Scrum Reference Card

by Michael James and Luke Walter

About Scrum

A Management Framework

Scrum is a management framework for incremental product development using one or more cross-functional, self-organizing teams of about seven people each.

It provides a structure of roles, meetings, rules, and artifacts. Teams are responsible for creating and adapting their processes within this framework.

Scrum uses fixed-length iterations, called Sprints. Sprints are no more than 30 days long, preferably shorter. Scrum teams try to develop a potentially releasable (properly tested) product increment every Sprint.

An Alternative to Waterfall

Scrum's incremental, iterative approach trades the traditional phases of "waterfall" development for the ability to develop a subset of high-value features first, incorporating feedback sooner.

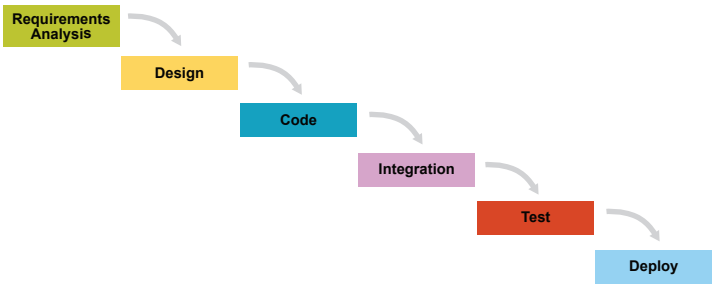

Figure 1: *Traditional "waterfall" development depends on a perfect understanding of the product requirements at the outset and minimal errors executing each phase.*

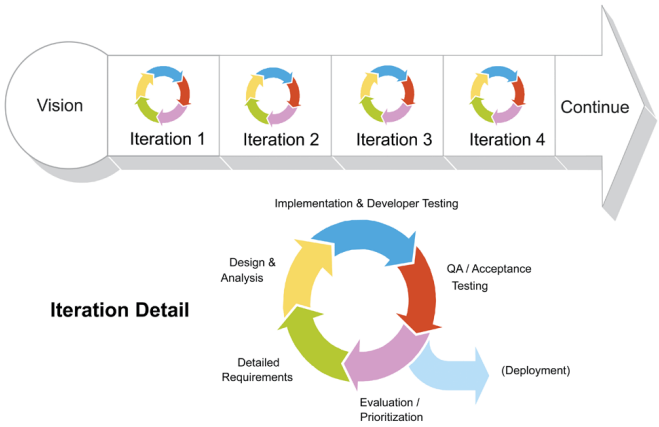

Figure 2: *Scrum blends all development activities into each iteration, adapting to discovered realities at fixed intervals.*

The greatest potential benefit of Scrum is for complex work involving knowledge creation and collaboration, such as new product development. Scrum is usually associated with object-oriented 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 relentless reality checks expose dysfunctional 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 Roles

Scrum Development Team

- Cross-functional (e.g., includes members with testing skills, and others not traditionally called developers: business analysts, designers, domain experts, etc.)
- Self-organizing / self-managing, without externally assigned roles
- Plans one Sprint at a time with the Product Owner
- Has autonomy regarding how to develop the increment
- Intensely collaborative
- Most successful when located in one team room, particularly for the first few Sprints
- Most successful with long-term, full-time membership. Scrum moves work to a flexible learning team and avoids moving people or splitting them between teams.
- 6 ± 3 members
- Has a leadership role

Product Owner

- Single person responsible for maximizing the return on investment (ROI) of the development effort
- Responsible for product vision
- Constantly re-prioritizes the Product Backlog, adjusting any long-term expectations such as release plans
- Final arbiter of requirements questions
- Decides whether to release
- Decides whether to continue development
- Considers stakeholder interests
- Has a leadership role

Scrum Master

- Works with the organization to make Scrum possible
- Ensures Scrum is understood and 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
- Has a leadership role

