A Novel Technique to Assess Agile Systems for Stability

Robert Healy[1(✉)], Tapajit Dey[2], Kieran Conboy[3], and Brian Fitzgerald[2]

[1] Intive Ireland, 6th Floor O'Connell Bridge House, Dublin 2, Ireland
rob.healy@intive.com
[2] Lero – The SFI Research Centre for Software, University of Limerick, Limerick, Ireland
[3] School of Business and Economics, University of Galway, Galway, Ireland

Abstract. Agile systems, like the Kanban and Scrum frameworks, are built on assumptions of sustainability and stability, however, there is little empirical evidence on whether such systems are stable in practice or not. Therefore, in this study we aim to inspect the stability of Agile systems by leveraging the concept of stability described in Queueing Theory. We define a novel metric, the Stability Metric, as a way of assessing queueing systems, especially Agile systems. We inspect 926 Jira projects in 14 organizations with over 1.6 million product backlog items using this metric. The analysis showed that 72.89% of these Jira projects were not stable and stable systems, on average, had product backlog sizes 10 times shorter than unstable ones. These results suggest that while the goal of Agile is to create a sustainable, stable way of working, this is not guaranteed, and a better understanding of systems and queues may be required to help design, create, coach, and maintain optimal Agile systems.

Keywords: Agile · Queueing Theory · Stability · Jira · Backlogs

1 Introduction

The United Nations define sustainability as "meeting the needs of the present without compromising the ability of future generations to meet their own needs" [1]. Agile software development has existed at least since the 2001 Agile Manifesto, with many of the frameworks predating the manifesto itself [2]. One of the principles of Agile is that "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely" [3]. Two of the most popular Agile frameworks are Scrum and Kanban [2]. Both frameworks can be modelled as queuing systems. In queueing systems, sustainability occurs under conditions of mathematical stability [4]. In this paper, we present a novel approach by leveraging the concept of stability to assess if Agile systems are stable and sustainable in practice as is commonly assumed.

To measure the stability of Agile systems we used queueing theory to derive a new metric, the Stability Metric (SM) to classify the performance of queues. We applied this metric to 926 collections of Jira Projects (JPs), of Product Backlog Items (PBIs) from the

C. J. Stettina et al. (Eds.): XP 2023, LNBIP 475, pp. 20–33, 2023.
https://doi.org/10.1007/978-3-031-33976-9_2

more than 2.5 million records in the Public Jira Dataset [5]. The distribution of Stability Metric across all JPs and the relationship between Stability Metric and product backlog size and inter-service arrival time was extracted and analyzed.

The rest of the paper is organized as follows: In Sect. 2, we discuss and briefly present the background and related concepts. We describe the research approach in Sect. 3 and the results in Sect. 4. We provide further discussion about the results and its implications in Sect. 5. Finally, we describe the limitations to our study in Sect. 6 and offer conclusions in Sect. 7.

2 Background

2.1 Queueing Theory for Stable Queueing Systems

Queueing systems have been researched since Erlang's work on telecommunications in the early 20th century [6]. A queue forms in a service when demand for that service exceeds supply [6] and one or more interconnected queues form a queueing system [4]. Each queue can be stable, unstable, or marginally stable [4]. A queue is considered stable when the Markov chain of all possible queuing states is ergodic in nature [4]. This means that a stable queue must include the possibility of occasionally having no items in it. Erlang defined the traffic intensity, ρ as the dimensionless ratio of the average arrival rate, λ, to the average service rate, μ, where both λ and μ are measured in items per unit time as shown in Eq. 1 [6].

$$\rho = \frac{\lambda}{\mu} \tag{1}$$

In this case, the queue can be seen to be stable when the traffic intensity is less than one, or when the service rate is greater than the arrival rate. In telephony systems, this allows for the service rate to be fixed and the system is controlled by altering the arrival rate to ensure that the system is stable. From Jackson's Theorem for networks of queues we know that a network of queues will be stable only when all the sub-queues within that network are stable [4]. For this research we will model Agile frameworks as one or more queues.

2.2 Modelling Agile Frameworks as Systems of Queues

Kupainen *et al.* [7] conducted a systematic literature review of 774 papers to identify key metrics in use by Agile teams for planning, progress tracking, software quality measurement, fixing software process problems, and motivating people. One of the metrics for progress tracking they identified is Work-in-Progress (WIP) limits. WIP extends from Little's Law – an observation that in stable queues the amount of time an item spends in a queue, *W,* is equal to the average number of items in the queue, *L,* divided by the average arrival rate of items, λ [8]. Therefore, in theory, it is possible to control the throughput of a stable Agile queue by simply enforcing a limit on the number of WIP items by using Eq. 2 [8].

$$W = \frac{L}{\lambda} \tag{2}$$

The Kanban framework is already constructed around the principles of queueing theory [8]. A simple Kanban system, such as that shown in Fig. 1, is a queue with one or more servers. The "Backlog" column represents the queue, the "In progress" column represents the servers and the "Done" column represents items exiting the queue. Unlike traditional queueing systems, the order of a Kanban backlog tends not to be a simple First-In First-Out (FIFO) or Last-In First-Out (LIFO) queue and instead tends to be dynamically re-ordered. Also, many real Kanban systems tend to have multiple c*olumns* and *swimlanes*. A *column* usually represents an upstream or downstream service such as "development" and "testing". However, a column can also be a separate queue if items are placed there without any action being performed on them. Examples of this include "Ready for testing" or "Waiting / Blocked". A *swimlane* is a horizontal division across a Kanban board that often represents a separate class of service – e.g., a priority item. Each swimlane may be considered as an independent queue.

