Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 14, Issue. 5, May 2025, pg.130 – 136

A Model for Managing Unplanned Maintenance Work in the Scaled Agile Framework

Ammar Osaiweran

Faculty of Computers and Informatics, Thamar University, Yemen
ammar.osaiweran@tu.edu.ye

DOI: https://doi.org/10.47760/ijcsmc.2025.v14i05.010

Abstract: Scaled Agile Framework (SAFe) is one of the widely used Agile methodologies that enables the use of Agile practices on a large scale. SAFe is recognized for its ability to produce high-quality software systems through program increments. It provides a structured framework on planning software development activities at various levels of the enterprise via various planning events. However, a major challenge of SAFe method is the lack of a structured framework to handle unplanned, previously unknown, maintenance work that suddenly appear during the execution of program increments. This paper provides a solution to this shortcoming by proposing a model to manage the unplanned work emerged during SAFe program increments execution. The model is successfully applied on large industrial setting employing SAFe to manage large software.

Keywords: Scaled Agile, SAFe, Software Maintenance, Software Projects Planning, Software Estimation

I. INTRODUCTION

The Scaled Agile Framework (SAFe) [1] is gaining popularity as an Agile method that scales large enterprises. SAFe provides a set of principles and practices to enable large organizations implement Agile methodologies at scale. It provides an approach to coordinate multiple agile teams, ensuring proper alignment, collaboration, and delivery of large, complex solutions.

Similar to Agile methods [2,3,4,5] SAFe works well for identified planned work with known deadlines. It provides principles and a structure to organize software features across program increments (PI). SAFe ensures a transparent plan of features enabling development teams to accelerate without interruptions. The framework

enables early delivery of minimum viable products to customers to deliver value and to get customers feedbacks as early as possible.

SAFe is known to be an effective method for building large software systems. The software artefacts are delivered in program increments which span roughly 3 months. Each program increment may include several iterations each of which has clear goals of what the team should deliver by the end of the iteration.

The software artefacts built in SAFe are organized as *Stories* which are work that can be done within an iteration, *Features* that consist of multiple stories and usually done within a PI, and *Epics* and *Capabilities* that include multiple features that span three months or more.

Before the start of each PI, the list of features to be treated is known in advance. The features are divided to smaller stories and related work is completed by the development team. The scope of features is fixed and planned in iterations based on priority of the features and the total available capacity of the development team. SAFe advocates focused development iterations where the team works on completing the features without interruptions.

Therefore, for software development where a software system is created from scratch, and similar to other Agile methods [6,7,8,9,10,11,12], SAFe is known to be effective method due to the clear, expected deliverables of each iteration. However, for the software systems that already passed its development phase and entered to its maintenance mode, the plan of features may not be fixed and interruptions become inevitable due to many factors like fixing critical bugs reported from the field or urgent external requests that need immediate response. This results in changes in the PI plan where features may fully or partially descoped.

During software maintenance, urgent requests may occur at any time [19]. There are relatively two types of requests: those that can be planned in later program increments without interrupting current program increment (new features, new version, non-critical bug fixes) and those that cannot (urgent integration issues with other teams, critical bug fixes). Iteration interruptions, pausing some features and reprioritization are therefore inevitable [20,21]. Since the plan of features across iterations matches the capacity of the development team, fitting any new unplanned work will inevitably result in delaying planned features. Renegotiation and communication with affected stakeholders of planned features become a nightmare to the Agile team.

The literature on Agile does not provide much guidance on how to structure and handle urgent work demands during program increments and related iterations. In this paper, we propose a model to handle this type of unplanned work that appear during the iterations, attempting to provide a solution to the above shortcomings. The model is established based on our accumulated experience managing unplanned work while employing SAFe in many sizable projects and various Agile teams.

This paper is organized as follows. Section 2 highlights the related work. In section 3 we give a brief intro to the SAFe framework. Section 4 gives an overview of the type of maintenance work. Section 5 sheds a light on proposed way to handle the unplanned work during the execution of SAFe iterations. Section 6 concludes this paper.

