Abstract—Agile Development is used for many problems, often with different priorities and challenges. However, generalized engineering methodologies often overlook the particularities of a project. To solve this problem, we have looked at ways engineers have modified development methodologies for a particular focus, and created a generalized framework for leveraging Kanban towards focused improvements. The result is a parallel iterative board that tracks and visualizes progress towards a focus, which we have applied to security, sustainability, and high performance as examples. Through use of this system, software projects can be more focused and directed towards their goals.

I. INTRODUCTION

Agile Methodologies are useful for supporting productive development in Software Engineering projects. Some projects, however, have specific focuses that are of high priority for the desired software. In their basic forms, Agile Methodologies may neglect focuses. In these cases, it is useful for the project’s success that a focus be leveraged in the development process itself. By looking at a successful example of security-focused Scrum, this paper attempts to apply the same strategy to Kanban and generalize it for any possible focus.

II. SCRUM + SECURITY: SCRUM LEVERAGED FOR SECURITY

One successful example of an Agile methodology being modified for a specific focus is the leveraging of Scrum towards Security. Security is a major concern for certain software projects, such as web services [1]. However, Scrum, the most popular Agile methodology in the industry, is often criticized for neglecting security analysis and design [1]. In response, researchers at UPM, one of Malaysia’s leading research universities, created a modified Scrum that incorporated secure analysis and design into the Agile process [1]. They called this secure version of Scrum "S-Scrum"; its goal: to improve security of web service projects using Scrum [1]. S-Scrum adds a security analysis and security design step to the Scrum work-flow (see: Figure 1). These steps are called “Spikes” in accordance with Scrum terminology. Prior to the start of each sprint, these steps are completed, and the backlog is modified to reflect needed security changes. In this way, the Scrum process is leveraged towards its focus, and security is improved.

This idea of a Security-leveraged Scrum was further refined and evaluated by researchers at the Federal Institute of Education, Science and Technology of Sao Paulo [2] and the Munich IT Security Research Group [3]. The Sao Paulo research team’s revision, "ScrumS", placed the security analysis and design inside of the sprint cycle. As tasks are moved from the Product Backlog to the Sprint Backlog, security-related tasks are extracted from ordinary development stories [2]. These “Security User Stories” are completed in a parallel iterative loop within the same sprint (see Figure 2). As Security User Stories are completed, prescribed changes are fed back into the Sprint Backlog, where they are implemented with the rest of the sprint. By integrating security analysis and design into sprints, ScrumS does not slow down the time between iterations. Thus, it better leverages security within the context of Scrum’s iterative process.

The researchers performed a case study using ScrumS on a security-focused surveying project, and found the resulting software was more secure than previous Scrum projects [2].
Fig. 1.
The S-Scrum work flow. By adding steps for security analysis and design before the start of each sprint, S-Scrum makes security concerns unavoidable and improves the security of Scrum. The steps are called "Spikes" in accordance with Scrum terminology. After the analysis step, the backlog is modified in accordance with identified security issues [1].

Munich IT Security Research Group revised S-Scrum by adding a connection between security tasks and related user stories [3]. They called security tasks "S-Tags" (see: Figure 3). S-Tags are extracted from user stories like Security User Stories, but remain connected to the original story through "S-Marks", allowing developers to track security-related concerns for each user story. User stories can have multiple S-Tags, and vice versa [3]. S-Tags are placed into the backlog and moved through the development process like ordinary stories. However, if a user story is added to a sprint, its related S-Tags must be added as well, keeping security a constant priority [3]. S-Tags improve S-Scrum by improving the visualization and traceability of security within the context of Scrum's user stories. The Munich IT Security Research Group also performed an evaluation of their revised S-Scrum with a team of sixteen developers, and found that security of developed software was improved [3].

Using the lessons from these versions of a security-focused Scrum, this paper attempts to translate them to Kanban and create a generalized framework for software focuses.

