Scrum Reference Card

by Michael James and Luke Walter

About Scrum

An Empirical Framework

Scrum is a framework for product development using cross-functional teams. It emphasizes empirical (real world) feedback and team self management.

Scrum provides a structure of roles, events, rules, and artifacts. In this framework, teams must create and adapt their own ways of working.

Scrum uses fixed-length iterations, called Sprints. Sprints can be no longer than a month, and preferably a week or two. Teams try to develop a usable, potentially shippable, properly tested product increment every Sprint. An increment that is shippable to its end user – not just handed off internally – closes the Sprint's feedback loop.

An Alternative to Waterfall

Scrum's incremental, iterative approach trades the traditional phases of "waterfall" development for the ability to deliver small features first, then to revise plans based on ongoing discovery.

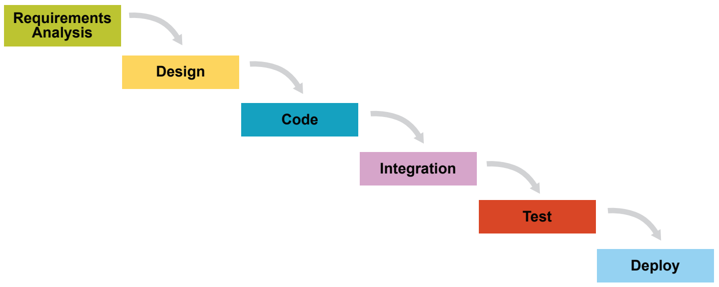


Figure 1: *Traditional "waterfall" development depends on a perfect understanding of the product requirements at the outset and minimal errors executing each phase.*

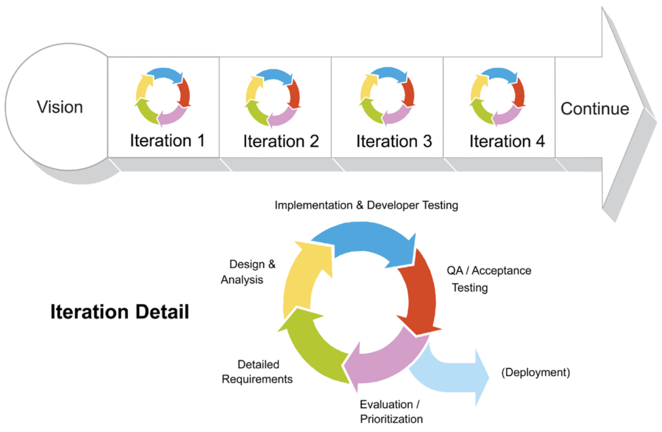


Figure 2: *Scrum blends all development activities into each iteration, adapting to discovered realities at fixed intervals.*

Scrum is for complex work involving knowledge creation and collaboration such as software development. Its use has also spread to the development of products such as semiconductors, mortgages, and wheelchairs.

Doing Scrum, or Pretending to Do Scrum?

Scrum's reality checks expose harmful constraints in individuals, teams, and organizations. Many people claiming to do Scrum modify the parts that require breaking through organizational impediments and end up robbing themselves of most of the benefits.

Scrum Team

Team

- Sometimes called "developers" but intended to be cross-functional, e.g. including members with testing skills, designers, domain experts, data scientists, business analysts, etc.
- Self-organizing / self-managing, without externally assigned roles.
- Plans one Sprint at a time with the Product Owner, and other teams if applicable.
- Owns *how* to develop the increment.
- Owns both internal collaboration and external collaboration (e.g. working with other teams, clarifying details with end users, etc.).
- More successful when located in one team room, particularly for the first few Sprints.
- More successful with long-term, full-time membership. Scrum moves work to a flexible learning team and avoids moving people or splitting them between teams.
- Around six members, give or take a few.
- No appointed lead. On a healthy team, leadership emerges and shifts naturally.

Product Owner

- Maximizes the value of the development effort by declaring vision and priorities.
- Only one per product, even with multiple teams.
- Constantly re-prioritizes the Product Backlog, adjusting any long-term expectations such as release plans.
- Final arbiter of requirements questions.
- Decides whether to release the product.
- Decides whether to continue developing the product.

