A Case for Scrumban

Halim Dunsky
hdunsky@solutionsiq.com
@halimdunsky

Abstract

Scrum and Kanban are Agile workflow frameworks that can contribute significantly to software quality. Aspects of a workflow context can be assessed to suggest using one or the other in a particular situation. The two approaches also share many attributes, and have been shown to work well together in mixed and blended configurations, generically termed Scrumban. Several Scrumban configurations are presented and discussed.

Biography

Halim Dunsky is a Director at Agile consultancy SolutionsIQ, and has over 35 years of experience in leadership and technical roles at firms including Microsoft, Deloitte and Touche, and Bank of America. At SolutionsIQ, Halim has helped shape large Agile transformation and organizational change programs and has used Scrum and Kanban with clients and as a member of the SIQ management team. Halim was Director of Operations and a founding Core Faculty member at Bainbridge Graduate Institute, where he taught systems thinking to MBA students. He received his M.A. in Organizational Systems from Saybrook Graduate School.

Introduction

> It's a shame that parallel lines never meet. They have so much in common!
>
> – Anonymous

It's likely that many readers have heard or participated in debates – sometimes verging on religious argument – about Scrum and Kanban and why one is better than the other. It seems clear, though, that the best tool will differ depending on the situation. Workflow methods can be important contributors to software quality, and finding a good match for the need is worthwhile.

It also turns out that Scrum and Kanban have a lot in common, and work well in combination. These combined uses can generically be termed Scrumban. *Mixed designs* configure multiple teams such that some use Scrum and some, Kanban. *Blended designs* combine attributes of both into the operations of a single team.

This paper assumes that readers have a foundational understanding of Scrum and Kanban[1,2]. We will first review some of the common attributes of these Agile workflow management frameworks. Next, we'll consider the question of the potential benefits for software quality of the two. We'll also look at

contextual indications favoring one or the other. In the final section, several mixed designs will be introduced and discussed, and a sample blended design will be presented.

Similarities of Scrum and Kanban

It will be helpful first to consider some of the dimensions of similarity between these two Agile workflow frameworks.

Pull-Based: Work items are pulled by the team as their capacity allows, they are not pushed onto the team by a scheduling authority. This is fundamental to the reduction of waste and the optimization of throughput. As the authority to pull reflects the autonomy of the team, it can also have a significant beneficial effect on the psychological atmosphere[3].

Process Policies: Policies state the criteria under which which an item of work is permitted to advance from one state to another. Policies generally reflect a quality bar. Teams establish policies by agreement; this is again autonomy.

Incremental and Iterative: By handling work in small pieces or small batches, both approaches make it natural to build on an early version of a feature with progressive improvement. Although Kanban is not intrinsically iterative, due to the provision for continuous input queuing, a response to feedback concerning a completed work item is easily incorporated in a future work item. This is explicit in Scrum in the expectation that sprint review feedback influences the future product backlog. These approaches allow both for new direction due to learning and for prioritization over time of successive improvements.

Kaizen: Continuous process inspection and improvement is built in as a foundational expectation.

Why Does Workflow Matter for Software Quality?

Let's begin with a quick review of the dimensions of software quality.

It's common in Agile circles to first distinguish *doing the right thing* and *doing the thing right*, where the former refers to requirements fit and the latter refers to quality of construction. Dimensions of software quality can be elaborated in many ways. Here's a simplified rendering of one example[4]:

- Accessibility
- Compatibility
- Concurrency
- Efficiency
- Functionality
- Installability
- Localizability
- Maintainability
- Performance
- Portability
- Reliability
- Scalability
- Security
- Testability
- Usability

I would add Timeliness and feel pretty good about this list.

Many of the attributes of Scrum and Kanban lend themselves to the support of these dimensions or goals:

Small, well-defined units of work: A small unit of functionality or work item, particularly when rendered as a well-formed user story, is easier to understand, update, and test than a large specification, which may suffer from overlaps, inconsistencies, unrecognized dependencies, prioritization difficulties, and obsolescence. A modular specification format shares many of the advantages of modularity in software: well-defined work items have low coupling, high cohesion, and well-defined interfaces. Well-defined modularity in specifications can support the emergence of well-defined modularity in the software under construction.