II. RELATED WORK

The work of [22] suggests having two different types of maintenance sprints - a short one for unexpected demands and a longer one for tasks that can be planned. But the work did not show how long is the short sprint and what will happen if there is no unplanned work appear during the short iterations. They found that they had to attend to urgent customer requests even during very short sprints.
In [23], using a buffer to handle unplanned requests is proposed. However, they clearly mentioned that the size of the buffer is always hard to predict and there were no directions on how to set the size of this buffer. The reason of introducing the buffet is that, sprint disruptions need to be managed in order to avoid frustration and demotivation in relation to planning and sprint goals. In [24], 9 heuristics are introduced but very limited attention was focussed on structuring the steps of treating unplanned work.

III. INTRODUCTION TO SAFE

SAFe provides the ability to scale agile methodologies from individual teams to entire enterprises. It promotes four levels of scale: Team, Program, Large Solution, and Portfolio. These levels aim at aligning teams across the organization and ensure flexibility in execution while strategic objectives are met. At its core, SAFe emphasizes transparency, alignment, and delivering continues value to customers.

For effective collaboration between development teams and stakeholders, manage risk and expectations, and relentlessly improve, SAFe promotes structured roles, ceremonies, events, and artifacts.
SAFe emphasizes clearly defined roles. We will introduce the main roles of SAFe which are needed in this paper namely: the development team, scrum master (SM), product owner (PO), chief product owner (CPO) and release train engineer (RTE).

The development team is a group of technical professionals (e.g., developers, testers, designers) responsible for delivering high-quality, valuable solutions within an Agile Release Train (ART). They collaborate to deliver increments of value in short iterations, following Agile principles. The team is self-organizing and is focused on continuous improvement to achieve the ART's objectives. An ART may consist of multiple teams.

The scrum master (SM) helps the development team by facilitating Agile practices and supporting the team resolving impediments. The Scrum Master organizes and facilitates Agile events such as Iteration Planning, Daily Stand-ups, Sprint Reviews, and Retrospectives.

The Product Owner is responsible for defining and prioritizing the features in backlog to deliver maximum value. The PO is the voice of the customer, works closely with stakeholders, and ensures the team understands the broader vision and goals of the ART. The PO is responsible for planning features across the PIs, balancing business needs, customer requirements, team competences, and technical feasibility to drive valuable outcomes.

The CPO oversees multiple Product Owners within an ART. The CPO ensures alignment of priorities, and coordinates value delivery across Agile teams to achieve the overall vision and objectives of the ART. The CPO aligns frequently with the POs of the team in recurrent PO sync meetings.

The Release Train Engineer oversees multiple Scrum Masters in the ART. The RTE is a facilitator for the Agile Release Train (ART). They coordinate the ART's processes, events, and execution, ensuring alignment, removing impediments, and promoting continuous improvement. The RTE acts as a bridge between teams, stakeholders, and Agile leadership to maximize the value of the deliverables.

In this paper, we are mainly concerned with the activities and events that happened at the level of the Agile team. The Agile team consists of a Product Owner, a Scrum Master and a development team of 3-10 members at maximum.

From planning perspective, for each Agile team, the year is divided into program increments (PI) of nearly 8-12 weeks long. The author is used to PIs of 12 weeks so the rest of the paper will be based on this foundation.

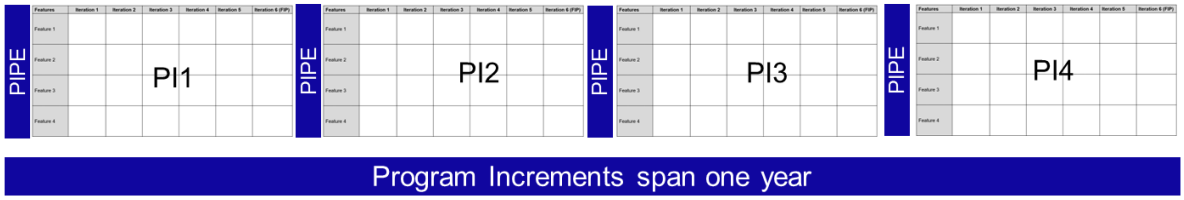

Fig. 1 Shows 4 PIs each starts with a PIPE event

Fig. 1 depicts the organization of four PIs in one year. Each PI consists of 6 iterations where 5 are used to work on the planned features and the last 6th iteration is used for planning the features of the following PI. The PO ensures that features are distributed across all PIs based on draft estimates and related deadlines.