Scrum Meetings

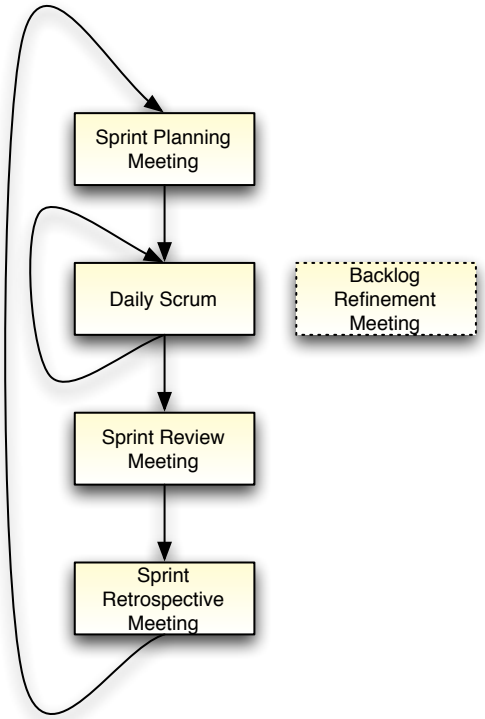

Figure 3: *Scrum flow.*

Sprint Planning Meeting

At the beginning of each Sprint, the Product Owner and team hold a Sprint Planning Meeting to negotiate which Product Backlog Items they will attempt to convert to working product during the Sprint. The Product Owner is responsible for declaring 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 "pulls" work from the Product Backlog to the Sprint Backlog.

When teams are given complex work that has inherent uncertainty, they must work together to intuitively gauge how much work to pull into a Sprint, while learning from previous Sprints. Planning their hourly capacity and comparing their estimates to actuals makes the team pretend to be precise and reduces ownership. Unless the work is truly predictable, they should discard such practices within the first few Sprints or avoid them altogether.

Until a team has learned how to complete a potentially releasable product increment each Sprint, it should reduce the amount of functionality it plans. Failure to change old habits leads to technical debt and eventual design death, as shown in Figure 14.

If the top of the Product Backlog has not been refined, a portion of the Planning meeting might be spent doing this.

Toward the end of the Sprint Planning Meeting, the team determines 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.

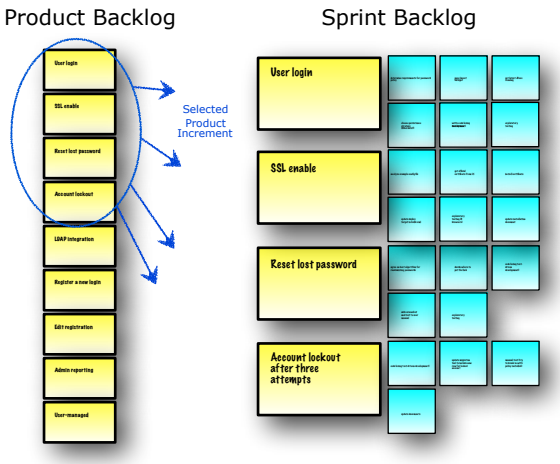

Figure 4: *Sprint Planning Meeting outcome is committed Product Backlog Items (PBIs) and subordinate Sprint Tasks.*

Daily Scrum and Sprint Execution

Every day at the same time and place, the Scrum Development Team members spend a total of 15 minutes inspecting their progress toward the Sprint goal, and creating a plan for the day. Team members may share with each other what they did the previous day to help meet the Sprint goal, what they'll do today, and what impediments they face.

Standing up at the Daily Scrum will help keep it short. Topics that require additional attention may be discussed by whoever is interested after every team member has spoken.

The team may find it useful to maintain a current Sprint Task List, and a Sprint Burndown Chart. During Sprint execution it is common to discover additional tasks necessary to achieve the Sprint goals. Impediments caused by issues beyond the team's control are considered *organizational impediments*.

The Daily Scrum is intended to disrupt old habits of working separately. Members should remain vigilant for signs of the old approach. For example, looking only at the Scrum Master when speaking is one symptom that the team hasn't learned to operate as a self-organizing entity.

Sprint Review Meeting

At the end of the Sprint, the Scrum Team holds a Sprint Review Meeting to inspect and adapt the product as it emerges. They demonstrate a working product increment to everyone who is interested, particularly customers and end users, and get their feedback.

The team reviews the items selected during the Sprint Planning Meeting and explains which items are considered *done*. For example, a software item that is merely "code complete" is considered *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.

The Scrum Master may help the Product Owner and stakeholders convert their feedback to new Product Backlog Items for prioritization by the Product Owner. Often, new scope discovery 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.

External stakeholders and end users should participate. 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 Meeting

Each Sprint ends with a retrospective. At this meeting, the team reflects on its own process. 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. An in-depth retrospective requires an environment of psychological safety not found in most organizations. Without safety, the retrospective discussion will either avoid the uncomfortable issues or deteriorate into blaming and hostility.

