Scrum Reference Card
by Michael James, CollabNet Inc.

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.

Scrum 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, which are typically two weeks or 30 days long. Scrum teams attempt to build a potentially shippable (properly tested) product increment every iteration.

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.
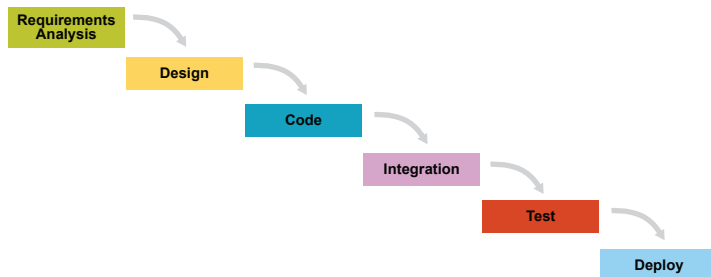

Figure 1: Traditional "waterfall" development depends on a perfect understanding of the product requirements at the outset and minimal errors executing each phase.
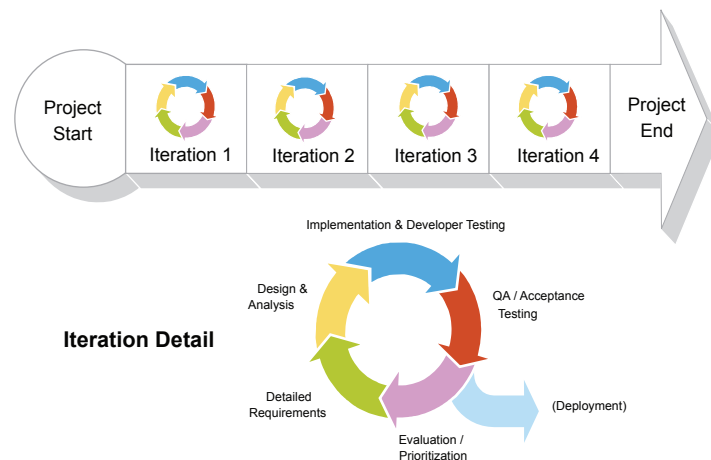

Figure 2: Scrum blends all development activities into each iteration, adapting to discovered realities at fixed intervals.

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

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
- Accepts or rejects each product increment
- Decides whether to ship
- Decides whether to continue development
- Considers stakeholder interests
- May contribute as a team member
- Has a leadership role

Scrum Development Team

- Cross-functional (e.g., includes members with testing skills, and often others not traditionally called developers: business analysts, domain experts, etc.)
- Self-organizing / self-managing, without externally assigned roles
- Negotiates commitments with the Product Owner, one Sprint at a time
- Has autonomy regarding how to reach commitments
- 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.
- 7 ± 2 members
- Has a leadership role

ScrumMaster

- Facilitates the Scrum process
- Helps resolve impediments
- Creates an environment conducive to team self-organization
- Captures empirical data to adjust forecasts
- Shields the team from external interference and distractions to keep it in group flow (a.k.a. the *zone*)
- Enforces timeboxes
- Keeps Scrum artifacts visible
- Promotes improved engineering practices
- Has no management authority over the team (anyone with authority over the team is by definition not its ScrumMaster)
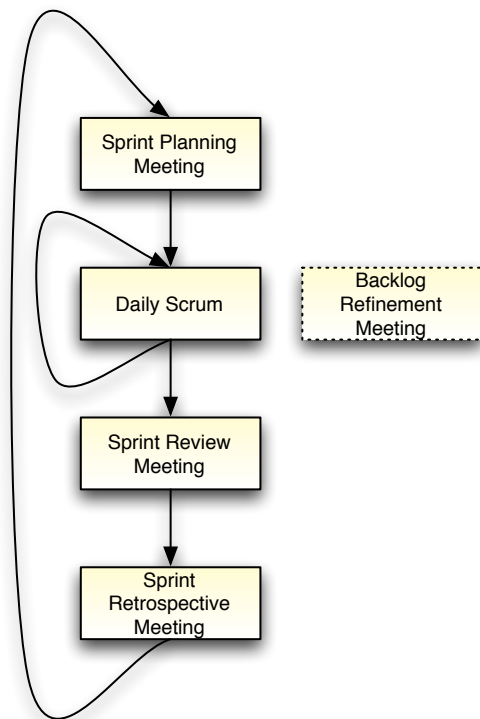- Has a leadership role

Scrum Meetings

Figure 3: Scrum flow.