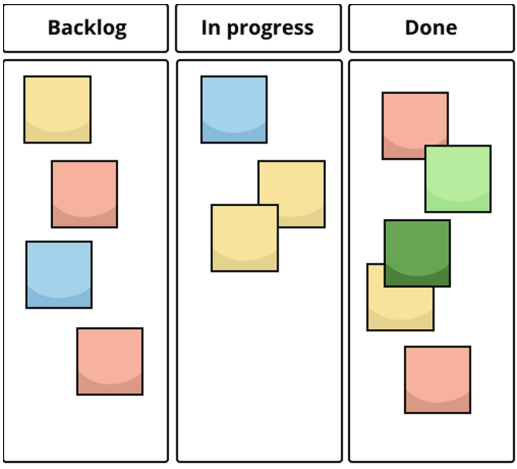

Fig. 1. Sample Kanban board

The Scrum guide, on the other hand, defines two queues – a primary "product backlog" that is managed and dynamically prioritized by the Product Owner and a "sprint backlog" that is selected by the team at the sprint planning event, with items chosen from the product backlog based on available capacity [9]. As a result of how items move from product backlog to sprint backlog, standard Scrum systems tend to be more batched than simple queues. However, the overall system still can be modelled as a simple queue if the arrival and service rates are stochastic in nature and can be modelled using a Poisson and exponential distribution respectively [4]. In practical terms, a stable Scrum system, like a stable Kanban system and like other similar stable Agile queueing systems, has the advantage of being highly predictable. This happens when the team are, on average, more than capable of delivering the work at least as fast as it is being requested of them.

3 Research Approach

With the theoretical foundation of the Queuing Theory in mind, the primary research question we are addressing in this paper is: ***RQ: Are Agile systems stable from a queueing perspective?*** To answer this question, we use a metric derived from the Queueing theory called the Stability Metric, as described below, and analyze a Public Jira dataset [5] containing 16 public Jira repositories involving 1822 Jira projects to determine if those systems were stable.

3.1 Introducing the Stability Metric (SM)

For Agile queues, the arrival rate of items into a product backlog is often not within the control of the team, or even the organization if customer-reported bugs are included. However, the service rates can be controlled through team design, scaling multiple teams, and training and coaching individual team members. Although queueing theory is known in Agile literature and metrics such as cycle time and WIP limits exist [7, 8], typical measures of queues as systems have not been used. We define a new term – the "Stability Metric (SM)", ψ, that is the inverse of the traffic intensity and is shown in Eq. 3.

$$\psi = \frac{\mu}{\lambda} \tag{3}$$

The Stability Metric ranges from zero to infinity; values from zero to lower than one indicates an unstable queue system, a value of one indicates a marginally stable system, and values above one indicates stability. The Stability Metric has the advantage over its predecessor, the "traffic intensity" metric, that the items that are within control are focused on the numerator rather than the denominator so a positive change in ability results in the Stability Metric growing. Unlike existing popular Agile metrics such as "velocity" or "cycle time" [7], the Stability Metric is not purely a lagging metric. Instead, we believe it may be used as both a diagnostic metric to understand why a system is performing/underperforming and as a design tool to help organize teams and Agile systems to manage predicted workloads effectively. We use this metric to help us address our research question.

3.2 Analyzing the Public Jira Dataset

Past studies of Agile systems have focused on contextual realism with case studies of real software engineering teams, although these studies are typically not grounded in theory [10]. Montgomery *et al.* [5] curated and published a dataset of the contents of the 16 public issue tracking systems with 1822 Jira projects and 2.7 million Product Backlog Items (PBIs) all using the Atlassian Jira issue tracking tool, which they suggest is the leading issue-tracking tool for Agile systems. We decided to analyze this dataset to make the results as generalizable as possible. This dataset, however, is not very clean for our purposes, i.e., it is hard to determine if individual Jira projects use any standard Agile framework, or the degree to which Jira projects can be mapped to a framework. While some parts of the dataset suggest an Agile implementation (e.g., the existence of User Stories which is an XP concept that is often used in Scrum [2]), there are other parts of

the dataset that suggest that it could be used by ITIL/traditional waterfall-type projects (for instance, "Change Requests") or helpdesk-type work such as "Support Request". We assumed that most or all the 2.7 million PBIs will be part of one or more queues and that some of these queues are part of an Agile framework. Using data from the dataset it is possible to calculate the arrival and service rates of each issue and calculate the stability.

