SKI: A New Agile Framework that supports DevOps, Continuous Delivery, and Lean Hypothesis Testing

Jeffrey Saltz
Syracuse University
jsaltz@syr.edu

Alex Sutherland
Scrum Inc.,
alex.sutherland@scruminc.com

Abstract

This paper explores the need for a new process framework that can effectively support DevOps and Continuous Delivery teams. It then defines a new framework, which adheres to the lean Kanban philosophy but augments Kanban by providing a structured iteration process. This new Structured Kanban Iteration (SKI) framework defines capability-based iterations (as opposed to Kanban-like no iterations or Scrum-like time-based sprints) as well as roles, meetings and artifacts. This structure enables a team to adopt a well-defined process that can be consistently used across groups and organizations. While many of SKI's concepts are similar to those in found in Scrum, SKI's capability-based iterations can support the demands of product development as well as operational support efforts, and hence, is well suited for DevOps and Continuous Delivery. SKI also supports lean hypothesis testing as well as more traditional software development teams where capability-based iterations are deemed more appropriate than time-based sprints.

1. Introduction

DevOps is the integration of development and operations, with a key goal of shortening the feedback loop and the development cycle through collaboration, automation and frequent software releases [1,2]. A related term, Continuous Delivery (CD), is a set of practices and principles to enable the release software faster and more frequently [1]. In considering the key goals and principles of DevOps and CD, we use the term DevOps/CD and adopt a definition provided by Jabbari et al. [3]:

> *"DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations (Ops), emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices"*

Hence, DevOps/CD is a development paradigm that enables continuous delivery and support via a set of well-defined processes.

DevOps/CD is a growing practice in organizations ranging from Amazon to Spotify [4]. Several studies have noted the potential benefits of DevOps/CD. For example, it enables teams to continuously track the current build state of the software, reduce integration and configuration errors, lower stress when dealing with releases, increase deployment flexibility and provides improved team collaboration [1,5]. More generally, it has been noted that DevOps/CD speeds up decision-making processes and helps teams meet the demands found in rapid changing environments [4].

One key benefit of a DevOps/CD approach is that it enables teams to rapidly test lean hypotheses [6]. For example, *The Lean Startup* [7] describes how to iteratively validate or reject a sequence of lean testable hypotheses to drive product development and innovation. Without DevOps/CD, delays in the deployment of software and in the collection of data on its usage slows down this empirical hypothesis testing process.

In practice, many teams have struggled to realize the benefits of DevOps/CD. Some issues have been technical, such as the ability to easily deploy a software release. These technical challenges are starting to be resolved via techniques such as containerization [8]. Teams have also struggled to leverage DevOps/CD due to the lack of a well-defined process framework that effectively supports DevOps/CD [4].

In fact, while some teams have found success using specific implementations of Scrum, Kanban, and other frameworks, these approaches have typically required ancillary patterns to support DevOps/CD. Because there is a lack of a broadly accepted and documented agile DevOps/CD practices that teams can turn to, each team has been forced to define their own process. In short, organizations have largely had to determine how to apply DevOps/CD by themselves [5] and many think that there is no defined process to use when using a DevOps/CD approach [9].

URI: https://hdl.handle.net/10125/64503
978-0-9981331-3-3
(CC BY-NC-ND 4.0)

This paper aims to address this issue by defining a standardized process framework that supports DevOps/CD and also naturally integrates lean hypothesis testing, and hence, can help teams realize the full benefits of an agile DevOps/CD approach. The new framework adheres to Kanban's lean principles and, at the same time, also adheres to most of the elements of the official Scrum Guide.

Specifically, this paper first provides some background on currently used agile frameworks (e.g., Scrum, Kanban) and the challenges in using these frameworks within a DevOps/CD context. Then, the principles used to define the new process are explained. Next, the new process is described and explored via an example of a team following the new framework. Finally, the conclusion, along with potential next steps, is presented.

2. Background

This section first reviews Scrum (the most popular Agile framework) including its strengths and how some have used Scrum within a DevOps/CD context, as well as reviewing the challenges that have been identified when using Scrum within a DevOps/CD context. It then reviews Kanban and describes the challenges when trying to use Kanban. This is followed by a description of Scrumban, which is an attempt to integrate Scrum and Kanban. Finally, usage trends of these different frameworks are explored.

2.1 Scrum

2.1.1 Scrum Overview: Scrum is an adaptive framework for "developing, delivering, and sustaining complex products" [10]. It divides a larger project into a series of mini-projects, called "sprints", each of which have a consistent and fixed length, typically one to four weeks long. Scrum teams have three roles: the product owner, the development team, and the scrum master. Each sprint, starts with a sprint planning meeting where the product owner explains the top items from the product backlog, which is an ordered list of product development ideas. The development team forecasts what items from the product backlog they can deliver by the end of the sprint and then makes a sprint plan to develop a product increment that includes the selected product backlog items. During a sprint, the team coordinates closely and holds daily standup meetings. At the end of each sprint, the team demonstrates the newly developed product increment to stakeholders and solicits feedback during sprint review. This increment should be potentially releasable and meet the predefined definition of done. To close a sprint, the team inspects itself and plans for how it can improve in the next sprint during the sprint retrospective. Throughout the process, the scrum master acts as a servant leader and coach to help everyone effectively implement Scrum [10].