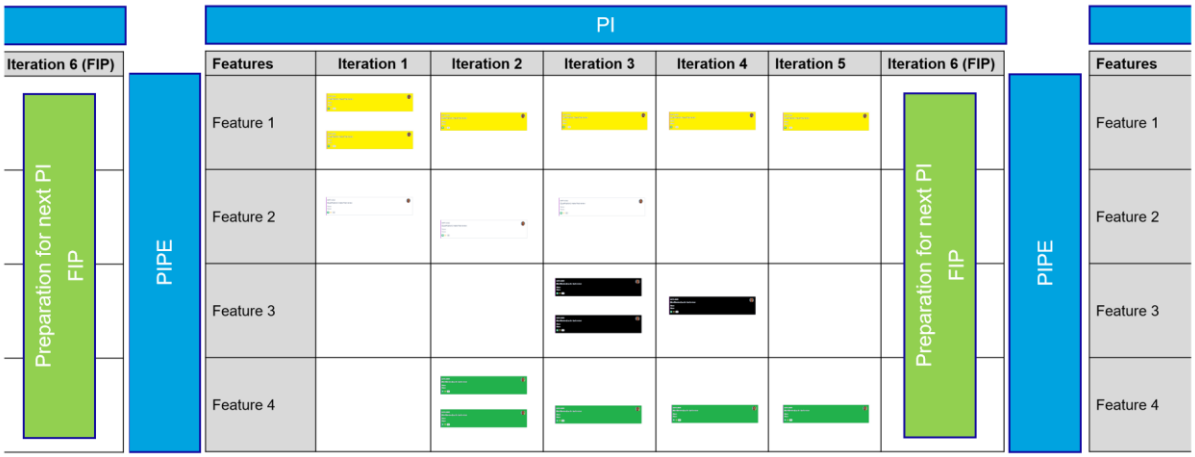

Fig. 2 A single PI that includes committed features and related stories distributed across iterations

Each PI is divided into six iterations each of which of two weeks, see Fig.2. The first five iterations are dedicated to execute planned features agreed during the program increment planning event (PIPE) by the team and customers while the last sixth iteration is called the Innovate and Planning iteration (IP) where the Agile team starts breaking down and estimating the features of the following PI. The features are divided into stories where effort of each story is decided as story points (SPs) during the poker session of each feature during the IP period.

After the IP, the team participates in the face-to-face PIPE event. The PIPE is used to discuss the product roadmap, decide on features to be planned in the PI, and identify dependencies and stakeholders of each feature. The team then distributes the stories of the features across the iterations of the PI taken into account a number of constraints like availability of team members in iterations, total SP capacity of the team in each iteration, deadlines of features, priorities and dependencies to other teams. At the end of PIPE, all stories are distributed across the iterations matching the full capacity of the development team.

As can be infer from the above, SAFe ensures good distribution of known features across the iterations of the PI. However, there is an apparent limitation of executing urgent unplanned work that appears during the course of the PI since the capacity of Agile team is fully occupied to work on the planned features.

IV. TYPE OF UNPLANNED WORK

The focus of this paper is to provide a framework for handling unplanned work that appear during the execution of iterations within a PI. Such software systems are released to customers and are already in the maintenance mode. Maintenance work has the following types [18]:

- Adaptive – responding to changes in the software environment
- Perfective – responding to new user requirements
- Corrective - fixing errors
- Preventive – preventing foreseeable problems in the future.

All these types of maintenance work can appear as a sudden urgent unplanned work during the iterations which require the development team to treat them immediately causing interruption on the execution of planned features. For example, an adaptive type of work may appear as a result of client's migrating to a newer operating system or infrastructure software with a newer incompatible version. A perfective work may appear as an urgent client request to make an existing feature in the software complete. A corrective type of work may correspond to fixing critical bugs that brings the software system at a client to a halt. And preventive work may be discovered by the development team as an urgent work to be completed immediately to avoid future issues.