Incremental and iterative development: Teams are encouraged to specify and build first the simplest thing that can possibly work. This creates the fastest opportunity to begin to prove the intended concept, supporting mid-course correction as the functionality is extended and enriched. It also makes explicit the opportunity to conduct in stages the work that will support more challenging dimensions of quality.

Focus on delivering working software: In contrast to large Waterfall projects that deliver major units of functionality at long intervals, these Agile methods explicitly focus on producing small units of working, tested software at short intervals. This supports the incremental and iterative emergence of a near-continuously functional codebase that lends itself to more frequent releases and assessment of goodness of fit.

Focus on quality standards: Definition of Done and Process Policies explicitly constitute a team agreement to hold themselves and each other accountable to a range of applicable quality standards. This is one foundation for a healthy culture of shared responsibility for excellence.

Focus on bringing testing early into development: Testing is viewed as ultimately or aspirationally the responsibility of the delivery team. Testing in all its dimensions is brought forward as much as possible to the time when the software is first created, when defects are less costly to repair.

Continuous stakeholder participation: Daily contact with Product Owners helps insure that software fit to requirements by clarification and controlling drift. Frequent demonstrations to other stakeholders serve a similar purpose: every two weeks or on a similar short cadence, incremental working software is presented for feedback.

Frequent product feedback: Shorter release cycles get the product or system in front of its users more quickly, allowing for more rapid corrective feedback to improve product-market fit. The value here can be explicitly emphasized under conditions of high uncertainty with the Lean Startup cycle of Build-Measure-Learn, which can be used equally well with Scrum or Kanban.

Get ROI on-stream earlier, stop when it makes sense: Shorter release cycles can also bring ROI on-stream earlier, which can support the continued funding of valuable software efforts and can help organizations recognize when the value may not justify continued investment.

YAGNI, 80/20: YAGNI stands for You Ain't Gonna Need It. If the organization can refrain from trying to do too much, this can reduce the schedule pressure that leads to cutting corners on software quality. Prioritization is a fundamental element of these Agile approaches, which emphasize building the software with the highest business or technical value first. Modular specification and the concept of a continuously evolving backlog, when coupled with appropriate organizational approaches to funding, act to counter historical pressures to jam as much functionality as possible into a requirements definition.

Kaizen: The practice of regular team retrospectives is a foundation for a culture of continuous improvement within the team. The practice of making impediments explicit and escalating them as needed can also be a foundation for organizational and technological improvement where the roots of an issue or its mitigations are beyond the team.

Making a Choice

Many characteristics of a workflow or team context will suggest whether Scrum or Kanban may be a better choice for that situation. Here are some general guidelines.

Scrum is Good When…

A sprint cadence provides helpful discipline: A regular sprint cadence serves to help teams develop good habits of defining, completing, and demonstrating small units of work that can be accomplished within the sprint length. This forcing function is not present in Kanban. The sprint cadence can also simplify some cross-team planning situations and support regular engagement of the stakeholder community.

Story size is not too big: A product backlog item needs to be completed within the timespan of a single sprint. Where some units of work seem to be inherently too big for this without awkward or expensive partitioning, Kanban may be a better choice. Note, though, that requirements often arrive in large units of functionality, and learning to break stories down to consumable size can take practice.

Work items are practical to estimate: It's important in Scrum to be able to know enough about an item of work that it can be sized to a reasonable approximation before scheduling. Relative sizing methods are generally used for this, and they deliver good results. Another approach to this is to break stories into units that are so small they are all about the same size and can simply be counted. One type of work unit for which either method is problematical is the defect. In the general case it's notoriously difficult to know ahead of time even the approximate size of the job of resolving a given defect. Defect flows are difficult to accommodate within a planned Scrum backlog, but sometimes this is desirable or necessary in order to keep the fix responsibility with those who generated the issue.