The Scrum framework provides a broad definition of the elements of Scrum. Teams that use Scrum may incorporate a variety of best practices, documented via *patterns,* that provide additional process details, which helps to ensure a suitable specific Scrum implementation for the team's specific needs.

Scrum has become the most commonly used agile approach with over 12 million practitioners [11]. Software companies have most heavily adopted Scrum but a wide variety of companies use it for diverse purposes. For example, National Public Radio uses it to create new programming, John Deere for new machinery development, Saab for fighter jets, Team WIKISPEED for electric cars, and C.H. Robinson applies scrum for human resources [12].

2.1.2 Scrum Support for DevOps/CD: DevOps/CD practices rely on the foundations of agile and lean software development, including continuous integration practices [2]. In other words, both DevOps/CD and Scrum have an emphasis on continuous integration, testing, and delivery of working software. In particular, both emphasize small and incremental releases [3] and many also think that it is desirable to have a Scrum team "eat their own dog food" in that if they produce a defect that gets into production, that same team should fix the issue as soon as possible [13].

Thus, some contend that Scrum is an appropriate framework to implement a DevOps/CD construct. From a Scrum perspective, in order to implement DevOps/CD, the team needs to be able to account for unknown issues during sprint planning. For example, a Scrum team that is in the middle of the sprint might discover an issue that needs to be fixed, which takes precedence over the tasks that are part of the sprint.

The Scrum Patterns Group has shown, via a defined pattern, how to enable this type of interrupt driven work within a sprint by establishing a buffer for the Scrum team [13]. In short, the team should explicitly allot time for interrupts (in a buffer for unexpected work). This buffer might be, for example, 30% of the team's capacity. If work exceeds the allotment, the Sprint should be altered or aborted.

Heeager and Rose [14] also explored how to enable maintenance within the Sprint framework. They conducted a study on agile maintenance and found major problems related to missed sprint goals due to the requested, but unplanned, emergency work during the sprints. They also suggested, similar to the Scrum Patterns Group, that teams should create a buffer, but noted that it is difficult to predict the size of this buffer. Furthermore, they suggested creating a well-managed

mechanism to handle these urgent emergency customer requests.

In fact, while determining the size of a buffer might be difficult, using a buffer is now a common practice within Scrum teams that need to support operational production related issues [15].

2.1.3 Challenges of Scrum for DevOps/CD: While buffers can be used to enable the use of Scrum in a DevOps/CD context, the practice of using a buffer can also be thought of as actually having two processes, one for the Ops (interrupt) work, and the other process for the development work [15]. This is why Heeager and Rose [14] suggested the creation of a well-managed mechanism to handle this buffer work. It has also been argued that the use of a buffer reduces project transparency [16].

More broadly, Ahmad et al. [17] identified several interconnected challenges with maintenance teams using Scrum. Beyond the challenge relating to the fact that operational support tasks are difficult to predict (e.g., emergency bug fixes), similar to Mitchell [16], they note that these unplanned tasks often lack visibility across the team. In addition**,** it has also been argued that maintenance work differs from development work [14]. For example, maintenance work can be organized into sprints, but there is not necessarily any task synergy or common goal for these tasks [18]. Hence, there may be less need (or perceived need) for openness, information sharing and collective ownership if the tasks are unrelated, and there is no coherent release [19]. Therefore, it has been noted that it's reasonable to expect a number of challenges when using Scrum in a maintenance context [14, 17].

Exemplifying the perceived need for a new process, Samarawickrama and Perera [9] suggest that a new "continuous scrum" framework is required to support DevOps/CD, since they think that there is a need to modify Scrum rituals and rules to address DevOps/CD goals such as reducing the required time for a feature to be put into production [9].

2.1.4 Scrum Challenges*:* We note that the key challenge of using a sprint-based framework within a DevOps/CD context include:

Long feedback cycle – In Scrum, a list of features is implemented during a sprint. For many Scrum teams, these are released throughout or at the end of the sprint, and the team starts working on a new set of features (during the next sprint). Hence, market feedback from that first sprint is often not incorporated until the third sprint, which means, if the team has two-week sprints, a refinement from one sprint might not be available in the product for a month (the first two weeks the team is working on other backlog items, and then there is a two-week sprint, independent of how long the requested item will take). Even if the Scrum team practices CD and releases new features many times per sprint, it is often the case that measurement and analysis of the performance of these features is not assessed until the end of the current sprint (during the sprint review), or the next sprint, and because the work to be done during a sprint is unchanged once the sprint starts, iteration on a feature released on the first day of a sprint will take at least a full sprint length to occur and often longer.

Decoupling Scrum events (meetings) from item swarms - the goal of minimizing sprint duration (so that items can be released faster) implies that the sprint meetings will also occur more frequently. However, at a certain point, there is diminishing value in these ceremonies. One should think logically about the frequency needed for meetings such as a retrospective (which might be different than the duration of very short sprints).