All Scrum Meetings are facilitated by the ScrumMaster, who has no decision-making authority at these meetings.

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 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 their capacity to commit to items, 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 of their commitments. 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-shippable product increment each Sprint, it should reduce the amount of functionality it commits to. 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 major portion of the planning meeting should be spent doing this, as described in the Backlog Refinement Meeting section.

Toward the end of the Sprint Planning Meeting, the team breaks the selected items into an *initial* list of Sprint Tasks, and makes a final commitment to do the work.

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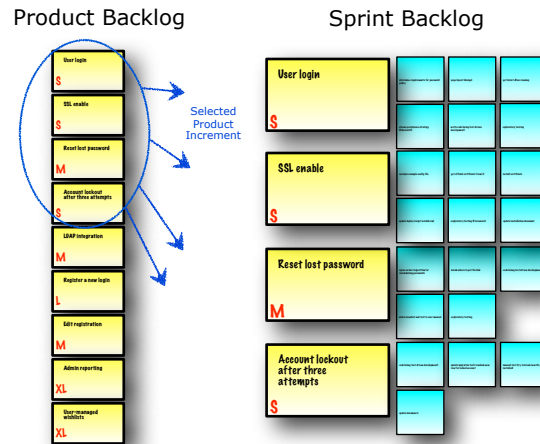

Figure 3: Sprint Planning Meeting outcome is committed Product Backlog Items (PBIs) and subordinate Sprint Tasks.

Daily Scrum and Sprint Execution

Every day at the same time and place, the Scrum Development Team members spend a total of 15 minutes reporting to each other. Each team member summarizes what he did the previous day, what he will do today, and what impediments he faces.

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

The team may find it useful to maintain a current Sprint Task List, a Sprint Burndown Chart, and an Impediments List. 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*.

It is almost always useful for the Product Owner to attend the Daily Scrum. But when any attendee also happens to be the team's boss, the *invisible gun* effect hampers self-organization and emergent leadership. People lacking real experience of team self-organization won't see this problem, just as fish are unaware of water. Conversely, a team that needs additional expertise in product requirements will benefit from increased Product Owner involvement, including Daily Scrum attendance.

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 ScrumMaster when speaking is one symptom that the team hasn't learned to operate as a self-organizing entity.

Sprint Review Meeting

After Sprint execution, the team holds a Sprint Review Meeting to demonstrate a working product increment to the Product Owner and everyone else who is interested.

The meeting should feature a live demonstration, not a report.

After the demonstration, the Product Owner reviews the commitments made at the Sprint Planning Meeting and declares which items he now considers *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 ScrumMaster helps 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.

The Sprint Review Meeting is the appropriate meeting for external stakeholders (even end users) to attend. It is the opportunity to inspect and adapt the product as it emerges, and iteratively refine everyone's understanding of the requirements. 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 ScrumMasters 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 ScrumMasters, *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 ScrumMaster should work to expand that influence, chipping away at the organizational impediments.

ScrumMasters 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 to the next level.

Backlog Refinement Meeting

Most Product Backlog Items (PBIs) initially need refinement because they are too large and poorly understood. Teams have found it useful to take a little time out of Sprint Execution — every Sprint — to help prepare the Product Backlog for the next Sprint Planning Meeting.

In the Backlog Refinement Meeting, the team estimates 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] 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 in estimation.

A skilled ScrumMaster 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.