III. KANBAN + X

A. Translation: Kanban + Security

To create a generalized framework for focused improvements, we will first translate the same security improvements to Kanban. Following the same steps as the S-Scrum researchers, we need to:

1) Incorporate steps for security analysis and design into the Kanban work flow [1]
2) Place these steps into a parallel iterative cycle to be completed alongside development [2]
3) Visualize and track security concerns in "S-Tags" tied back to the original user stories [3]

The results of these Kanban modifications can be seen in Figure 4. We call this security-leveraged version Kanban + Security. First, we added an additional kanban board, called "Security" to act as the parallel iterative cycle found in ScrumS [2]. This board acts like an ordinary Kanban board. Security tasks are still moved as needed to reduce waste, and contribute to the limit on tasks “in progress” to keep down inefficiency from switching tasks [4]. As in ScrumS, security tasks are extracted from user stories and put into the parallel iterative loop [2]. As security tasks are moved through “doing”, the analysis and design steps described in S-Scrum are completed [1]. Needed changes are fed into the beginning of the development board, like in ScrumS [2], and are implemented normally. The security board’s tasks are S-Tags, which are unchanged from their Scrum versions [3], increasing the visibility and traceability of security concerns.

In this way, we have translated the security-focus of S-Scrum and its modifications to Kanban. Since it uses the same concepts, we expect similar benefits to the security of projects.

B. Generalization

Incorporating a specific focus does not only apply to security, but can be generalized to allow any focus on software
Fig. 2. The ScrumS work flow. ScrumS improves S-Scrum by incorporating security analysis and design into Scrum’s Sprint cycles. Security-related tasks are extracted from user stories and completed in an iterative process running parallel to primary development. Results are passed on to the sprint backlog for appropriate security modifications. [2]

Fig. 3. S-Tags. S-Tags function as tasks representing a security concern of a user story. S-Tags are connected to their related user stories through S-Marks, allowing a User Stories security concerns to be tracked. User stories can have multiple S-Tags, and a single S-Tag may be related to several user stories. Once created, S-Tags are put into the backlog and completed normally. If a user story is added to a sprint, its related S-Tags must be added as well. [3].

projects. This generalization is simple, and the results can be seen in Figure 5. We call this focus-leveraged modification to Kanban Kanban + X, with "X" being whatever focus the project is being leveraged towards.

Kanban + X works mostly the same way as Kanban + Security. Focus-related tasks are extracted from the original user story and tracked on an additional Kanban board, functioning as a parallel iterative loop. Here, focus-related tasks are analyzed; Changes are designed based on said analysis and fed back into the original board. Most differences between Kanban + Security and Kanban + X are minor. References to security, such as “Security” board, “S-Tag”, and “S-Marks” are replaced with "X" board, "X-Tags", and "X-Marks", with "X" standing-in for whatever focus Kanban is being leveraged towards.

The largest change for Kanban + X is the addition of "principles" (See: Figure 5). Here "principles" refers to the set of good practices, values, and quality measures which make up a specific focus, as agreed upon by the team. For example, in security, the "principles" would be the concepts of risk and vulnerabilities [2]. Principles increase the applicability of Kanban + X because any focus can be created by identifying the principles that define it. The principles are displayed and continuously reviewed and revised by the team to improve the focus-leveraging over time. During development, principles are referenced during creation of X-Tags, and the analysis and design of focus-required changes, thereby keeping X board progress within the agreed-upon definition of the focus. Every X-Tag is connected to one or more principles, in much the way Tasks are connected to X-Tags with X-Marks (See: Figure 5). This way, each X-Tag has a visible indication of how that task supports the focus.

Furthermore, Kanban + X allows for the leveraging of multiple focuses at once (See: Figure 4). This is done by stacking one X board on top of another, each tracking progress towards a particular focus. Since X boards are parallel iterative loops, they function independently. X-Tags for each focus are extracted from user stories using each focus’s principles. X-Tags are individually analyzed on their respective boards, and necessary changes from each focus are fed into the development board together. Obviously, leveraging for multiple focuses will also require additional balancing from the development team, but Kanban + X at least supports incorporation of multiple focuses for complex project.

C. Examples:

To demonstrate the use of Kanban + X, we will apply the framework for leveraging towards Sustainability and Performance.

1) Kanban + Sustainability: We define sustainability as the ability of a software project to be developed and supported for its intended lifespan [5]. This focus is important for large or complex project, which commonly fail due to non-sustainable practices [5], buildup of technical debt [6] and loss of knowledgeable team-members [7].