Scrum Master

- Works with the organization to make Scrum possible.
- Ensures Scrum is understood and can be enacted.
- Creates an environment conducive to team self-organization.
- Shields the team from external interference and distractions to keep it in group flow (a.k.a. the *zone*).
- Promotes improved engineering practices.
- Has no management authority over the team.
- Helps resolve impediments.
- Serves the team, the Product Owner, and the organization.
- See also https://scrummasterchecklist.org

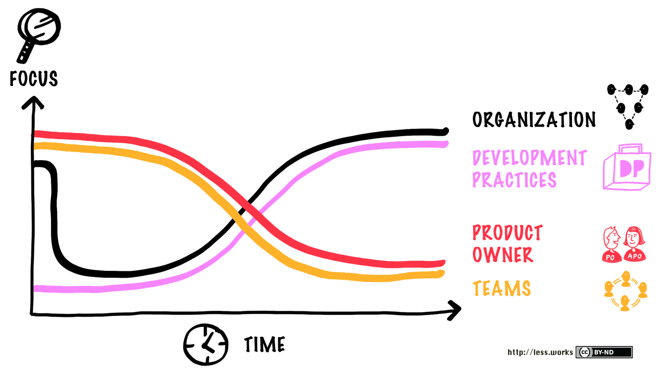


Figure 3: *The Scrum Master's focus shifts over time.*

Scrum Events

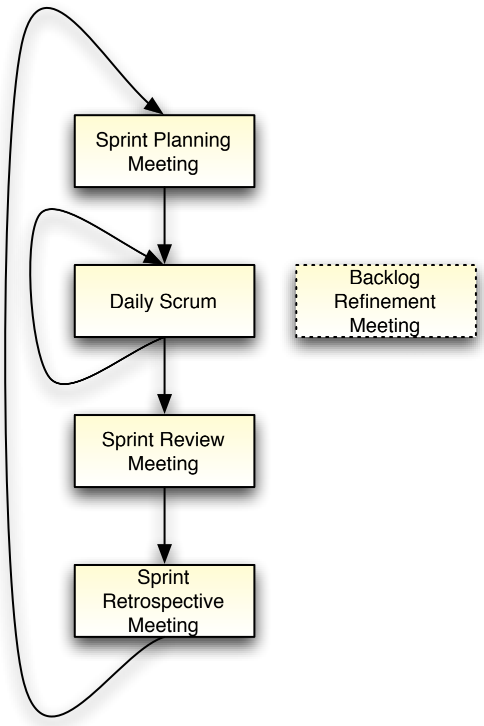


Figure 4: *Scrum flow.*

Sprint Planning

At the beginning of each Sprint, the Product Owner and team(s) plan which Product Backlog Items they'll try to convert to working product during the Sprint. The Product Owner declares which items are the most important to the business. The development team is responsible for selecting the amount of work they feel they can implement without accruing technical debt. The team *selects* work from the Product Backlog for the Sprint Backlog.

Declaring a Sprint Goal can increase focus on the big picture.

Software development has inherent uncertainty. Teams can really only *guess* how much work to select each Sprint, while learning from previous Sprints. Traditional habits of trying to plan by hourly capacity can make the team pretend to be precise and reduce ownership of the plan. While relative estimation (e.g. "story points") may help, it's often led to the same problem: the over-certainty that numbers imply, an example of what Luke Walter calls *left brain poisoning*. Some teams produce better Sprint plans by ditching quantitative practices.

Until a team has learned how to complete a shippable product increment each Sprint, it should reduce the feature scope that it plans, while increasing emphasis on testing, integration, and source code understandability. Failure to change old habits leads to technical debt and eventual design death, as shown in Figure 16.

A portion of Sprint Planning may be needed to further refine the selected items.

In the last part of Sprint Planning, the team forecasts how it will accomplish the work. For example, they may break the selected items into an initial list of Sprint Tasks.

The maximum allotted time (a.k.a. *timebox*) for planning a 30-day Sprint is eight hours, reduced proportionally for a shorter Sprint.