A common impediment to full transparency on the team is the presence of people who conduct performance appraisals.

Another impediment to an insightful retrospective is the human tendency to jump to conclusions and propose actions too quickly. *Agile Retrospectives*, the most popular book on this topic, describes a series of steps to slow this process down: Set the stage, gather data, generate insights, decide what to do, close the retrospective.[1] Another guide recommended for Scrum Masters, *The Art of Focused Conversations*, breaks the process into similar steps: Objective, reflective, interpretive, and decisional (ORID).[2]

A third impediment to psychological safety is geographic distribution. Geographically dispersed teams usually do not collaborate as well as those in team rooms.

Retrospectives often expose organizational impediments. Once a team has resolved the impediments within its immediate influence, the Scrum Master should work to expand that influence, chipping away at the organizational impediments.

Scrum Masters should use a variety of techniques to facilitate retrospectives, including silent writing, timelines, and satisfaction histograms. In all cases, 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.

Backlog Refinement Meeting

Most Product Backlog Items (PBIs) initially need refinement because they are too large and poorly understood. While Backlog Refinement is not a required event, it is a required activity. Most Scrum Teams find it useful to take a short time out of every Sprint for this activity. They get together to prepare the Product Backlog for upcoming Sprint Planning Meetings.

In the Backlog Refinement Meeting, large vague items are split and clarified, considering both business and technical concerns. Sometimes a subset of the team, in conjunction with the Product Owner and other stakeholders, will compose 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 provides other technical information to help the Product Owner prioritize them.[3]

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.

It is common to write Product Backlog Items in *User Story* form.[4] In this approach, oversized PBIs are called *epics*. Traditional development breaks features into horizontal tasks (resembling waterfall phases) that cannot be prioritized independently and lack business value from the customer's perspective. This habit is hard to break.

Agility requires learning to split large epics into user stories representing very 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.

Since most customers don't use most features of most products, it's wise to split epics to deliver the most valuable stories first. While delivering lower-value features later is likely to involve some rework, rework is better than no work.

This activity has also been called "Backlog Grooming."

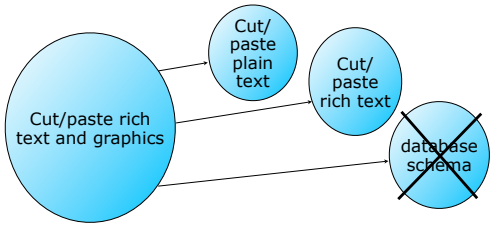

Figure 5: *During Backlog Refinement, large PBIs (often called "epics") near the top of the Product Backlog are split into thin vertical feature slices ("stories"), not horizontal implementation phases.*

Scrum Artifacts

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

Product Backlog

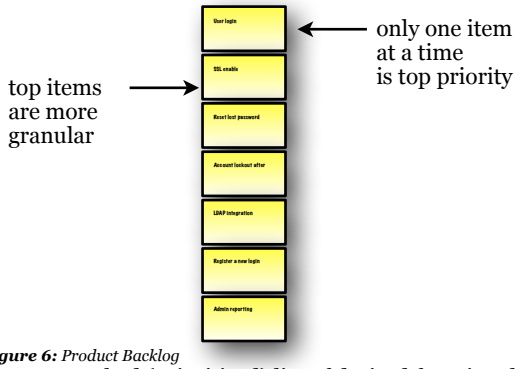

Figure 6: *Product Backlog*

- Force-ranked (prioritized) list of desired functionality
- Visible to all stakeholders
- Any stakeholder (including the Team) can add items
- Constantly re-prioritized by the Product Owner
- Constantly refined by the Scrum Team
- Items at top should be smaller (e.g. smaller than 1/4 of a Sprint) than items at bottom

---

[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)

Product Backlog Item (PBI)

- Describes the *what* (more than the *how)* of a customer-centric feature
- Often written in *User Story* form
- Has a product-wide definition of *done* to prevent technical debt
- May have item-specific acceptance criteria
- Effort is estimated by the Development Team, ideally in relative units (e.g., story points)

Figure 7*: A PBI represents a customer-centric feature, usually requiring several tasks to achieve definition of done.*