Arbitrary sprint duration - For some teams, the length of a fixed sprint might not make sense. For example, it might be the case that sometimes it makes sense to have a sprint that lasts one day, and other times, it makes sense for a sprint to last three weeks (ex. due to how long specific bug fix or set of backlog items will take to complete). This could allow smaller logical chunks of work to be released in rapid and coherent fashion, rather than creating software in predefined sprint windows, which may not match the length of a logical group of tasks. The Scrum Guide suggests that the central focus of each sprint should be a specific Sprint Goal (a single goal that the team wants to achieve in the given sprint). Limiting the choice of Sprint Goals to things that take a specific known duration, reduces the choice set for the team which may result in non-optimal decision making.

Task estimation reliability - If the team can not accurately estimate task duration, the concept of a sprint, and what can get done within a sprint is problematic. If the team underestimates the work, they either have to work very long hours, extend the sprint or in some other way, change the definition of a sprint. If the team overestimates the duration of tasks, the team will not have a clear direction of what to do. Note that there are many reasons why a team might have unreliable task duration estimates (e.g., a bug fix task where it is not yet clear what is causing a bug, a task that is exploratory in nature, etc.).

Operational Support Challenges - Teams that have to support production code might have to stop their software development to fix an issue. This typically can't be scheduled, but when it occurs, will impact the team's ability to deliver the agreed upon sprint tasks, unless a buffer has been allocated to account for this

unscheduled work. However, the use of a buffer implies an expected consistent amount of unscheduled work, which might not be an appropriate assumption.

2.2 Kanban

2.2.1 Kanban Overview: Kanban aims to support lean thinking by focusing on maximizing value and minimizing waste in production processes. Specifically, Kanban defines a set of principles which include: visualize the workflow, limit work-in-progress, measure and manage flow, make process policies explicit, and improve collaboratively / implement feedback loops [20].

Two key strengths of Kanban are that (1) it visually represents work on a Kanban board with work items flowing across the columns (or bins) of increasing work status completion (i.e., work items are represented visually on a Kanban board, allowing all team members to see the state of every piece of work at any time), typically starting with a 'to-do' column and ending with a 'done' column, and (2) it aims to minimize work-in-progress, often with WIP limits. Minimizing WIP enables a lean approach (by focusing on reducing the time it takes to complete a task or user story) and also enables agility (since tasks are re-prioritized each time a new task starts).

Kanban proponents claim that Kanban offers improved project visibility, software quality, team motivation, communication and collaboration compared to other Agile methodologies [17]. In addition, a survey of Kanban software development practitioners reported that they perceived Kanban as easy to learn and use [21]. The respondents noted several perceived benefits for using Kanban, such as bringing visibility to work, helping to reduce work in progress, improving development flow, increasing team communication and facilitating coordination.

2.2.2 Kanban Challenges*:* While Agile practices such as Scrum have a well-defined process framework to structure work, Kanban has no such specified process framework. Hence, teams that are looking to apply Kanban report a variety of challenges. These challenges include the lack of organizational support and culture, lack of training and the misunderstanding of key concepts [21]. In fact, since Kanban does not define project roles nor any process specifics, the freedom that Kanban provides can be part of the challenge in implementing Kanban. Specifically, while this lack of process can be a strength when using Kanban (since it allows teams to implement Kanban within existing organizational practices), the lack of process definition also means that every team can implement Kanban differently. In short, a team using Kanban needs to define their own processes and artifacts.

2.2.3 Kanban Challenges: Since teams need to define their own processes and artifacts, it is not surprising that there remains a lack of cohesion and consensus around how to use Kanban. For example, Ahmad et al. [22] identified seven different implementation definitions during a literature review of Kanban. Stated another way, the fact that Kanban does not explicitly specify a process framework suggests that Kanban needs to be supported by additional practices [23]. This lack of process definition also explains why teams that use Kanban note that "Kanban requires integration with existing agile techniques, which can be complicated, expensive, and time-consuming" [22].

2.4 Scrumban as a Possible Solution

2.3.1 Scrumban Overview: Scrumban, which was introduced by Ladas [24], is a lean approach that defines a set of processes for teams that use Kanban. Ladas viewed Scrumban as a way to transition from Scrum to Kanban, and focused on using Kanban within a Scrum sprint. Many others use a different definition for Scrumban, such as Nikitina, Kajko-Mattsson & Stråle [25], who suggest that Scrumban is the implementation of some of the Scrum practices with some of the Kanban principles. Others, such as Reddy [26], leave significant freedom in the definition (e.g., use some, all or none of the Scrum artifacts and ceremonies).

This shows that there is not a single, commonly accepted, definition of Scrumban. Even looking at popular web sites such as Wikipedia and the Agile Alliance makes it clear that there is not one definition. This was highlighted by Reddy [26] who noted that "although Scrumban has evolved as a framework over the years, it has no definitive guide or definition. In fact, … several 'authoritative' sources disagree about what Scrumban actually represents".

2.3.2 Summary of Scrumban*:* While there is not a commonly agreed upon definition, most definitions of Scrumban agree that Scrumban includes the concept of a sprint. Hence, Scrumban loses Kanban's focus on continuous delivery. Most definitions also include the concept of task estimation, but many Scrumban definitions do not have specifically defined roles. All agree on using visual board, and many view the Scrumban board as being persistent across iterations.