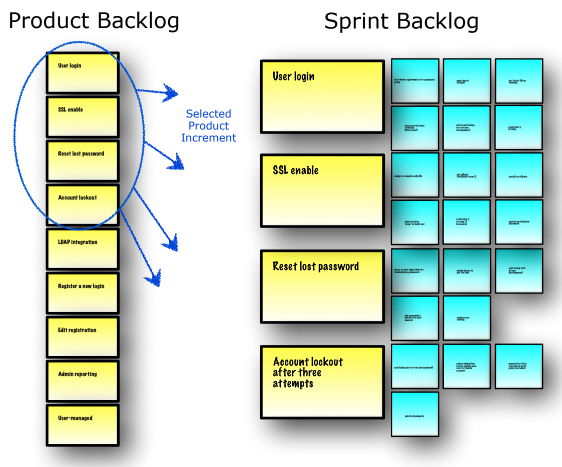


Figure 5: *Sprint Planning outcome is selected Product Backlog Items (PBIs) and subordinate Sprint Tasks.*

Daily Scrum and Sprint Execution

Every day at the same time and place the team spends 15 minutes inspecting their Sprint in progress and creating a shared plan for the day.

Standing up at the Daily Scrum helps keep it short. Topics that require additional attention may be discussed after the event by whomever is interested.

Teams find it useful to gather around *information radiators* such as a physical task-board.

During Sprint execution, it is common to discover additional work necessary to achieve the Sprint goal.

The Daily Scrum was intended to disrupt old habits of working separately, but by itself has not proven sufficient. Teams can increase collaboration further with techniques such as *mob programming*.

During the Sprint, the team strives for a rigorous definition of *done*. For example, a software item that is merely "code complete" is *not done* because untested software isn't shippable. Incomplete items are returned to the Product Backlog and ranked according to the Product Owner's revised priorities as candidates for future Sprints.

Sprint Review

The purpose of the Sprint Review is to inspect the Product Increment and adapt plans for it. The participation of customers, end users, and other interested parties provides information the Product Owner may consider acting on.

The Scrum Master may help the Product Owner and stakeholders convert their feedback to new Product Backlog Items for prioritization by the Product Owner. New scope discovery usually outpaces the team's rate of development. If the Product Owner feels that the newly discovered scope is more important than the original expectations, new scope displaces old scope in the Product Backlog. Some items will never be done.

New products, particularly software products, are hard to visualize in a vacuum. Many customers need to be able to react to a piece of functioning software to discover what they will actually want. Iterative development, a *value-driven* approach, allows the creation of products that couldn't have been specified up front in a plan-driven approach.

Sprint Retrospective

Each Sprint ends with a retrospective. The team, Product Owner, and Scrum Master reflect on their own way of working together. They inspect their behavior and take action to adapt it for future Sprints.

Dedicated Scrum Masters will find alternatives to the stale, fearful meetings everyone has come to expect. In-depth retrospectives can happen in an environment of psychological safety difficult to create in most organizations. Practices such as performance appraisals and the job title ladder hamper full trust and teamwork. But without safety the retrospective discussion may either avoid the uncomfortable issues or deteriorate into blaming and hostility.

A second impediment to insightful retrospectives is our human tendency to jump to conclusions and propose actions too quickly. The book *Agile Retrospectives* suggests steps to slow this down: Set the stage, gather data, generate insights, decide what to do, close the retrospective.[1] Another useful book *The Art of Focused Conversations* suggests focusing on questions in this order: Objective, Reflective, Interpretive, and Decisional (ORID).[2]

A third impediment to psychological safety is geographic distribution. *Dislocated* teams rarely bond as well as those in team rooms.

Scrum Masters use a variety of techniques to facilitate retrospectives, such as silent writing, timelines, and satisfaction histograms. The goals are to gain a common understanding of multiple perspectives and to develop actions that will take the team and organization to the next level.

Large Scale Scrum adds an *Overall Retrospective* to resolve cross-team problems, and problems with the organization's structure and policies.

Backlog Refinement

(Note for test takers: This is not an "event" in single-team Scrum.)

Product Backlog Items (PBIs) initially need refinement because they are too large or poorly understood. Teams use some of every Sprint (say 10%) to prepare the top of the Product Backlog for upcoming Sprints.

In Backlog Refinement, large vague items near the top are split and clarified, considering both business and technical concerns. Sometimes a subset of the team and others (e,g. customers, end users) will draft and split Product Backlog Items before involving the entire team.