Additionally, from our experience, the unplanned work may emerge as support requests from clients to understand the system or how to use it. Other type of work is a request from other teams in the organization, who develop an external module, to understand how to use the interfaces of the modules created by the development team. Moreover, other support requests may emerge as helping external team with debugging errors that originated during system integration or regression testing.

V. PROPOSED FRAMEWORK FOR HANDLING THE UNPLANNED WORK

One common practice to handle and solve urgent requests is to delay low priority features to next PI. But this becomes a nightmare to the team when the amount of unplanned work increases throughout the iterations affecting the ongoing planned feature and causing unpleasant context switching.

To remedy this, we propose an alternative way to plan the unplanned work in the PI by reserving story points as a buffer for the unplanned work. The following diagram shows the proposed steps to be followed for handling unplanned work during the PI. The steps are grouped in three main phases during the PI, see Fig. 3., and as follows:

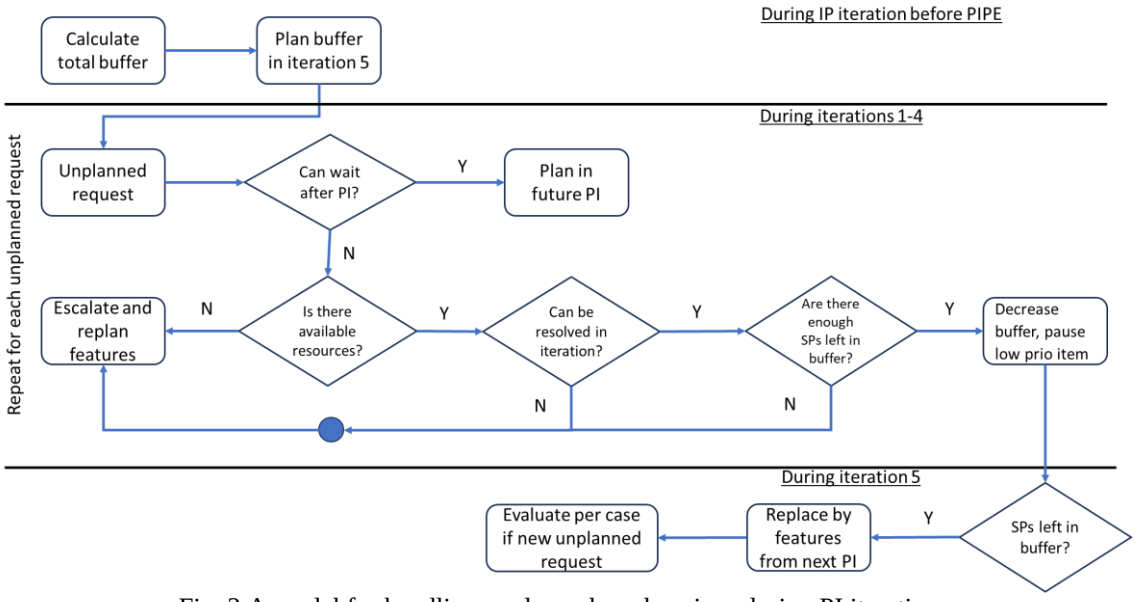

Fig. 3 A model for handling unplanned work arrives during PI iterations

A. During IP iteration before PIPE

Before the PIPE event, 2 steps can be performed. The first step is to calculate total buffer. The calculation is done as follows. During the IP iteration, the total story points available for features of the next PI are calculated based on the availability of team members. Once the final story points are known, the team subtract a percentage to reserve story points in advance for working on the unplanned work. The rest of story points will be used for the planned features, see Fig.4. The percentage is calculated based on historical data accumulated during past four PIs. We calculate the average percentage spent for the unplanned work during the past four PIs and use it for the future PI. As mentioned earlier, the remaining story points will be used to plan future features known in advance.

Fig. 4 Buffer reserved for unplanned work from the total available capacity

The second step is to plan the story points of the buffer in iteration 5. During the IP iteration, the team will breakdown the features to stories and estimate each story as a team in an event called poker session. Based on that, the final breakdown of features should match the predefined story points allocated to each feature before the breakdown. If the breakdown of a feature results in story points more than the predefined story points, the PO will distribute the feature across multiple PIs and align with related stakeholders on the time of the final delivery. If the story points match, the team then distributes the stories to the iterations based on the priority of features and the availability of the team members who will work on the stories in the iterations.