For an example of how Scrumban has been used within a DevOps/CD context, in describing how one organization migrated from Scrum to Scrumban (to implement Continuous Deployment), it was noted that the organization migrated to Scrumban, but their version of Scrumban (which had no sprints and no and Scrum

ceremonies) was, in reality, Kanban [27]. Furthermore, it was noted that the organization realized that there was more to implementing Kanban than using a Kanban board. Specifically, the team realized that there needs to be a shared understanding of "when to move things from one column to the next" as well as "what will be done when WIP limits are reached".

2.3.3 **Scrumban Challenges***:* While there are many flavors of Scrumban, most have timeboxed sprints, which means the key Scrum challenges previously noted are applicable to Scrumban (arbitrary sprint duration, the need to decouple meetings from item swarms, the long feedback cycle, and challenges in task estimation are also applicable to Scrumban).

2.4 Agile Usage Trends

Dingsøyr & Lassenius [28] noted that there is a transition from teams wanting to use an iterative development approach via Scrum sprints to the desire for continuous deployment of new features. For example, in the latest "State of Agile Report" [29, 30], VersionOne identified Scrum as the most common agile framework, used by 56% of respondents. However, for the previous two years, it was used by 58% of the respondents, so it appears that the use of Scrum has plateaued. The survey also noted that the use of Kanban, as an agile technique, grew to 65% (in 2017) from 50% in 2016, and from 31% in 2014. The survey also noted that 8% of teams reported using Scrumban and 5% reported using Kanban (as their main process framework). In short, the use of Kanban is growing quickly, but most teams currently view Kanban as a technique (similar to the daily stand-up) that is used within Scrum projects.

This view is consistent with what has been reported by the Scrum Alliance [31]. Specifically, that the number of teams exclusively using Scrum is dramatically decreasing (from 43% in 2015 to 16% in 2017), and "Scrum with Others" is now used by 78% of the respondents, with Kanban being, by far, the most often used other framework (with 60% of the teams incorporating Kanban).

3. SKI: A New Agile Kanban Framework

> *"Scrumban is one of many possible stories about Lean transformation—we need more!"*
> Ladas, 2009

To address these challenges and usage trends, we have defined SKI, which is an agile Structured Kanban Iteration framework, leveraging some of the key concepts of Scrum and Kanban, but differently than Scrumban (which as previously noted, is more of Kanban within a Scrum Framework). The rest of this section first describes the principles driving the definition of SKI, and then describes the roles, artifacts, and events that comprise the framework.

3.1 Principles Driving SKI

In general, we view that the key concepts and benefits for a lean agile project are based on the following three key tenets:

1. Agile is intended to be a sequence of iterative experimentation and adaptation cycles.
2. The goal of such cycles should be to have an idea or experiment in mind, to build it, and then to observe its performance in the real-world use case for which it is intended, and then to analyze those observations to create the next idea or experiment.
3. Going from an initial idea, through implementation, deployment and the measurement and analysis of results should be the basis for an iteration. The completion of the empirical process should mark the end of an iteration (not a predetermined number of elapsed hours).

Teams should focus on maximizing the number of empirical processes that they can achieve in a given year, weighted by the value of each empirical process, by minimizing the cycle time of each empirical iteration. By following these tenets, teams will naturally focus on their process efficiency (i.e., focusing on trying to ensure that the time spent during an iteration goes towards work that was actually required to run the given experiment/iteration).

3.2 The Framework

SKI teams use a visual board and focus on working on a specific item or collection of items during an iteration that is task-based, not time boxed. Thus, an iteration more closely aligns with the lean concept of pulling tasks, in a prioritized manner, when the team has capacity, and each iteration may be viewed as validating or rejecting a specific lean hypothesis.

Specifically, an iteration is defined by the following three steps:

1. ***Create***: A thing or set of things that will be created and put into use with a hypothesis about what will happen.
2. ***Observe***: A set of observable outcomes of that use that will be measured (and any work that is needed to facilitate that measurement).
3. ***Analyze***: Analyzing those observables and creating a plan for the next iteration

The create, observe, analyze process is similar to "build, measure, learn" from *The Lean Startup* [7], but with an emphasis on ensuring that the work that is required for data collection and data analysis is directly incorporated into the team's tasks for a given iteration. Note that we use the word "analyze" rather than "learn" to indicate that the team should dedicate time specifically to active quantitative or qualitative analysis of data according to a statistically sound methodology. The results of this analysis are specific to the performance of the creation, according to the selected observables, which is different than other learning that might occur during an iteration (e.g., learning how to execute a task more efficiently, identification of impediments, etc).

Furthermore, SKI has a well-defined set of roles, artifacts, and events, which are explained below.