While refining items, the team may estimate the amount of effort they would expend to complete items in the Product Backlog and provide other technical information to help the Product Owner prioritize them.[3]

It is common to think of a Product Backlog Item as a *User Story*.[4] In this approach, oversized PBIs may be called *epics*.

Traditional approaches break features into sequential tasks (resembling waterfall phases) that cannot be prioritized independently and lack business value from the customer's perspective. This habit is hard to break.

A skilled Scrum Master can help the team identify thin *vertical* slices of work that still have business value, while promoting a rigorous definition of "done" that includes proper testing and refactoring.

Agility requires learning to carve out small product features. For example, in a medical records application, the epic "display the entire contents of a patient's allergy records to a doctor" yielded the story "display whether or not any allergy records exist." While the engineers anticipated significant technical challenges in parsing the internal aspects of the allergy records, the presence or absence of *any* allergy was the most important thing the doctors needed to know. Collaboration between business people and technical people to split this epic yielded a story representing 80% of the business value for 20% of the effort of the original epic.

Slicing large items shortens the end-to-end cycle time with users, accelerating the discovery of their real needs.

[1] *Agile Retrospectives*, Pragmatic Bookshelf, Derby/Larson (2006)

[2] *The Art of Focused Conversations*, New Society Publishers (2000)

[3] The team should collaborate to produce a jointly-owned estimate for an item.

[4] *User Stories Applied: For Agile Software Development*, Addison Wesley, Cohn (2004)

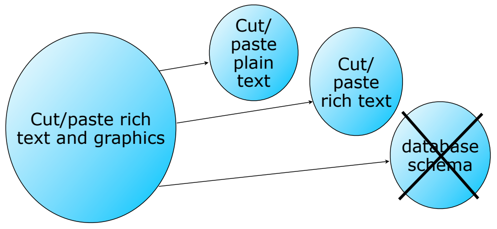


Figure 6: *During Backlog Refinement, large PBIs (often called "epics") near the top of the Product Backlog are split into thin vertical feature slices ("stories"), not horizontal implementation phases.*

Scrum Artifacts

Scrum defines three artifacts: Product Backlog, Sprint Backlog, and Increment.

Product Backlog

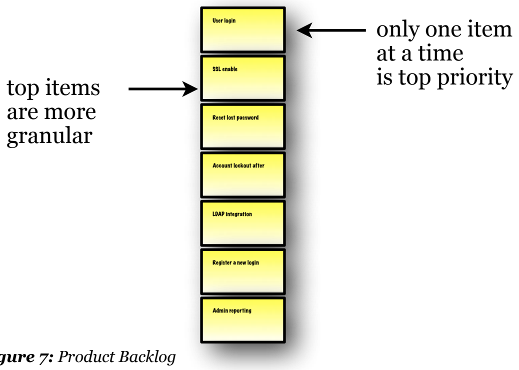


Figure 7: *Product Backlog*

- Force-ranked (prioritized) list of desired functionality
- Visible to all stakeholders
- Anyone can suggest items
- Constantly re-prioritized by the Product Owner
- Constantly refined by teamwork
- Items at top should be smaller (e.g., smaller than 1/4 of a Sprint) than items at bottom

Product Backlog Item (PBI)

- Describes the *what* (more than the *how)* of a customer-centric feature
- Often considered a *User Story*
- Has a product-wide definition of *done* to prevent technical debt
- May have item-specific acceptance criteria
- Time/effort estimate, if used, is provided by the team, ideally in relative units (e.g., story points)

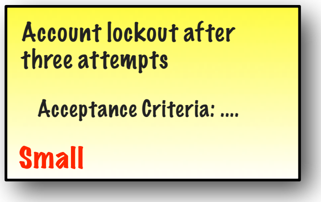
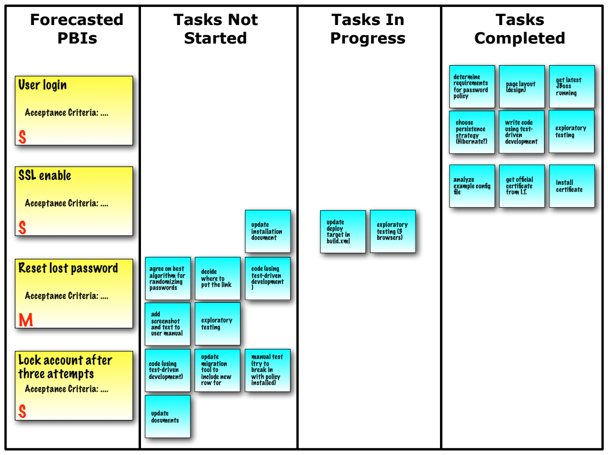