To plan the buffer of the unplanned work in iterations, there are 2 options. The first is to distribute the story points of the buffer evenly across the iterations. The second is to plan the buffer at the last 5th iteration.

From experience, the first option has one drawback. If the unplanned work comes, it can be treated immediately because the story points to work on it is available. However, if non comes, the intention is to replace the reserved buffer by stories planned in the next iteration. In practice, this proofs to be impractical especially when the iteration is about to reach its end and it is not guaranteed that the unplanned work may not appear. More important, it is possible to lose these story points without working on any feature making the development team more relaxed.

The second option is more preferred and will be the basis for the rest of the steps in the model. It proofed to be more practical and led to using the story points reserved for the unplanned work more effectively.

B. Steps during iterations 1-4

Iterations 1-4 are reserved to work on the planned features that were broken down during the IP iteration. At the start of every iteration, the team revise the stories planned in the IP iteration of which story points match the total capacity of the team during the iteration. The team sets the goals of the iteration and starts work on the planned stories. A number of steps can be performed during the execution of the iteration in case unplanned work arrives.

The following activities are proposed for the *Unplanned request.* If there are no unplanned request comes during the iteration, the team is expected to realize the goals of the iteration and finish all planned stories. However, if unplanned request comes, a number of queries and alignments must be performed by the PO.

The PO checks with the requester the urgency of the request. From the one hand, if the request is not urgent and *can wait* after the end of existing PI, the PO will align with the requester and related stakeholders on working on the request as a planned feature in future PIs. The PO will make sure that members of the development team will not be distracted shielding the team from external noise.

From the other hand, if the request is urgent and needs immediate attention, the PO will check if there are available team members with the required competence needed to resolve the request. If no team member can help resolving the issue, the PO will escalate to the CPO and RTE to discuss the details of possible solution directions and consequences on the current planning. Usually, development team will require more time to build a competence first before providing solutions to the issue. Consequently, existing plan will be adapted and some features may be fully or partially descoped from the PI.

If the unplanned request is known and there is available competence in the team, the team will evaluate the effort needed to provide a solution to the request. Usually, the unplanned work is treated by a subset of the team while the rest continue working on the planned features.

If the effort needed to resolve the unplanned work cannot fit the iteration, the PO will start the escalation procedure as explained above. If the effort is small enough to be treated during the iteration by the allocated team members, the team will pause current feature. The SM of the team will create a new story to work on the unplanned work assigning the necessary SPs. The same SPs will be subtracted from the main buffer allocated at iteration 5 and low priority stories with the same SPs will be candidate to move to the following iteration. The above behaviour repeats itself for each unplanned feature appear during iterations 1-4. This means new stories will be added during iteration 1-4 and the buffer planned in iteration 5 will in turn decrease.

C. During iteration 5

At the start of iteration 5, the PO will check how many SPs left in the buffer. Based on these SPs, the PO may bring stories of features planned in next PI to be completed in iteration 5.

VI. CONCLUSIONS

In this paper we presented a model to handle unplanned requests that arrive during the iterations of the program increments. The aim is to provide a solution to the known shortcomings of Agile methods that deals very well with planning known deliverables via perfect features, epics and solutions. The framework appeared to be practical in industrial settings and successfully applied to treat unplanned work in various PIs for many SAFe Agile teams.

An apparent limitation is proposing the correct size of the buffer when past PIs unplanned requests substantially vary in size. The resulting average size of the buffer may at some cases be smaller than the total story points needed to cover all unplanned requests. This results in descoping planned features to next PIs. Also, the size of buffer may be big in size to cover all unplanned requests. This will result in breaking down future features of next PI during iteration 5 to bring the feature to the team compensating the remaining story points left in the buffer. As a future work we are planning to resolve this shortcoming and apply the framework on more teams.

REFERENCES

[1]. Scaled Agile, Inc. (2023). SAFe 6.0 Framework Overview. Retrieved from https://www.scaledagileframework.com/

[2]. Leffingwell, D. (2020). Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley.

[3]. Knaster, R., & Leffingwell, D. (2020). SAFe 5.0 Distilled: Achieving Business Agility with the Scaled Agile Framework. Addison-Wesley.