The Backlog Refinement Meeting lacks an official name and has also been called "Backlog Grooming," "Backlog Maintenance," or "Story Time."
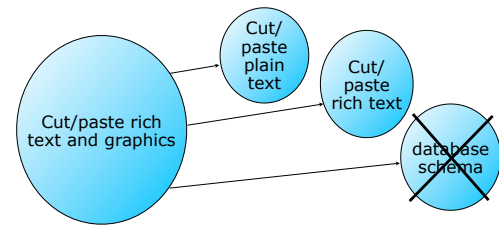

Figure 4: During Backlog Refinement, large PBIs (often called "epics") near the top of the Product Backlog are split into thin vertical feature slices ("stories"), not horizontal implementation phases.

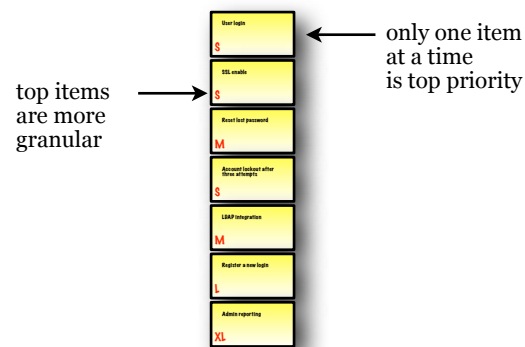# Scrum Artifacts

Product Backlog

Figure 5: Product Backlog

- Force-ranked list of desired functionality
- Visible to all stakeholders
- Any stakeholder (including the Team) can add items
- Constantly re-prioritized by the Product Owner
- Items at top are more granular than items at bottom
- Maintained during the Backlog Refinement Meeting

---

[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. See http://blogs.danube.com/estimation-game

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

Product Backlog Item (PBI)

- Specifies 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 team, ideally in relative units (e.g., story points)
- Effort is roughly 2-3 people 2-3 days, or smaller for advanced teams

Account lockout after three attempts

Acceptance Criteria: ....

Small

Figure 6*: A PBI represents a customer-centric feature, usually requiring several tasks to achieve definition of done.*

Sprint Backlog

- Consists of committed PBIs negotiated between the team and the Product Owner during the Sprint Planning Meeting
- Scope commitment is fixed during Sprint Execution
- Initial tasks are identified by the team during Sprint Planning Meeting
- Team will discover additional tasks needed to meet the fixed scope commitment during Sprint execution
- Visible to the team
- Referenced during the Daily Scrum Meeting

Figure 7*: Sprint Backlog is often represented with an "information radiator" such as a physical taskboard.*

PBIs	Not Started — 7 Tasks	Impeded — 0 Tasks	In Progress — 9 Tasks	Done — 57 Tasks
Add SVN revision numbe... Estimate: 2 ☐ Done [+ Task]				Do It — Hrs: 0 — Zoltan Szugyi
Web Client log in with... Estimate: 1 ☐ Done [+ Task]				Fix It — Hrs: 0 — Kevin Hobbs
Add new SWP fields to ... Done when: - customer number - indicate SW edition - form fields which generate the file Estimate: 2 ☐ Done [+ Task]				Add customer number, e... — Hrs: 0 — Eric Barendt; Update license generators — Hrs: 0 — Eric Barendt; Nice error messages — Hrs: 0 — Eric Barendt; Inform user when SWBas... — Hrs: 0 — Eric Barendt
Add support "link" to ... to hardcoded page with customer number in the url Estimate: 1 ☐ Done [+ Task]				Do It — Hrs: 0 — Victor Szalvay; Do It — Hrs: 0 — Eric Barendt
Display # of Licensed ... Swing client 'About' page Estimate: 2 ☐ Done [+ Task]				Do It — Hrs: 0 — Kevin Hobbs
Downgrade from SW Pro... and re-upgrade from a previous Pro upgrade. Estimate: 6 ☐ Done [+ Task]			Test throughly — Hrs: 0 — Kelly Louie	Installer should downg... — Hrs: 0 — Eric Barendt; Did we remove columns ... — Hrs: 0 — Eric Barendt

Figure 8*: Sprint Backlog may also be represented electronically in a collaboration tool such as ScrumWorks® Pro.*