Account lockout after three attempts
Acceptance Criteria: ....
Small

Figure 8*: A PBI represents a customer-centric feature, usually requiring several tasks to achieve definition of done.*

Sprint Backlog

- PBIs selected by the team during Sprint Planning, plus their continuously-updated plan to accomplish them (e.g. Sprint Tasks)
- Initial tasks are identified by the team during Sprint Planning
- Team will discover additional work needed to meet the Sprint Goal during Sprint execution
- No changes are made during the Sprint that would endanger the Sprint Goal
- Visible to the team
- Referenced during the Daily Scrum

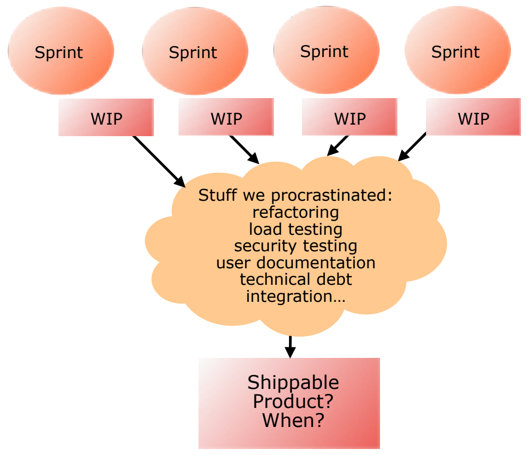


Forecasted PBIs
Tasks Not Started
Tasks In Progress
Tasks Completed
User login
Acceptance Criteria: ....
$
SSL enable
Acceptance Criteria: ....
$
Reset lost password
Acceptance Criteria: ....
M
Lock account after three attempts
Acceptance Criteria: ....
$

Figure 9: *Sprint Backlog is best represented with an "information radiator" such as a physical taskboard.*

(Potentially Shippable Product) Increment

- The product capabilities completed during the Sprints.
- Brought to a usable, shippable state at least by the end of each Sprint, or more frequently than that.
- Released as often as the Product Owner wishes.
- Inspected during every Sprint Review.
- *Definition of Done* is a standard applied to all PBIs from all contributing teams. For example, all new changes should:
    - Be properly tested.
    - Be fully integrated.
    - Be peer reviewed or developed by pair/mob programming.
    - Be documented (when applicable).
    - Maintain or improve source code understandability.

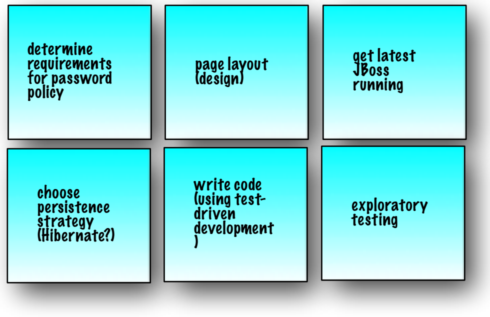


Sprint
Sprint
Sprint
Sprint
WIP
WIP
WIP
WIP
Stuff we procrastinated: refactoring load testing security testing user documentation technical debt integration…
Shippable Product? When?

Figure 10: *Un-done work causes risk and delay.*

Sprint Task (optional)

- Describes *how* to achieve the PBI's *what*
- Typically involves one day or less of work
- Owned by the team; collaboration is expected



determine requirements for password policy
page layout (design)
get latest JBoss running
choose persistence strategy (Hibernate?)
write code (using test-driven development)
exploratory testing

Figure 11: *Sprint tasks required to complete one backlog item require a mix of activities no longer done in separate phases (e.g., requirements elicitation, analysis, design, implementation, deployment, testing).*

Sprint Burndown Chart (optional)

- Summation of total team work remaining within one Sprint
- Updated daily
- May go up before going down
- Intended to help team self-management, not as a report
- Fancy variations, such as itemizing by point person or adding trend lines, tend to reduce effectiveness at encouraging collaboration
- Seemed like a good idea in the early days of Scrum, but in practice often misused as a management report, inviting intervention. Discontinue use of this chart if it reduces team self-management.