[4]. Larman, C., & Vodde, B. (2016). Large-Scale Scrum: More with LeSS. Addison-Wesley.

[5]. Boehm B, Turner R (2003) Observations on balancing discipline and agility. In: Proceedings of the Agile Development Conference, Salt Lake City, Utah, USA. IEEE Computer Society, pp 32–39

[6]. Abrahamsson P, Salo O, Ronkainen J, Warsta J (2002) Agile software development methods: review and analysis. VTT, Finland

[7]. Cockburn A (2006) Agile software development: the cooperative game. Addison-Wesley Professional, Boston

[8]. A. Cockburn and J. Highsmith, "Agile software development, the people factor," in Computer, vol. 34, no. 11, pp. 131-133, Nov. 2001, doi: 10.1109/2.963450

[9]. Nerur S, Mahapatra RK, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Commun ACM 48(5):73–78

[10]. Moe NB, Dingsøyr T, Dybå T (2010) A teamwork model for understanding an agile team: a case study of a Scrum project. Inf Softw Technol 52(5):480–491

[11]. Paulk MC (2002) Agile methodologies and process discipline. Crosstalk-J Def Softw Eng 1(1):15–18

[12]. Michal Hron, Nikolaus Obwegeser, Why and how is Scrum being adapted in practice: A systematic review, Journal of Systems and Software, Volume 183, 2022, 111110, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2021.111110.

[13]. F. u. Rehman, B. Maqbool, M. Q. Riaz, U. Qamar and M. Abbas, "Scrum Software Maintenance Model: Efficient Software Maintenance in Agile Methodology," 2018 21st Saudi Computer Society National Computer Conference (NCC), Riyadh, Saudi Arabia, 2018, pp. 1-5, doi: 10.1109/NCG.2018.8593152.

[14]. Vasanthapriyan, S. & Arachchi, K. M. (2020). Effectiveness of Scrum and Kanban on Agile-Based Software Maintenance Projects. In C. Pang (Ed.), Software Engineering for Agile Application Development (pp. 282-300). IGI Global Scientific Publishing. https://doi.org/10.4018/978-1-7998-2531-9.ch013

[15]. Mohammed Almashhadani, Alok Mishra, Ali Yazici and Muhammad Younas , Challenges in Agile Software Maintenance for Local and Global Development: An Empirical Assessment, Journal: Information, 2023, Volume 14, Number 5, Page 261, DOI: 10.3390/info14050261

[16]. K. S. K. Ibrahim, J. Yahaya, Z. Mansor and A. Deraman, "The Emergence of Agile Maintenance: A Preliminary Study," 2019 International Conference on Electrical Engineering and Informatics (ICEEI), Bandung, Indonesia, 2019, pp. 146-151, doi: 10.1109/ICEEI47359.2019.8988815.

[17]. Agh, H., Ramsin, R. Scrum metaprocess: a process line approach for customizing Scrum. Software Qual J 29, 337–379 (2021). https://doi.org/10.1007/s11219-021-09551-4

[18]. Lientz BP, Swanson EB (1980) Software maintenance management. Addison Wesley, Reading MA

[19]. Bennett KH, Rajlich VT Software maintenance and evolution: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, 2000. ACM, pp 73-87

[20]. Poole C, Huisman JW (2001) Using extreme programming in a maintenance environment. Software, IEEE 18 (6):42-50

[21]. Poole CJ, Murphy T, Huisman JW, Higgins A Extreme maintenance. In: Software Maintenance, 2001. Proceedings. IEEE International Conference on, Florence, Italy, 2001 2001. IEEE, pp 301-309

[22]. Pino FJ, Ruiz F, Garcia F, Piattini M (2012) A software maintenance methodology for small organizations: Agile_MANTEMA. Journal of Software: Evolution and Process 24 (8):851-876

[23]. Schwaber K, Beedle M (2001) Agile software development with Scrum. Prentice Hall, Upper Saddle River, New Jersey, USA

[24]. Heeager, L.T., Rose, J. Optimising agile development practices for the maintenance operation: nine heuristics. Empir Software Eng 20, 1762–1784 (2015).