3.2.1 Roles: Similar to Scrum, each SKI team is a group of three to nine people, one of whom is the ***product owner***, and one of whom is the ***process master***. As in Scrum, the product owner in SKI is the empowered central point of product leadership – the person who decides which features and functionality to build, the order in which to build them, and what aspects of them to observe and analyze. The process master (SKI master) acts as a coach, facilitator, impediment remover as well as helping everyone involved understand and embrace the SKI values, principles, and practices to aid the organization obtain exceptional results from applying SKI. Both the ***product owner*** and the ***process master*** are part of the ***SKI Team*** and may contribute to creating, observing and analyzing throughout an iteration. Finally, the team should be comprised of a cross-functional collection of people that have all the skills needed to design, build, test and deploy the desired product. The team self-organizes to determine the best way to accomplish the goal defined by the product owner.

3.2.2 Artifacts: A ***Product Backlog Item (PBI)*** may take a variety of forms such as "user stories" or "testable hypotheses" as popularized by XP and Lean. Each PBI should include at least one thing to create, one thing to observe and one thing to analyze. The ***Product Backlog*** is a prioritized list of PBIs (i.e., work to be done). The product owner, with input from the stakeholders and the team, is responsible for maintaining the product backlog, which evolves and changes throughout the project. The ***Task Board*** is a visual representation of the work items currently in progress. For any item that is in progress (i.e., being worked on by the team), the tasks for that item are displayed on the board (items not yet started are on the product backlog). The board has several columns (at a minimum, 'to do', 'in progress', 'done') and each task flows across the board, thus visually showing work being done within the team. As with Kanban, to facilitate task throughput, each team defines a maximum number of tasks within a single column.

3.2.3 Events: An ***Iteration*** is a collection of one or more product backlog items that are combined into a single testable experiment, the outcome of which must have business value derived either from the thing that is created, the information gained through observation and analysis or both. Each iteration should aim to be a minimally viable set of work that can deliver value and allows the given lean hypothesis to be tested, and should not last more than one month, but can be as short as the team wants (e.g., one day). The team typically breaks this set of one or more PBIs into several tasks that the team collectively strives to complete as soon as possible. These tasks are placed on the board, in the 'to do' column. The current status of these items is always visually represented on the task board and the iteration is completed when all the tasks for that item are in the 'done' column.

The ***Daily Meeting*** occurs each day, when the team meets for a 15-minute inspect-and-adapt activity. An important goal of this meeting is to help a self-organizing team better manage the flow of its work (ex. helping a team member get past an issue). Just as with Scrum Standups, a common approach for conducting this meeting is for team members to share with each other what they did yesterday, what they are planning to do today, and any obstacles they are facing.

The ***Iteration Review*** occurs on a regular and repeating basis, and is scheduled by the product owner. Reviews might be weekly and are calendar based to account for the fact that there might be several iterations per week, and there would be diminishing returns if iteration reviews occurred on a daily (or more frequently) basis. They would also be logistically difficult to schedule if they were needed on an ad hoc basis.

The review is intended to foster conversation about completed functionality and the observations and analysis that the team has generated regarding the performance of the completed iteration(s). Participants include the team, stakeholders, customers, and anyone else interested in the outcome of the project. A successful review results in bidirectional information flow. The people who aren't on the team get to sync up on the project effort, the observed product performance, and the team's analysis of that performance. At the same time, the team can get suggestions from the other attendees for potential features, metrics and experiments for future iterations. Furthermore, during this meeting, the group discusses the prioritization of the backlog items (since, for example, the insights gained might suggest a change in item priority or the creation of new

items). At the end of the review, the tasks on the board relating to the discussed and now completed item(s) are archived.

Note that in addition to the SKI team working on one or more iterations, the team also spends time ***grooming and prioritizing the PBI***, which is the activity of creating, refining and prioritizing potential iteration items. While the product owner owns the prioritization process, the other members of the team typically budget 5% to 10% of their total capacity to assist the product owner with product backlog grooming (e.g., breaking an item into two smaller, but still useful, items). As part of the grooming process, the team defines a relative unit of measure, decided upon by the team, to provide relative an estimate of the effort for completing different items. This effort estimation, which could, for example, be a T-Shirt sized view of the work to do (large, medium, small), or be a number representing relative effort, is used to help prioritize backlog items, but not define what is part of an iteration.

Product backlog Selection occurs when the team has capacity to start a new iteration (e.g., when an iteration has completed, or when an iteration observation does not require full-time focus). The team reviews the prioritized backlog items (that have been updated via grooming) and selects the top backlog item(s) that will now be the team's focus. Note that since the iteration is capability-based, and is the minimally viable set of items that can deliver value, the item estimation is used to help prioritize items, not determine how many items should be included in an iteration (e.g., if two items deliver the same value but one is deemed a "small" effort and one is a "large" effort, the team might select the smaller level of effort item). Combining multiple items into a single iteration is generally only desirable in the case that the associated hypothesis or observable data overlap.

Finally, the ***Retrospective*** occurs at regular intervals (ex. once a month) and is a time to inspect and adapt the process. In the spirit of continuous improvement, the team comes together to discuss what is and is not working with the current process and associated technical practices. The goal is to help a good SKI team become great. At the end of a retrospective, the team should have identified and committed to a practical number of process improvement actions that will be undertaken by the team going forward.