To apply Kanban + X to sustainability, we first replace generalities. The X board, X-Tags, and X-Marks are all renamed for sustainability and used for the parallel iterative loop shown in Figure 5. Sustainability-Tags are extracted from user stories and completed based on the agreed-upon principles for sustainable development. We suggest three such principles based on published research on sustainable development:
The Kanban+Security work-flow: a translation of S-Scrum concepts into Kanban. First, a second board, Security, is added as a parallel iterative cycle like in ScrumS \cite{2}. Security related tasks, called S-Tags \cite{3} are extracted from user stories and placed on the Security board. Tasks are connected to one or more S-Tags through S-Marks for traceability \cite{3}. When tasks are started, their related S-Tags are started as well \cite{3}, contributing to the limit on in-progress tasks. As S-Tags are completed, security concerns are analyzed and needed changes are put into the beginning of the development board \cite{1}.

The Kanban + X framework work-flow. As a generalized version of Kanban+Security (See: Figure 4), the structure, appearance, and workflow are almost identical. Rather than Security or S-Tags, the second "X" board and "X-Tags" track whatever focus the project is being leveraged towards. X-Tags are also connected to one more principles, which are set by the development team and define the focus.
Using Kanban+X with multiple focuses. Each focus has an X board which serves as a parallel iterative loop (See Figure: 5). The X-Marks attached to Tasks may connect to any X-Tag from any board. X-Tags for each focus are extracted from the original user story according to principles for each focus, as set by the team. Each of X-Tag is analyzed, and changes are designed according to each focus’s principles. Needed changes from each focus are fed into the development board together.

1) Team Code Ownership.
   Code should be understood and contributable by all team members. This reduces risk of knowledge loss when developers leave, which can be devastating to sustainability [7]. Furthermore, strong feelings of code ownership has been shown to increase quality and modifiability of the code [8]. Sustainability-Tags related to Team Code Ownership may include knowledge sharing [7], requiring members to experience unfamiliar areas of code [9], verifying that code meets members’ quality standards [7], and reviewing the team for cohesion [7].

2) Manage Technical Debt.
   Technical debt is a trade-off: a short-term solution in exchange for needed refinement later. While useful, excessive debt can overwhelm a project [6]. This principle refers to making good debt decisions, managing accrued debt, and eliminating unnecessary or useless debt. This principle’s goal is to reduce technical debt, and make the remaining debt more useful. Related Sustainability-Tags may include documenting and tracking accrued debt, evaluating the origin and value of a debt decision, estimating debt repay-ability, and removing excess debt [6].

3) Preventative Maintenance
   This principle refers to regularly pausing feature-production to catch up on overlooked or maintenance work [5]. "Maintenance" includes many software engineering practices, such as automated tests, documentation, and refactoring, which improve long-term productivity but are often overlooked in favor of features [5]. Related Sustainability-Tags may include writing and updating automatic tests, writing useful documentation [10], removing dependencies [5], and fixing low-quality code.

By integrating these principles into the creation and analysis of Sustainability-Tags, changes for improved sustainability can be fed into the development cycle of the project, increasing sustainability.

2) Kanban + Performance: By "Performance", we refer to High Performance Computing, which is software’s ability to complete complex programs efficiently, reliably, and quickly using multiprocessing. Performance is especially important for scientific research involving complex, highly computational simulations [11]. Performance is challenging because it comes at the expense of other quality measures, such as readability, portability, etc [12]. Because of the trade off, and research typically prioritizing results over software quality, most software
engineering tools and practices are not used in HPC software development [11]. Therefore, a development methodology leveraged towards Performance is desirable within the field [13].

We apply Kanban + X to performance much like Kanban + Sustainability. The "X" generalities are replaced with references to performance. Performance-Tags are extracted from user stories and completed in the parallel iterative loop of the Performance Board (see: Figure 5). Performance’s unique challenges are handled through agreed-upon principles, used for extracting and completing performance-tags. We have created three such principles based on the scientific goals of HPC and the unique trade-off of performance:

1) Resource Management
This principle refers to the primary action of HPC: the reduction of run time through efficient use of a computer’s limited resources. This includes memory allocation, hardware optimization, and multiprocessing [14]. Through this principle, tasks are analyzed for their efficiency, and changes to directly improve code performance are found. Related Performance-Tags may include documenting performance results, analyzing code for inefficiency, and optimizing code for specific hardware.