The organization is ready for a significant step: Succeeding with Scrum can require significant shifts in role expectations, planning, requirements definition, and the working relationship between a software development organization and its business stakeholders.

Kanban is Good When…

There is a continuous flow of work item input: The continuous flow of work through a Kanban team is best matched by a continuous flow of arriving work requests. This is often different from the large batch arrival of work that is typical larger projects and is more suited to a Scrum context.

Work items tend to be:

> **Smaller and/or repetitive:** Although a workflow of items of this kind can be handled in Scrum, a Kanban flow seems ideal for maximizing throughput in this situation. Control chart analysis will yield better predictability when items are repetitive or similar than when the workflow has a lot of variation. Predictability is key to being able to establish service level expectations with stakeholders.

Bigger than a sprint and hard to partition: In some situations it may not be possible to break stories down into small units that can be completed within a sprint. These can be handled in Kanban without process stress. Note, however, that many items that look big have simply not been decomposed yet.

Emergent, date-driven, or expected quickly rather than at sprint boundaries: Very quick turnaround is not natural to Scrum, where planning and delivery occur at intervals of a sprint length, typically 2 or 3 weeks. Kanban is oriented to delivering items immediately as completed. Emergent date-driven items can easily be given priority in Kanban, whereas this can be achieved but is less natural in Scrum.

Subject to rapidly shifting priorities: Changing priorities is difficult on short notice in Scrum, where plans are expected to be firm for the duration of a sprint. In Kanban, it is easy to shift priorities in an intake queue; there is no pre-commitment to the next work item to be started.

Difficult to estimate: Defect flows, for example, due to the typical uncertainty within many or most items, are difficult to schedule in Scrum but easy to handle in Kanban. Kanban does not require planning based on an estimate of size or duration.

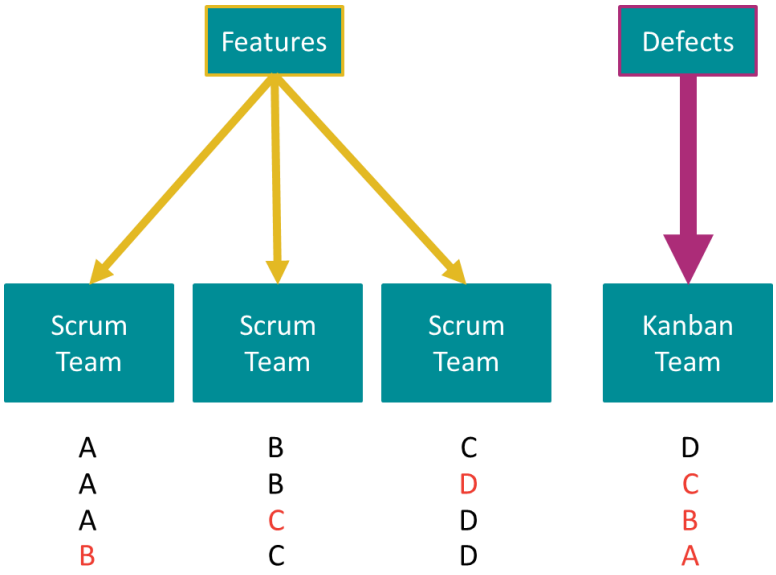
Organization change needs to be taken in small increments: Kanban is less prescriptive than Scrum. In some cases it may be more practical for an organization to begin with a Kanban approach built closely on existing practices, then gradually apply improvements as the bottlenecks in the workflow become visible.

Mixed and Blended Scrumban Designs

Many organizations have found that their needs represent a mix of the considerations reviewed above. Thankfully, the affinity between Scrum and Kanban is so close that it's quite practical to combine the two approaches, deploying them to work in concert or simply side by side.