3.2.4 Comparing SKI to Scrum and Kanban: Table 1 provides a summary of how SKI compares to Scrum, Scrumban and Kanban. The table focuses on the key attributes of the SKI framework. Specifically, in reviewing the table, one can see that SKI can be viewed as an instance of Kanban. With respect to Scrum, SKI can be viewed as being similar in many aspects, but SKI differs from Scrum in the use of capability-based iterations, the flexibility to not have to accurately estimate task duration and having key meetings (ex. retrospective) be calendar based, not sprint-based.

	SKI	Scrum	Scrumban*	Kanban
Iteration	Capability / Item-based	Time-based	Time-based	No iteration
Unplanned / Ops work supported via	New task on board	Buffers	Buffers	New task on board
Iteration & Retrospective reviews	Time-based	After each sprint	After each sprint	Not defined
Iteration coordination	Kanban flow	Not defined	Kanban flow	Kanban flow
Task Estimation Usage	Only for PBI prioritization	PBI priority & What fits into a sprint	PBI priority & What fits into a sprint	No Task Estimation
Use of PBI	Yes	Yes	Yes	Yes
Backlog selection	When there is capacity (to start new iteration)	When sprint completes	When sprint completes	When there is capacity
Daily Standup	Yes	Yes	Yes	Not defined
Roles	Proc Master, SKI Team member, PO	Proc Master, Dev Team, PO	Proc Master, Dev Team, PO	None Defined

**Based on the most commonly used definition of Scrumban*

Table 1: Comparing SKI, Scrum, Scumban and Kanban

4. An Example

4.1 Background

A team is in charge of the website documentation, tutorials and new user experience for a complex web application. The application has a one-week free trial after which users must either purchase the full version of the software or cease to use it. The teams core responsibility is to maximize the frequency with which a user who starts a free trial of the software goes on to purchase the full version.

The *product owner* is responsible for assessing the business value of each potential experiment, considering both the immediately created value (e.g., the increase in sales) as well as the potential value of what the team might learn when analyzing their results.

The product owner collaborates with the rest of the team to determine what needs to be done to create the desired features, what data should specifically be observed and analyzed, and what is required to collect and analyze the data to create ready Product Back Items (PBIs). These PBIs are estimated, with respect to how much effort is required to finish each item. During the *product backlog selection* discussion, led by their *SKI master*, the team collectively might combine, separate, simplify, or alter their product backlog items to come up with a specific experiment to run (i.e. an iteration).

4.2 Example Iteration

An *item* on their *product backlog* is to translate their FAQ, which currently is only provided in English, into additional languages, because they have noticed that their conversion rate in non-English speaking European countries is substantially lower than it is in the UK. Another item on their backlog is to create a guided walkthrough that is triggered when a new user starts using the application, as they notice that a large percentage of their users launch the application once and never use it again and they hypothesize that they lose a substantial number of new users who are confused by the application the first time they use it.

With respect to this iteration, the team might elect to create the automated guided walkthrough, but to make it available both in English and in German as they realize that translating the walkthrough into just one additional language is a small additional effort compared to translating the entire FAQ and might give insight into the value of translation. They would then measure the increase in how many users launched the application more than once, and how many converted to a sale of the full version of the application, among users from both the UK and Germany, and compare that to existing baseline rates.

Analyzing the resulting data might inform the team of both the potential future value of adding additional language support to their documentation while also helping them understand if new user confusion results in a significant loss of sales or if there are other issues (e.g., once a user tries the application they often feel that the product doesn't meet their needs).

Once the elements to create, analyze and observe are fully fleshed out, the team begins their *iteration*. They would take the larger experiment and break it into specific tasks which would flow through the *task board*. This team also had recurring regular responsibilities that are required to "keep the lights on", as well as bug fixes that were also required as needed. The team adds the "keep the lights on" items to the task board as needed, and adds the "bug fix" items if/when bugs are identified.

In this situation, an example set of tasks might be "Select which features the automated walkthrough will highlight and write text copy for each step in English" or "Send the English text to our German translator" or "Create code that will allow us to easily highlight a specific element of the interface and show text in a bubble near it with the ability to step forwards and backwards through a list of walkthrough steps". The tasks will also include the steps required for the observation and analysis, such as "randomly show the new guided walkthrough to 50% of new users and track how many users in each sample use the application more than once, and how many buy the full version within a week, separated by country" and "Run a statistical test on the resulting data and assess the impact of the guided walkthrough overall and the German language specific version of it."

Each day, the team has their *daily standup* to identify issues and roadblocks. In this example, it might take 7-10 days to finish observing and analyzing the data as the team would want to compare conversion rates among people who had used the automated walkthrough on their first use of the product and whose one-week free trial had ended prior to drawing conclusions.

Note that, in this situation, the team may have a lighter workload during the observe and analyze portion of the iteration. During this time, the team might work on "keep the lights on" tasks or perform *grooming* of their product backlog. In addition, the team might start their next iteration and work on concurrently while observing and analyzing the data from the current iteration. Having the next iteration start while still in the final phases of a different iteration is similar to a more advanced pipeline of sprints found in type C Scrum, described by Sutherland [32], but not commonly used by Scrum teams.

4.3 After the Iteration has Completed

Once the data has been analyzed, the *grooming and prioritization* is done before the next iteration starts. The team also discusses their findings with their stakeholders at their next scheduled *iteration review*.