2) Purpose
This principle handles the trade off between performance and other quality measures. Its goal, in the style of Kanban, is to avoid unnecessary work by only optimizing code as needed. Code should only be changed for performance if the resulting performance increase is worth the cost to readability, portability, etc, and if increasing a piece of code’s performance would be significant to the overall performance of the software. Related Performance-Tags may include analyzing the cost of performance increases, balancing performance with other concerns, and determining whether a piece of code is worth optimizing.

3) Verifiability
Due to the size and complexity of their calculations, HPC software is often difficult to verify. Scientific simulations especially may be expensive, dangerous, or impossible to physically replicate [12]. Paradoxically, verifying results is extremely important to scientific research [11]. This principle seeks to address this concern and improve HPC verifiability. Related Performance-Tags include analyzing verifiability of certain tasks, and improving verifiability where possible.

Through these principles, Performance-Tags are created and completed, feeding changes for improved performance into the development cycle of the project. Kanban + Performance offers an agile development methodology for HPC while maintaining the emphasis on performance, which could be very useful for the community [13].

Specific principles of focus-leveraged development cycles will be different depending on the team and project. To be truly effective, the process should be further refined by practice and experience over time. Through the examples of Kanban + Sustainability and Kanban + Performance, we have demonstrated how the Kanban + X framework can be applied to any focus.

IV. CONCLUSION
As the security community has demonstrated, leveraging agile development processes towards a focus is beneficial to software projects for which the focus is valuable. Kanban + X provides a generalized framework for leveraging Kanban towards any particular focus, or combination thereof. We believe Kanban + X could be a useful tool for projects with a high-priority focus, such as sustainability or performance. By taking advantage of focus-leveraged development, such as Kanban + X, software projects can be more productive towards their specific goals.

ACKNOWLEDGMENT
This material is based upon work supported by the National Science Foundation under Grant No. 1148188. This material is based upon work funded by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs. We thank program managers Thomas Ndousse-Fetter, Paul Bayer, and David Lesmes for their support.

REFERENCES
[1] D. Mougouei, N. F. M. Sani, and M. M. Almasi, “S-scrum: a secure methodology for agile development of web services,” World of Computer Science and Information Technology Journal, vol. 3, no. 1, pp. 15–19, 2013.
[2] R. E. Maria, “Scrum: a model for safe agile development,” pp. 43–47, 2015.
[3] C. Pohl and H.-J. Hof, “Secure scrum: Development of secure software with scrum,” arXiv preprint arXiv:1507.02992, 2015.
[4] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, “Empirical study of agile software development methodologies: A comparative analysis,” SIGSOFT Softw. Eng. Notes, vol. 40, no. 1, pp. 1–6, 2015.
[5] T. Kevin, Sustainable software development: An agile perspective. Pearson Education India, 2006.
[6] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, and I. Ozkaya, “Managing technical debt in software-reliant systems,” in Proceedings of the 2016 Conference on Future of software engineering research. ACM, 2016, Conference Proceedings, pp. 47–52.
[7] T. Sedano, P. Ralph, and C. Praire, “Practice and perception of team code ownership,” in Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. ACM, 2016, Conference Proceedings, p. 36.
[8] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, “Don’t touch my code!: examining the effects of ownership on software quality,” in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011, Conference Proceedings, pp. 4–14.
[9] T. Sedano, P. Ralph, and C. Praire, “Sustainable software development through overlapping pair rotation,” in Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, 2016, Conference Proceedings, p. 19.
[10] T. C. Lethbridge, J. Singer, and A. Forward, “How software engineers use documentation: The state of the practice,” IEEE software, vol. 20, no. 6, pp. 35–39, 2003.
[11] V. R. Basili, J. C. Carver, D. Cruzes, L. M. Hochstein, J. K. Hollingsworth, F. Shull, and M. V. Zelkowitz, “Understanding the high-performance-computing community: A software engineer’s perspective.” IEEE software, vol. 25, no. 4, p. 29, 2008.
[12] R. P. Kendall, D. E. Post, J. C. Carver, D. B. Henderson, and D. A. Fisher, “A proposed taxonomy for software development risks for high-performance computing (hpc) scientific/engineering applications,” 2007.
[13] M. L. V. D. Vanter, “Hpc needs a tool strategy,” pp. 55–59, 2005.
[14] E. Jeannot and J. Zilinskas, High-performance Computing on Complex Environments. John Wiley & Sons, 2014, vol. 96.