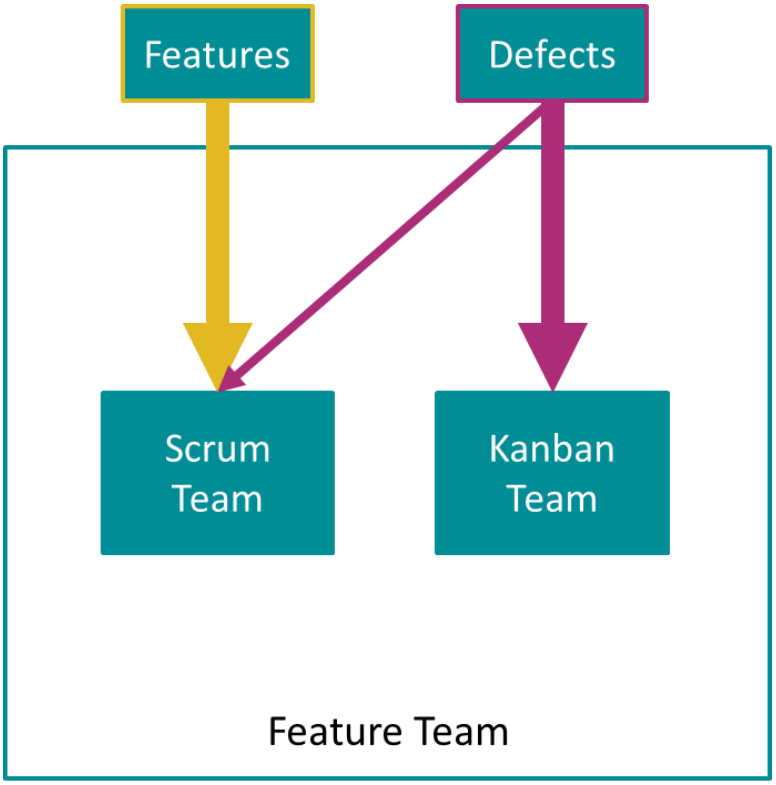
Here are a number of designs drawn from real-world examples.

Rotating Defect Teams (Telco A)



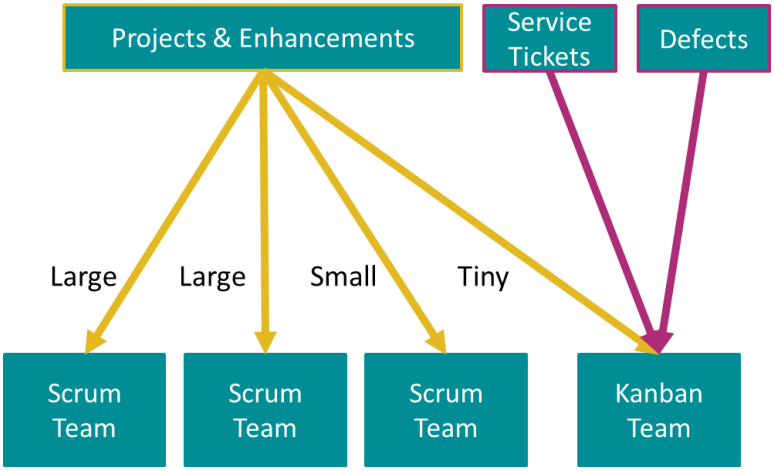
In this example, new feature flow is directed through a set of Scrum teams, while a dedicated Kanban team is tasked with handling defect flow during the endgame of a release hardening period. To avoid consigning one team to perpetual defect jail, the responsibility is rotated with successive releases.

Defect Sub-Team (Software Company A)