To continue our example, if the team found that the conversion rate in their German user segment was significantly higher than expected, they might prioritize a smaller iteration where they would add support for 3 additional languages. If they found that their experiment resulted in no significant increase in sales conversions among either the German or the English demographic, then perhaps they would look to create an experiment to see if a specific missing feature was causing users to abandon the product after a single use. As these tasks were already on the backlog, the start of this new iteration (i.e., starting their next experiment) would not need to wait for the team's next *iteration review* but rather, the priority of these items would be adjusted via the team's *grooming and prioritization* effort. However, the iteration review might uncover additional items for the product backlog, such as potentially additional guidance for more advanced features of the application.

Finally, the team has a monthly *Retrospective* to discuss what is and is not working with the current process and associated technical practices, and explore how to improve the team's process and results.

5. Discussion

SKI can be viewed as an instantiation of Scrum that is mostly consistent with the official Scrum Guide, with a few notable exceptions. The most important exception is that the Scrum Guide requires all sprints to be of equal length in time. However, iterations in SKI vary in duration, so as to allow a logical increment of work to be done in one iteration. The other notable exception is that retrospectives and item reviews are not done at the end of every iteration, but rather, on a frequency the team deems appropriate.

In addition, SKI enables teams to achieve continuous delivery by providing a structure for how the team should coordinate operation, development, and maintenance tasks without the reliance on an estimated buffer. Specifically, SKI enables "keep the lights on" tasks to be a set of repeating tasks that go on the team's board and "bug fix" tasks, if deemed an emergency, go straight on the team's board as their highest priority task (perhaps as a new iteration). If the bug fix is not urgent (as deemed by the product owner), then that code fix goes on the product backlog and prioritized as appropriate during grooming and prioritization.

Furthermore, in many Scrum implementations, observing, analyzing and reacting to market feedback is solely the responsibility of the product owner. This part of the product owner's job largely falls outside of the codified process. Collecting and analyzing well-chosen data and drawing appropriate conclusions is a crucial part of the empirical process. By building these steps directly into the core workflow and ensuring that the entire team is involved in that the process, SKI will help teams make better data-driven decisions.

Similarly, SKI adheres to Kanban (e.g., there is a Kanban board, teams need to limit WIP, and work items flow across the board). In fact, it is the Kanban philosophy and artifacts that enables the coordination and integration of interrupt requests (ex. bug fixes) with a planned enhancement iteration. However, the framework provides more structure than defined by Kanban (ex. roles, meeting and artifacts). Having a more clearly defined process, which leverages agile best practices, enables teams to implement the lean process in a more consistent and repeatable manner.

5.1 Potential Metrics

Since this framework implements Kanban, many Kanban metrics are appropriate for SKI. In addition, calculating velocity (which is the key Scrum metric) is also feasible. Hence, potential metrics include:

- *Top Item time*: How much the team is being interrupted / working on non-priority items.
- *Lead time*: The total time it takes for an iteration to complete.
- *Cycle item time*: How long it takes a work item to complete after the item work is started.
- *Cycle iteration time*: How long it takes an iteration to complete after the item work is started.
- *Throughput*: The number of tasks (or iterations) processed per time unit (e.g., per week).
- *Velocity:* The number of points completed per unit time (e.g., per week).

5.2 Conclusion

This paper describes SKI, a new agile Structured Kanban Iteration-based framework and explains how the framework can be used within a DevOps/CD context. While leveraging key aspects of Scrum and Kanban, the framework provides several advantages as compared to Scrum and Kanban.

As compared to Scrum, SKI defines an iteration that is capability-focused (not time-based) so as to provide a team the ability to execute small logical iterations as well as supporting unplanned operational support. While SKI has these key differences, as compared to Scrum, SKI's artifacts and time-based meetings enables SKI to be easily integrated within organizations that use Scrum.

As compared to Kanban, this framework provides clear guidance on roles, artifacts and events, which enables teams to more easily and reliably achieve the benefits of Kanban.

While this paper focused on the use of SKI within a DevOps/CD context, the framework could be appropriate in other contexts, such as when a team is using Kanban but wants some a more structured framework or when the team thinks that capability-based iterations are more appropriate than time-based iterations. More generally, SKI could be appropriate when the project:

1. Has the ability to rapidly release iterations and observe that release in use
2. Faces a significant degree of uncertainty in what they need to build or how the market/client will react to an iteration
3. Can dedicate a significant amount of their effort to new product development

Finally, future work is planned to explore and validate the effectiveness of SKI. The validation of SKI will require surveys and case studies of teams using SKI. For example, future work will document real world usage of SKI within a DevOps/CD context as well as other contexts, such as within data science projects or more traditional software teams, where capability-based iterations are deemed more appropriate than time-based sprints. Furthermore, future research will also explore

the metrics that were proposed in this paper and evaluate how they can be leveraged to compare SKI with other agile approaches such as Scrum and Kanban, with a goal of understanding when SKI is more appropriate than these other frameworks.

References

[1] Humble, J. and Farley, D., (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1st ed. Boston: Addison-Wesley.