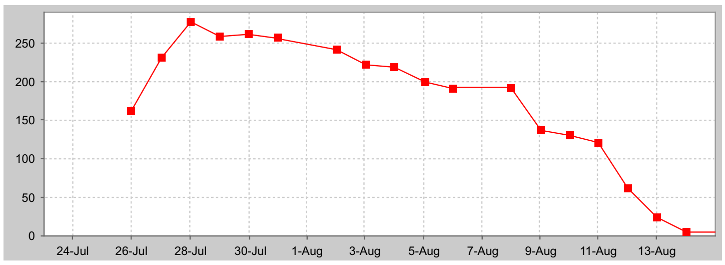


Figure 12: *Sprint Burndown Chart*

Product / Release Burndown Chart (optional)

- Tracks the remaining Product Backlog effort from one Sprint to the next
- X axis is time in Sprints
- May use relative units such as *story points* for Y axis
- Depicts historical trends to adjust forecasts

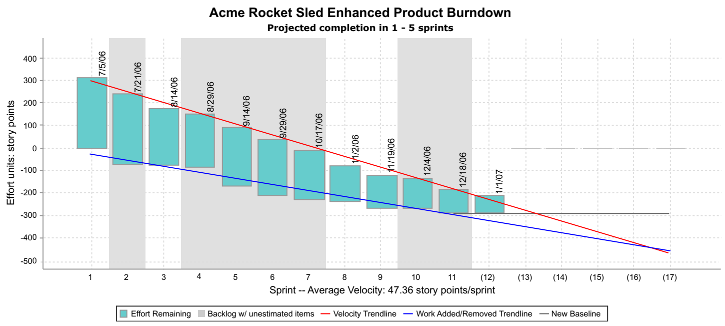


Figure 13: *A Release Burndown Chart variation popularized by Mike Cohn. The red line tracks PBIs completed over time (velocity), while the blue line tracks new PBIs added (new scope discovery). The intersection projects release completion date from empirical trends.*

Multiple Teams

Your Organization is Designed to Impede Agility

Introducing Scrum without simplifying the organization's structure and policies leads to *change theater* and no real improvement. Large organizations are usually just pretending.[5] Successful adoptions of Large Scale Scrum are both top down and bottom up.

Scrum addresses uncertain requirements and technology risks by grouping people from multiple disciplines into one team — in one team room — to increase bandwidth, visibility, and trust.

Adding too many people to a team makes things worse. Grouping people by specialty also makes things worse. Grouping people by architectural components (a.k.a. component teams) makes things worse.

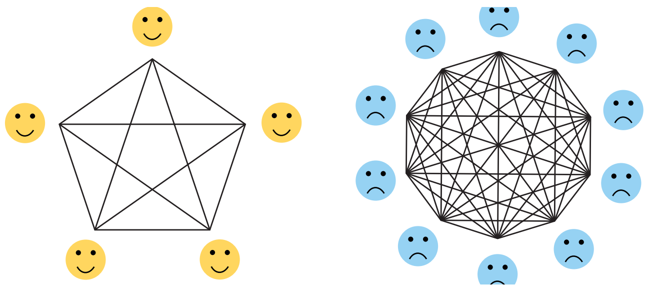


Figure 14: *Communication pathways increase as a square of team size.*

Feature Teams

Fully cross-functional "feature teams" are able to operate at all layers of the architecture in order to deliver customer-centric features to end users. In a large system, this requires learning new skills.

As teams focus on learning — rather than short-term micro-efficiencies — they can help create a *learning organization*.