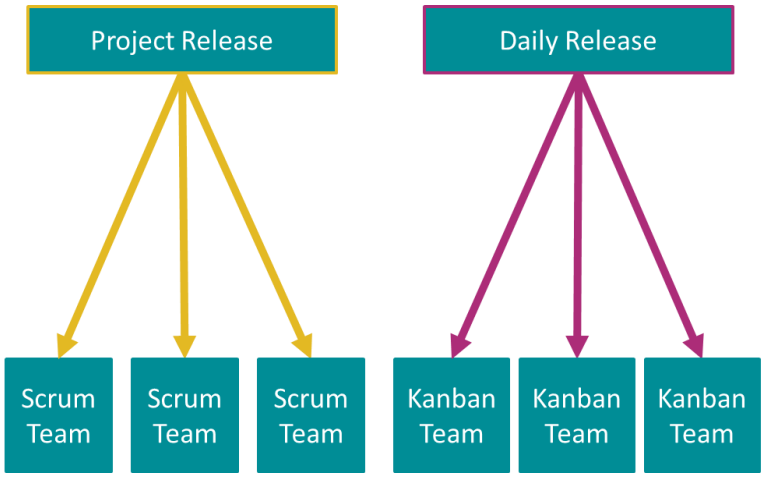
This organization has a feature team structure, with some feature teams comprising multiple Scrum teams. In this case, there is a single Scrum team handling mostly the new feature flow, with a Kanban team dedicated to handling most of the defect load.

Large and Small Items (Energy Company)



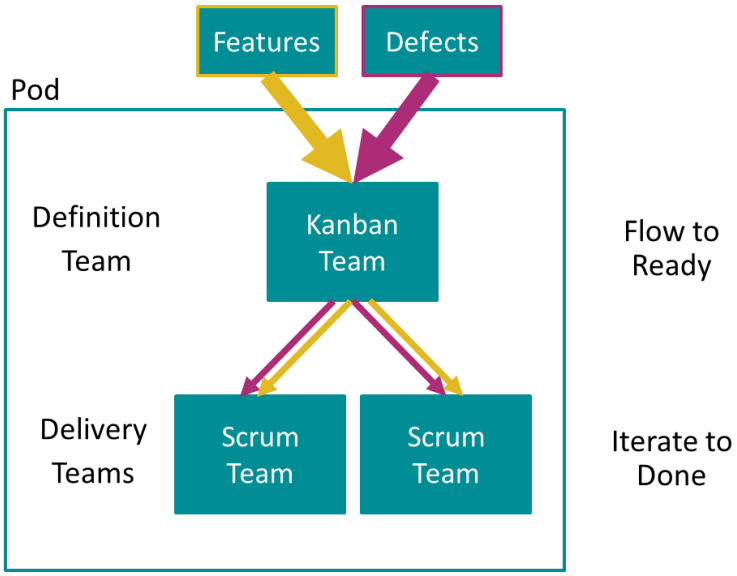
This organization serves both software and operational needs. A set of Scrum teams handles projects and enhancements, with tiny enhancements joining the flow of service tickets and defects to the Kanban team.

Project and Daily Release (Software Company B)



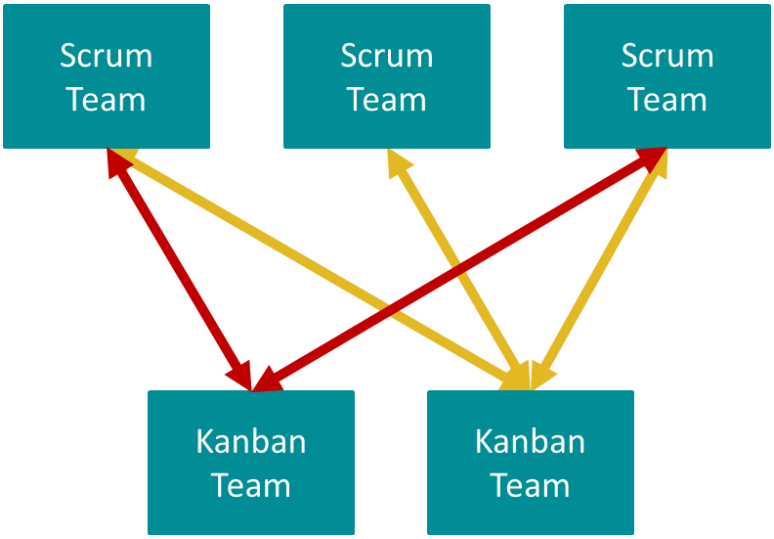
This organization supports releases on a project timescale as well as daily content changes.