Sprint Task

- Specifies *how* to achieve the PBI's *what*
- Requires one day or less of work
- Remaining effort is re-estimated daily, typically in hours
- During Sprint Execution, a *point person* may volunteer to be primarily responsible for a task
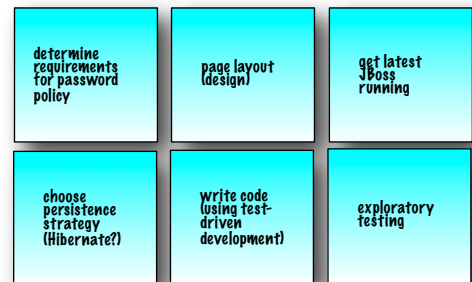- Owned by the entire team; collaboration is expected

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 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

- Indicates total remaining team task hours within one Sprint
- Re-estimated daily, thus 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 has often been misused as a management report, inviting intervention. The ScrumMaster should discontinue use of this chart if it becomes an impediment to team self-organization.
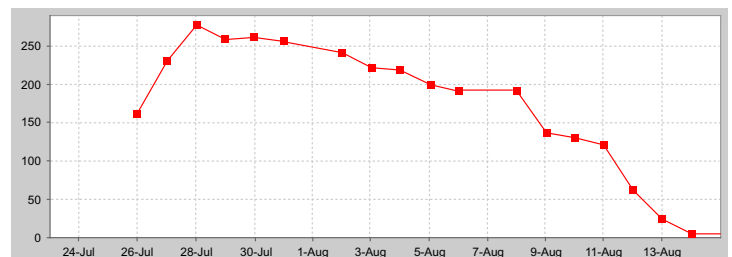

Figure 10*: Sprint Burndown Chart*

COLLABNET®

Product / Release Burndown Chart

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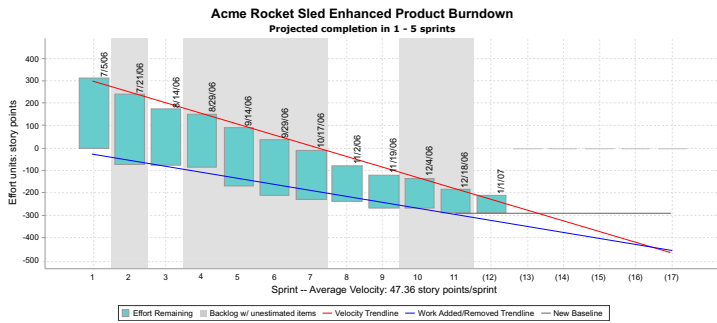- Depicts historical trends to adjust forecasts

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.[6]*
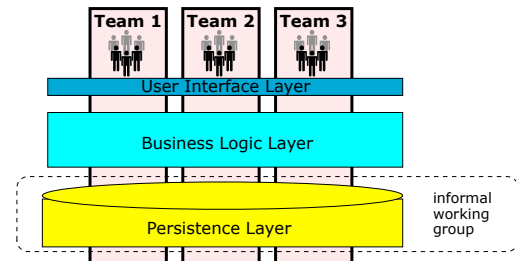
Scaling

Bad News: It's Hard.

Scrum addresses uncertain requirements and technology risks by grouping people from multiple disciplines into one team (ideally in one team room) to maximize communication bandwidth, visibility, and trust.

When requirements are uncertain and technology risks are high, adding too many people to the situation makes things worse. Grouping people by specialty also makes things worse. Grouping people by architectural components (a.k.a. component teams) makes things worse . . . eventually.

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

Good News: Feature Teams May Help.

The most successful approach to this problem has been the creation of fully cross-functional "feature teams," 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.*

More Bad News: It's Still Hard.