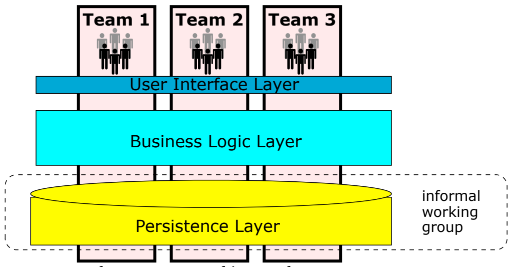


Figure 15: *Feature teams learn to span architectural components.*

One Product Backlog, One Product Owner

In Large Scale Scrum, multiple teams share a single Product Backlog prioritized by a single Product Owner. They share the responsibility of maintaining this backlog. To avoid asynchronous dependencies, they collaborate across teams in one shared Sprint, using overall and multi-team versions of the events described in this card, often with team-appointed representatives.[6] As in single-team Scrum, they attempt to develop one properly tested, integrated, shippable product increment every Sprint.

Related Practices

Lean

Scrum is a general framework coinciding with the Agile movement in software development, which is partly inspired by Lean manufacturing approaches such as the Toyota Production System.[7]

Extreme Programming (XP)

While Scrum does not prescribe specific engineering practices, Scrum Masters are responsible for promoting increased rigor in the definition of *done*. Items that are called "done" should stay done. Automated regression testing prevents *vampire stories* that leap out of the grave. Design, architecture, and infrastructure emerge over time, subject to continuous reconsideration and refinement, instead of being "finalized" at the beginning, when we know almost nothing.

The Scrum Master can inspire the team to learn engineering practices associated with XP: Continuous Integration (continuous automated testing), Test-Driven Development (TDD), constant merciless refactoring, pair programming, mob programming, frequent check-ins, etc. Informed application of these practices prevents technical debt.

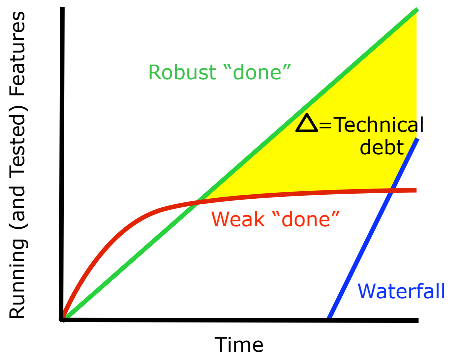


Figure 16: *The straight green line represents the general goal of Agile methods: early and sustainable delivery of valuable features. Doing Scrum properly entails learning to satisfy a rigorous definition of "done" to prevent technical debt.*[8]

[5] "Seven Obstacles to Enterprise Agility," Gantthead, James (2010) http://scrumreferencecard.com/7-obstacles-to-enterprise-agility/

[6] See http://less.works to learn about Large Scale Scrum

[7] Agile movement defined at http://agilemanifesto.org

[8] Graph inspired by discussions with Ronald E. Jeffries

Team Self-Management

Engaged Teams Outperform Manipulated Teams

During Sprint execution, team members develop an intrinsic interest in shared goals and learn to manage each other to achieve them. The natural human tendency to be accountable to a peer group contradicts years of habit for many workers. Allowing a team to become self-propelled, rather than manipulated through extrinsic punishments and rewards, contradicts years of habit for many managers.[9] The Scrum Master's observation and persuasion skills increase the probability of success, despite the initial discomfort.

Challenges and Opportunities

Self-organizing teams can radically outperform larger, traditionally managed teams. Family-sized groups naturally self-organize when the right conditions are met:

- members are committed to clear, short-term goals
- members can gauge the group's progress
- members can observe each other's contribution
- members feel safe to give each other unvarnished feedback

Psychologist Bruce Tuckman describes modes of group development as "forming, storming, norming, performing."[10] Optimal self-organization takes time. The team may perform worse during early iterations than it would have performed as a traditionally managed working group.[11]

Heterogeneous teams outperform homogeneous teams at complex work. They also experience more conflict.[12] Disagreements are normal and healthy on an engaged team; team performance will be determined by how well the team handles these conflicts.

Bad apple theory suggests that a single negative individual ("withholding effort from the group, expressing negative affect, or violating important interpersonal norms"[13]) can disproportionately reduce the performance of an entire group. Such individuals are rare, but their impact is magnified by a team's reluctance to remove them. This can be partly mitigated by giving teams greater influence over who joins them.

Other individuals who underperform in a boss/worker situation (due to being under-challenged or micromanaged) will shine on a Scrum team.