Sprint Backlog

- Consists of selected PBIs negotiated between the team and the Product Owner during the Sprint Planning Meeting
- No changes are made during the Sprint that would endanger the Sprint Goal.
- Initial tasks are identified by the team during Sprint Planning Meeting
- Team will discover additional tasks needed to meet the Sprint Goal during Sprint execution
- Visible to the team
- Referenced during the Daily Scrum Meeting

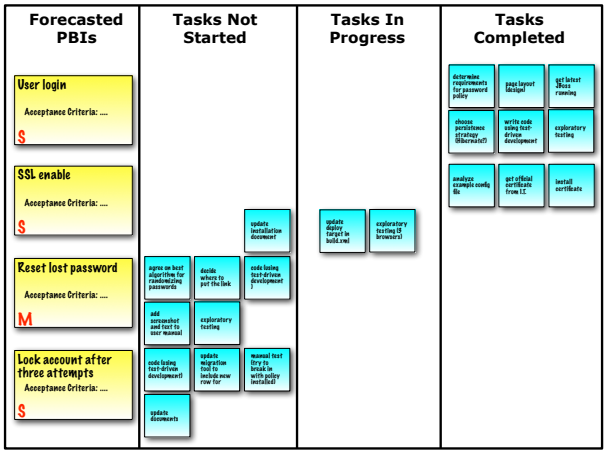

Figure 8: *Sprint Backlog is best represented with an "information radiator" such as a physical taskboard.*

Increment

- The product capabilities completed during the Sprints
- Brought to a usable, releasable state by the end of each Sprint
- Released as often as the Product Owner wishes
- Inspected during every Sprint Review Meeting

Sprint Task (optional)

- Describes *how* to achieve the PBI's *what*
- Typically involves one day or less of work
- During Sprint Execution, a *point person* may volunteer to be primarily responsible for a task
- Owned by the entire team; collaboration is expected

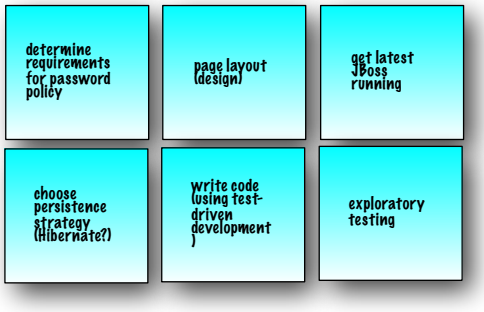

Figure 9: *Sprint tasks required to complete one backlog item require a mix of activities no longer done in separate phases (e.g., requirements elicitation, analysis, design, implementation, deployment, testing).*

Sprint Burndown Chart (optional)

- Summation of total team work remaining within one Sprint
- Updated daily
- May go up before going down
- Intended to facilitate team self-organization
- 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. The Scrum Master should discontinue use of this chart if it becomes an impediment to team self-organization.

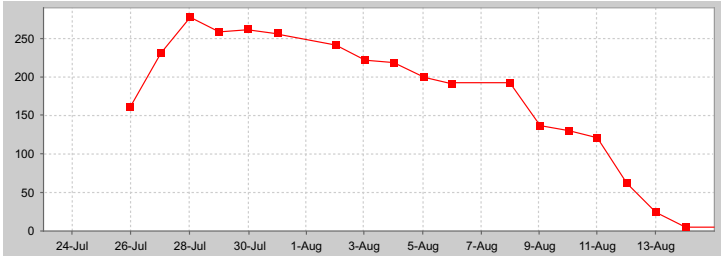

Figure 10: *Sprint Burndown Chart*

Product / Release Burndown Chart (optional)

- Tracks the remaining Product Backlog effort from one Sprint to the next
- May use relative units such as *Story Points* for Y axis
- Depicts historical trends to adjust forecasts

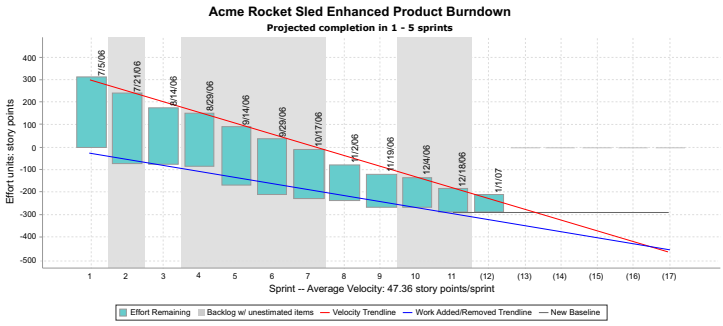

Figure 11: *A Release Burndown Chart variation popularized by Mike Cohn.[5] The red line tracks PBIs completed over time (velocity), while the blue line tracks new PBIs added (new scope discovery). The intersection projects release completion date from empirical trends.*

[5] *Agile Estimation and Planning*, Cohn, Addison Wesley (2006)

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.[6] 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.