[2] Lwakatare, L. E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H. H., Bosch, J., and Oivo, M. 2016. "Towards DevOps in the Embedded Systems Domain: Why Is It so Hard?," in System Sciences (HICSS), 2016 49th Hawaii International Conference On, IEEE, pp. 5437–5446.

[3] Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016, May). What is devops?: A systematic mapping study on definitions and practices. In Proceedings of the Scientific Workshop Proceedings of XP2016 (p. 12). ACM.

[4] Wiedemann, A. 2018. IT Governance Mechanisms for DevOps Teams-How Incumbent Companies Achieve Competitive Advantages. In *Proceedings of the 51st Hawaii International Conference on System Sciences*.

[5] Nybom, K., Smeds, J., and Porres, I. (2016). On the Impact of Mixing Responsibilities Between Devs and Ops, in *International Conference on Agile Software Development*.

[6] Kevic, B. Murphy, L. Williams and J. Beckmann, Characterizing Experimentation in Continuous Deployment: A Case Study on Bing, *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*.

[7] Ries, E. (2011). *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. NY: Crown Business.

[8] Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2017). Cloud container technologies: a state-of-the-art review. *IEEE Transactions on Cloud Computing*.

[9] Samarawickrama, S. S., & Perera, I. (2017). Continuous scrum: A framework to enhance scrum with DevOps. In 17th International Conference on Advances in ICT for Emerging Regions (ICTer) (pp. 1-7). IEEE.

[10] Sutherland, J., & Schwaber, K. (2017, November). *The Scrum Guide*. Retrieved from scrumguides.org: www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

[11] Scrum Alliance. (2017). *The State of Scrum Report 2017 Edition*.

[12] Rigby, D. K., Sutherland, J., & Takeuchi, H. (2016, May). *Embracing Agile*. Retrieved from Harvard Business Review: https://hbr.org/2016/05/embracing-agile

[13] ScrumPlop.org. (2018). Published Patterns:. https://sites.google.com/a/scrumplop.org/published-patterns/product-organization-pattern-language/illegitimus-non-interruptus

[14] Heeager, L. T., & Rose, J. (2015). Optimising agile development practices for the maintenance operation: nine heuristics. Empirical Software Engineering, 20(6).

[15] Reddit, 2019. Sprint buffer: Is it a good practice? https://www.reddit.com/r/scrum/comments/b0fomi/sprint_buffer_is_it_a_good_practice/

[16] Mitchel, J. 2019. Expedite! Handling Unplanned Work in Scrum, www.scrum.org/resources/blog/expedite-Humhandling-unplanned-work-scrum

[17] Ahmad, M. O., Kuvaja, P., Oivo, M., & Markkula, J. (2016). Transition of software maintenance teams from Scrum to Kanban. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (pp. 5427-5436).

[18] Poole, C., & Huisman, J. W. (2001). Using extreme programming in a maintenance environment. *IEEE Software*, *18*(6), 42-50.

[19] Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*

[20] David. J, Anderson, "Kanban: Successful Evolutionary Change for Your Technology Business". Sequim, WA: Blue Hole Press, 2010.

[21] Ahmad, M. O., Markkula, J., & Oivo, M. (2016, May). Insights into the Perceived Benefits of Kanban in Software Companies: Practitioners' Views. In *International Conference on Agile Software Development* (pp. 156-168).

[22] Ahmad, M. O., Dennehy, D., Conboy, K., & Oivo, M. (2018). Kanban in software engineering: A systematic mapping study. Journal of Systems and Software, 137.

[23] Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P., & Abrahamsson, P. (2011). On the impact of kanban on software project work: An empirical case study investigation. In *IEEE International Conference on Engineering of Complex Computer Systems*

[24] Ladas, C. (2009). *Scrumban-essays on kanban systems for lean software development*. Lulu.com.

[25] Nikitina, N., Kajko-Mattsson, M., & Stråle, M. (2012). From scrum to scrumban: A case study of a process transition. In *2012 International Conference on Software and System Process (ICSSP)* (pp. 140-149). IEEE.

[26] Reddy, A. (2015). *The Scrumban [r] evolution: getting the most out of Agile, Scrum, and lean Kanban*. Addison-Wesley Professional.

[27] Neely, S., & Stolt, S. (2013). Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *2013 Agile Conference* (pp. 121-128). IEEE.

[28] Dingsøyr, T., & Lassenius, C. (2016). Emerging themes in agile software development: Introduction to the special section on continuous value delivery. Information and Software Technology, 77, 56-60.

[29] VersionOne Inc. 2018. The 12th Annual State of Agile Report. Technical Report. VersionOne Inc. https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report

[30] VersionOne, Inc. 2016. The 10th Annual State of Agile Survey. Technical Report. VersionOne Inc. http://www.agile247.pl/wp-content/uploads/2016/04/VersionOne-10th-Annual-State-of-Agile-Report.pdf

[31] Scrum Alliance. 2018. The State of Scrum Report, http://info.scrumalliance.org/State-of-Scrum-2017-18.html

[32] J. Sutherland, Future of scrum: parallel pipelining of sprints in complex projects, *Agile Development Conference (ADC'05)*, USA, 2005, pp. 90-99.