Self-organization is hampered by conditions such as geographic distribution, boss/worker dynamics, part-time team members, and interruptions unrelated to Sprint goals. Most teams will benefit from a full-time Scrum Master who works hard to mitigate these kinds of impediments.[14]

When is Scrum Appropriate?

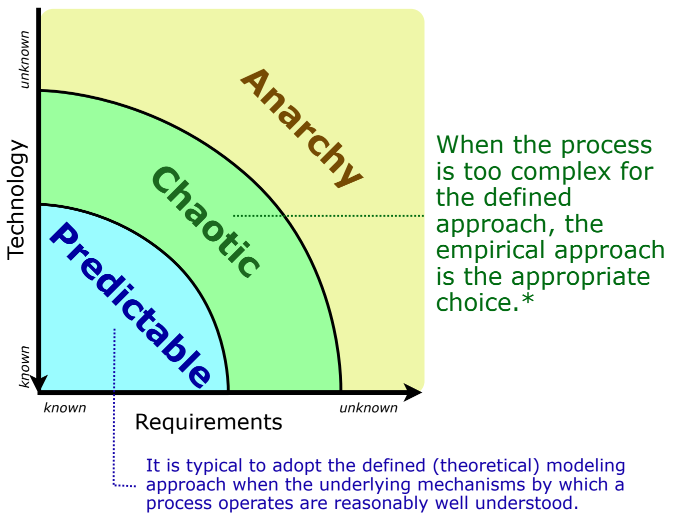

unknown
Technology
known
known
Requirements
unknown
Anarchy
Chaotic
Predictable
When the process is too complex for the defined approach, the empirical approach is the appropriate choice.*
It is typical to adopt the defined (theoretical) modeling approach when the underlying mechanisms by which a process operates are reasonably well understood.

Figure 17: Scrum, an empirical framework, is appropriate for work with uncertain requirements and uncertain technology issues.[15][16]

Scrum is intended for the kinds of work people have found unmanageable using defined processes — uncertain requirements combined with unpredictable technology implementation risks. When deciding whether to apply Scrum, as opposed to plan-driven approaches such as those described by the PMBOK® Guide, consider whether the underlying mechanisms are well-understood or whether the work depends on knowledge creation and collaboration. For example, Scrum was not originally intended for repeatable types of production and services.

Also consider whether there is sufficient commitment to grow self-organizing teams.

About the Authors

Michael James learned to program many years ago. He worked directly with Ken Schwaber to become a Scrum trainer. He coaches technical folks, managers, and executives on optimizing businesses to deliver value. Please send feedback to mj@seattlescrum.com or http://twitter.com/michaeldotjames

Luke Walter learned empirical product development many years ago as an industrial designer. He encountered Scrum on a development team with Michael James, before they both became Scrum trainers. He coaches businesses to recognize wasteful practices and organize around customer value. Please send feedback to lwalter@collab.net.

For help with Agility, please see https://seattlescrum.com.

[9] Intrinsic motivation is linked to mastery, autonomy, and purpose. "Rewards" harm this http://www.youtube.com/watch?v=u6XAPnuFjJc

[10] "Developmental Sequence in Small Groups." Psychological Bulletin, 63 (6): 384-99 Tuckman, referenced repeatedly by Schwaber.

[11] *The Wisdom of Teams: Creating the High-Performance Organization*, Katzenbach, Harper Business (1994)

[12] *Group Genius: The Creative Power of Collaboration*, Sawyer, Basic Books (2007). (This book is #2 on Michael James's list of recommended reading for Scrum Masters.)

[13] "How, when, and why bad apples spoil the barrel: Negative group members and dysfunctional groups." Research in Organizational Behavior, Volume 27, 181–230, Felps/Mitchell/Byington, (2006)

[14] An example detailed list of full-time Scrum Master responsibilities: http://ScrumMasterChecklist.org

[15] Extensively modified version of a graph in *Strategic Management and Organizational Dynamics*, Stacey (1993), referenced in *Agile Software Development with Scrum*, Schwaber/Beedle (2001).

[16] *Process Dynamics, Modeling, and Control*, Ogunnaike, Oxford University Press, 1992.