Large organizations are particularly challenged when it comes to Agility. Most have not gotten past pretending to do Scrum.[7] ScrumMasters in large organizations should meet with each other regularly, promoting transformation through a visible list of organizational impediments, and read books such as *Scaling Lean & Agile Development.[8]*

Related Practices

Lean

Scrum is a general management framework coinciding with the Agile movement in software development, which is partly inspired by Lean manufacturing approaches such as the Toyota Production System.[9]

eXtreme Programming (XP)

While Scrum does not prescribe specific engineering practices, ScrumMasters 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 ScrumMaster 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, frequent check-ins, etc. Informed application of these practices prevents technical debt.
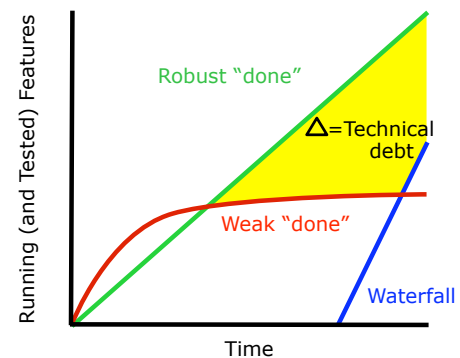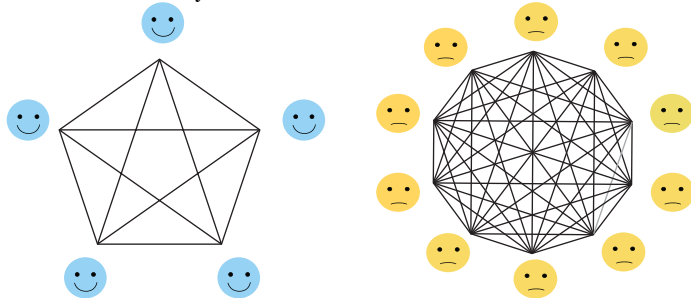

Figure 14: *The straight green line represents the general goal of Agile methods: early and sustainable delivery of valuable features. Doing Scrum properly entails learning to satisfy a rigorous definition of "done" to prevent technical debt.[10]*

---

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

[6] This example graph produced for Wiley E. Coyote by CollabNet ScrumWorks® http://www.scrumworks.com

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

[8] *Scaling Lean & Agile Development*, Larman/Vodde, Addison Wesley (2008)

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

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

COLLABNET®

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.[11] The ScrumMaster'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."[12] Optimal self-organization takes time. The team may perform worse during early iterations than it would have performed as a traditionally managed working group.[13]

Heterogeneous teams outperform homogeneous teams at complex work. They also experience more conflict.[14] 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"[15]) 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 ScrumMaster who works hard to mitigate these kinds of impediments.[16]

When is Scrum Appropriate?

It is typical to adopt the defined (theoretical) modeling approach when the underlying mechanisms by which a process operates are reasonably well understood.

When the process is too complicated for the defined approach, the empirical approach is the appropriate choice.
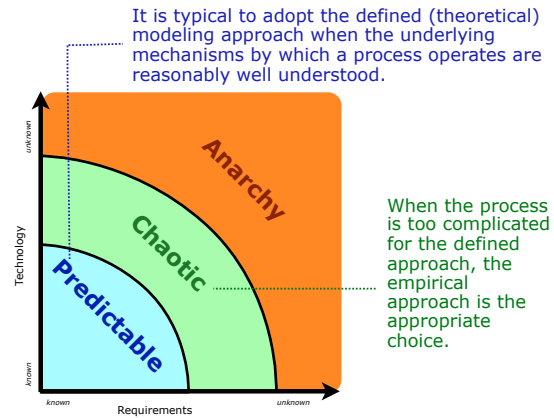
Figure 15: Scrum, an empirical framework, is appropriate for work with uncertain requirements and/or uncertain technology issues.[17] [18]

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 Author

Michael James learned to program a fairly long time 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. He invites feedback at mj@collab.net or http://twitter.com/michaeldotjames

About CollabNet

CollabNet produces ScrumWorks® Basic and Pro — free and low-cost products to help manage Scrum development. CollabNet also produces TeamForge®, an Agile Application Lifecycle Management (ALM) platform optimized around Subversion®, CollabNet's open-source revision control system with millions of users. CollabNet provides Scrum training and consulting. Visit http://www.collab.net/scrumtraining for details.

---

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

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

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

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

[15] "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)

[16] An example detailed list of full-time ScrumMaster responsibilities: http://blogs.danube.com/a-scrummasters-checklist

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

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

COLLABNET®