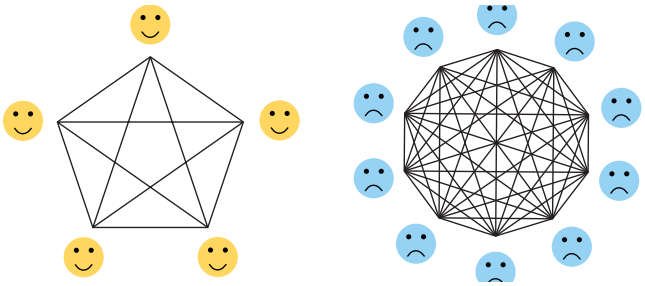

Figure 12: *Communication pathways increase as a square of team size.*

Feature Teams

Fully cross-functional "feature teams" are able to operate at all layers of the architecture in order to deliver customer-centric features. 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*.

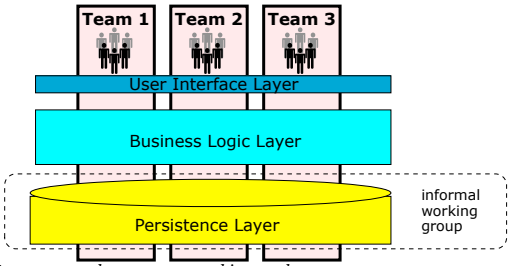

Figure 13: *Feature teams learn to span architectural components.*

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 meetings described in this card, often with team-appointed representatives.[7] 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.[8]

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 must emerge over time, subject to continuous reconsideration and refinement, instead of being "finalized" at the beginning, when we know 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.

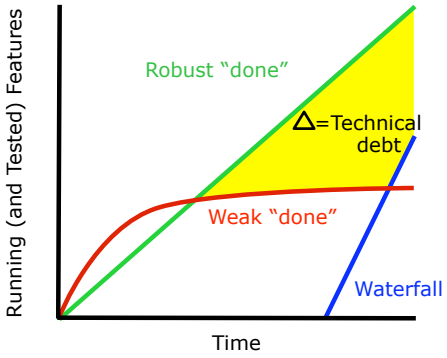

Figure 14: *The straight green line represents the general goal of Agile methods: early and sustainable delivery of valuable features. Doing Scrum properly entails learning to satisfy a rigorous definition of "done" to prevent technical debt.*[9]

[6] "Seven Obstacles to Enterprise Agility," Gantthead, James (2010) http://www.gantthead.com/content/articles/255033.cfm

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

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

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

Team Self-Organization

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 workers. Allowing a team to become self-propelled, rather than manipulated through extrinsic punishments and rewards, contradicts years of habit for managers.[10] 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 stages of group development as "forming, storming, norming, performing."[11] Optimal self-organization takes time. The team may perform worse during early iterations than it would have performed as a traditionally managed working group.[12]

Heterogeneous teams outperform homogeneous teams at complex work. They also experience more conflict.[13] 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"[14]) 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.[15]

When is Scrum Appropriate?

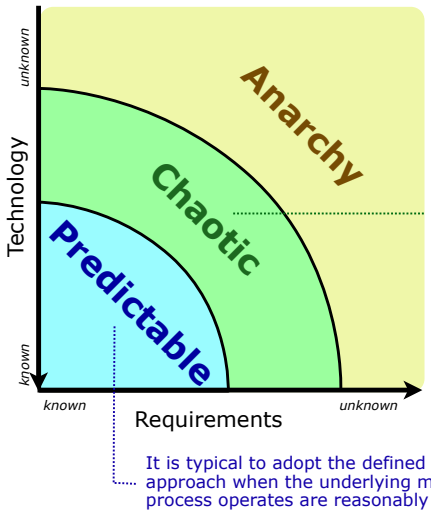

Figure 15: Scrum, an empirical framework, is appropriate for work with uncertain requirements and/or uncertain technology issues.[16][17]

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 a self-organizing team.

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 http://seattlescrum.com.

---

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

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

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

[13] *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.)

[14] "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)

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

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

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