Definition and Delivery Teams (Telco B)



Each Pod in this organization comprises one or more delivery teams supported by a single definition team. The definition team is an extension of the Product Owner concept that brings additional specialist expertise into that function. User stories are made ready by the definition team using a Kanban flow, then consumed by the delivery teams in a Scrum flow[5].

Specialty Service Queues



When the workflow (value stream) is not encapsulated within the Scrum team, throughput is impeded. The team will experience idle work and/or idle people while waiting for outside parties to execute on dependencies. This arrangement mitigates that situation by putting the outside parties into a Kanban workflow in which transparency and service level expectations can be better established.

Scrumban Blend

Experience has shown that Kanban teams can benefit from many of the same roles, ceremonies, artifacts, and concepts that provide benefit to Scrum teams. In the following table, elements commonly associated with (though not in all cases formally part of) Scrum and Kanban are arranged according to their approximate conceptual parallels. The Scrumban column contains recommendations for a blended approach.

Scrum	Kanban	Scrumban	Notes
Sprints	Continuous Flow	Continuous Flow	
Sprint Planning	Pull work as needed	Pull work as needed	
Sprint Review, Per-Item Demo	Per-Item Completion	Per-Item Completion, Periodic Review	Each item should be demonstrated to the Product Owner or responsible stakeholder at the time of completion. It may be advantageous to also assemble stakeholders periodically to review the cumulated work delivered in the period.
Daily Stand-Up		Daily Stand-Up	
Retrospective		Retrospective	
Product Owner		Product Owner	Product Owner may be unnecessary if items are served FIFO and limited work item clarification, stakeholder negotiation, or prioritization is required.
ScrumMaster		ScrumMaster	The ScrumMaster facilitates a team's activities of continuous improvement and helps them recognize best practices, opportunities for improvement, and self-directed course corrections. These are equally applicable to Kanban teams.
Cross-Functional Team		Cross-Functional Team	Throughput is enhanced when the team does not experience bottlenecks due to limitations in specialist expertise.
User Stories	Work Items	User Stories, Work	User Story format may be unnecessary for simple work

Scrum	Kanban	Scrumban	Notes
		Items	items.
Potentially Shippable Product Increment	Completed Item	Potentially Shippable Product Increment	Emphasizes that quality and complete testing are part of work completion, not to be added later or by someone else.
Definition of Done	Process Policies	Process Policies	
Story Point Estimates	Classes of Service, SLAs	Classes of Service, SLAs	Story Point Estimates may provide supplementary value in conjunction with Velocity as a cross-check or while statistics are accumulating
Velocity	Cycle Time or Lead Time	Cycle Time or Lead Time	
Story Board	Kanban Board	Kanban Board	
Tasks usually or often	Usually no tasks	Discretionary	Team choice; more useful for larger work items.
Product Backlog	Open & Ready Queues	Open & Ready Queues	
Sprint Backlog	Work in Process (WIP)	Work in Process (WIP)	
Priorities at Planning	Priorities by class & on demand	Priorities by class & on demand	
Burn-Down Chart	Continuous Flow Diagram, Cycle Chart	Continuous Flow Diagram, Cycle Chart	

Conclusion

Scrum and Kanban can be mixed and blended to suit your needs. There's no need to be limited to a single tool. Once you understand how these frameworks deliver value, you can be creative in designing a solution to meet your needs[6].

[1] For foundational information about Scrum, see http://www.scrumalliance.org.
[2] For foundational information about Kanban, see *Kanban* by Daniel J. Anderson (2010).
[3] For a discussion of autonomy as a motivating factor for knowledge workers, see www.ted.com/talks/dan_pink_on_motivation. For the book version, see *Drive* by Daniel Pink (2011).
[4] For more detail, see http://SoftwareTestingFundamentals.com/dimensions-of-software-quality.
[5] The concept of Flow to Ready, Iterate to Done originated with Serge Beaumont. See http://blog.xebia.com/2009/07/04/flow-to-ready-iterate-to-done/.
[6] Also see *Kanban and Scrum: making the most of both* by Henrik Kniberg and Mattias Skarin (2009), http://www.infoq.com/minibooks/kanban-scrum-minibook.