1. Commitment: Team members individually commit to achieving their team goals, each and every sprint.
2. Courage: Team members know they have the courage to work through conflict and challenges together so that they can do the right thing.
3. Focus: Team members focus exclusively on their team goals and the sprint backlog; there should be no work done other than through their backlog.
4. Openness: Team members and their stakeholders agree to be transparent about their work and any challenges they face.

5. Respect: Team members respect each other to be technically capable and to work with good intent.

4.10. Adaptations

The hybridization of Scrum with other software development methodologies is common as Scrum does not cover the whole product development lifecycle; therefore, organizations find the need to add in additional processes to create a more comprehensive implementation. For example, at the start of product development, organizations commonly add process guidance on the business case, requirements gathering and prioritization, initial high-level design, and budget and schedule forecasting.

Various authors and communities of people who use Scrum have also suggested more detailed techniques for how to apply or adapt Scrum to particular problems or organizations. Many refer to these methodological techniques as 'patterns' - by analogy with design patterns in architecture and software.TEMPLATE[cite_web, url https://sites.google.com/a/scrumorgpatterns.com/www/, title Scrum as Organizational Patterns, last Bjørnvig, first Gertrud, last2 Coplien, first2 Jim, date June 21, 2008, website , publisher Gertrude & Cope] Such patterns have extended Scrum outside of the software development domain into Manufacturing,TEMPLATE[cite_web, url http://agilebusinessmanagement.org/content/wikispeed-%E2%80%93-applying-agile-software-principles-and-practices-fast-automotive-development, title WIKISPEED – Applying Agile software principles and practices for fast automotive development, date December 3, 2013, website , publisher Agile Business Management Consortium, accessdate September 11, 2015] Finance and Human Resources.

Scrumban is a software production model based on Scrum and Kanban. Scrumban is especially suited for product maintenance with frequent and unexpected work items, such as production defects or programming errors. In such cases the time-limited sprints of the Scrum framework may be perceived to be of less benefit, although Scrum's daily events and other practices can still be applied, depending on the team and the situation at hand. Visualization of the work stages and limitations for simultaneous unfinished work and defects are familiar from the Kanban model. Using these methods, the team's workflow is directed in a way that allows for minimum completion time for each work item or programming error, and on the other hand ensures each team member is constantly employed.

To illustrate each stage of work, teams working in the same space often use post-it notes or a large whiteboard. In the case of decentralized teams, stage-illustration software such as Assembla, JIRA or Agilo.

The major differences between Scrum and Kanban is that in Scrum work is divided into sprints

that last a fixed amount of time, whereas in Kanban the flow of work is continuous. This is visible in work stage tables, which in Scrum are emptied after each sprint, whereas in Kanban all tasks are marked on the same table. Scrum focuses on teams with multifaceted know-how, whereas Kanban makes specialized, functional teams possible.

The scrum of scrums is a technique to operate Scrum at scale, for multiple teams working on the same product, allowing them to discuss progress on their interdependencies, focusing on how to coordinate delivering software, especially on areas of overlap and integration. Depending on the cadence (timing) of the scrum of scrums, the relevant daily scrum for each scrum team ends by designating one member as an ambassador to participate in the scrum of scrums with ambassadors from other teams. Depending on the context, the ambassadors may be technical contributors or each team's scrum master.

Rather than simply a progress update, the scrum of scrums should focus on how teams are collectively working to resolve, mitigate, or accept any risks, impediments, dependencies, and assumptions (RIDAs) that have been identified. The scrum of scrums tracks these RIDAs via a backlog of its own, such as a risk board (sometimes known as a ROAM board after the initials of resolved, owned, accepted, and mitigated),TEMPLATE[cite_web, url http://www.allaboutagile.com/risk-management-how-to-stop-risks-from-screwing-up-your-projects/, title Risk Management – How to Stop Risks from Screwing Up Your Projects!, publisher Kelly Waters] which typically leads to greater coordination and collaboration between teams.

This should run similar to a daily scrum, with each ambassador answering the following four questions:

As Jeff Sutherland commented,

Since I originally defined the Scrum of Scrums (Ken Schwaber was at IDX working with me), I can definitively say the Scrum of Scrums is not a "meta Scrum". The Scrum of Scrums as I have used it is responsible for delivering the working software of all teams to the Definition of Done at the end of the sprint, or for releases during the sprint. PatientKeeper delivered to production four times per Sprint. Ancestry.com delivers to production 220 times per two-week Sprint. Hubspot delivers live software 100-300 times a day. The Scrum of Scrums Master is held accountable for making this work. So the Scrum of Scrums is an operational delivery mechanism.

Large-scale Scrum (LeSS) is a product development framework that extends Scrum with scaling rules and guidelines without losing the original purposes of Scrum.

There are two levels to the framework: the first LeSS level is designed for up to 8 teams; the second level, known as "LeSS Huge", introduces additional scaling elements for development with up to hundreds of developers. "Scaling Scrum starts with understanding and being able to adopt

standard real one-team Scrum. Large-scale Scrum requires examining the purpose of single-team Scrum elements and figuring out how to reach the same purpose while staying within the constraints of the standard Scrum rules."

Bas Vodde and Craig Larman evolved the LeSS framework from their experiences working with large-scale product development, especially in the telecoms and finance industries. It evolved by taking Scrum and trying many different experiments to discover what works. In 2013, the experiments were solidified into the LeSS framework rules. The intention of LeSS is to "descale" organization complexity, dissolving unnecessary complex organizational solutions, and solving them in simpler ways. Less roles, less management, less organizational structures.

Many courses in higher education are adapting the scrum framework to give students in both IT and non-IT environments new tools and better insight for dealing with project management. The scrum framework helps students grasp the concepts of the coursework in new ways, while also fostering better teamwork, better communication, breaking tasks down into smaller parts, and it helps students become more self-driven and self-organized. Scrum has been used to help students become more self-aware and it encourages self-directed learning. Many of the aforementioned skills that are gained by classroom adapted scrum are sought after by many companies and help students become more prepared for the workplace, post-graduation. It is worth noting that classroom adopted scrum is not pure scrum, as it has to be adapted to fit a 16-week period and often many members of the team must play many roles—this does however help reinforce the Agile philosophy of always being ready for change, constant adaption, and being ready for new requirements and constraints.

1. What risks, impediments, dependencies, or assumptions has your team resolved since we last met?
2. What risks, impediments, dependencies, or assumptions will your team resolve before we meet again?
3. Are there any new risks, impediments, dependencies, or assumptions slowing your team down or getting in their way?
4. Are you about to introduce a new risk, impediment, dependency, or assumption that will get in another team's way?