To analyze the Public Jira Dataset, we first downloaded it from its public repository and, following instructions from Montgomery *et al.* [5], restored it to a MongoDB database. The dataset is organized into a set of 16 "repos" where each repo is the extracted Jira information from one of the issue tracking systems. Using a Python script, we extracted the following fields from each repo: *Issue ID, Project Name, Issue Type, Subtask Boolean, Assignee ID, Created Date/Time, Resolution Date/Time, Status Name.* We subsequently collated this data into a comma separated value (CSV) file for each organization for each month between January 2002 and January 2022. These CSV files were then combined into a Microsoft Excel file for all issues for each Jira repo.

The Jira repos are divided into "Projects" where Atlassian advise all issues related to a product should be assigned to a Jira Project (JP) [11]. Only JPs with more than 30 issues were considered, this was to reduce skew caused by new/inactive/abandoned JPs and allow the assumption of a normal distribution around the calculated mean as per the central limit theorem. With that filtering criterion in mind, we removed the *SecondLife* and *Mindville* repos as they had fewer than 30 issues each. For the remaining repos, all the Epic issue types and Subtask issue types were filtered out to ensure the work was approximately similarly sized. Epic issue types are used in Jira as "parents" of other ticket types [12] and remain unresolved longer than other ticket types, potentially skewing service rates. On the other hand, Subtask ticket types are used in Jira as "children" of standard ticket types [13] and, as such, tend to be resolved in shorter periods, also potentially skewing the data.

When team completes a PBI, it receives a resolution. Resolution types can be configured per JP, but some default types exist [14]. There were 81 separate resolution types across the JPs in the dataset. Of these only six denoted the delivery of a successful piece of work. Table 1 lists them below. The other seventy-six resolution types denoted abandoning the queue before the team completed the PBI. We removed all PBIs with these resolutions also.

Table 1. Top 6 successful resolution types.

Resolution	Total PBIs	Percentage of total resolved PBIs
Fixed	968080	41.9%
Done	371312	16.1%
Resolved	2904	0.1%
Deployed	136	0.0%
Fixed-Verified	116	0.0%
Delivered	102	0.0%

This resulted in 926 JPs with 1,633,166 PBIs between 2002 and 2022. 1,295,002 PBIs were successfully resolved by teams during that the sampling period. For each JP, we calculated the arrival rate and service rate. We calculated the arrival rate, λ, for each system, as per Eq. 4. We calculated the service rate, μ using Eq. 5. We calculated the Stability Metric, ψ, as shown in Eq. 3, from these and the results were grouped into Unstable ($\psi < 1$), Stable ($\psi > 1$), and Marginally Stable ($\psi = 1$).

$$\lambda = \frac{\#tickets\ created\ that\ are\ unresolved\ or\ resolved\ successfully}{(datetime\ of\ last\ ticket\ created - datetime\ of\ first\ ticket\ created)} \tag{4}$$

$$\mu = \frac{\#tickets\ created\ that\ are\ resolved\ successfully}{(datetime\ of\ last\ ticket\ resolved - datetime\ of\ first\ ticket\ resolved)} \tag{5}$$

As stated in Sect. 2.1, a requirement of stability is for a queue to be a Markovian chain with the property of ergodicity. In such a scenario, the size of the queue will eventually and temporarily drop to zero. The dataset allowed us to evaluate how big queue/product backlog was for each JP at the point where the data was captured. We calculated the product backlog, L_{JP}, by taking the total PBI's that had arrived, A_{JP}, and subtracting the total PBI's that has been resolved, Z_{JP}, as per Eq. 6. We plotted these in relation to the backlog size.

$$L_{JP} = A_{JP} - Z_{JP} \tag{6}$$

Finally, we devised a proxy test for Agility. Equation 7 shows the calculation for the average inter-service time for each JP. Both Scrum and Kanban advocate for discretization of work, breaking work into small parts and monitoring progress daily [2]. In Scrum all PBI's must fit into a timeboxed sprint that is commonly two weeks in duration but can be any period up to a month in length [9]. Large epics and very small sub-tasks had been filtered at an earlier stage, so this analysis acts as a gauge of how many JPs may have contained Agile systems by measuring how many could not have fit in a Scrum system. Systems with PBIs with durations of weeks, months or longer could indicate a phased based "waterfall" approach.

$$t_{JP} = \frac{1}{\mu_{JP}} \tag{7}$$

4 Results

In this section, we discuss the results of the analysis outlined above. First, we present the stability distributions of all 926 relevant JPs in the Public Jira Dataset. Then we present the relationship between the stability and backlog size. Finally, we show the data on average inter-service time and demonstrate its relationship to stability.

4.1 Stability of JPs

Table 2 shows that nearly three-quarters, 72.89%, of JPs are unstable from a queueing perspective, with around one quarter, 24.92%, appearing to be stable and the remaining

Table 2. Stability Metrics of PBIs

Stability	Stability metric, ψ	JP count	Percentage
Unstable	0.77	675	72.89%
Marginal	1.00	20	2.16%
Stable	1.80	231	24.92%

2.16% marginally stable. Figure 2 shows the distribution of stability across all JPs with a clear tendency for JPs to cluster around marginal stability and a tendency for most to be slightly unstable with 54.05% of all JPs being in the range 0.7–0.99.

Figure 2 also shows that there are outliers on both the upper and lower ends. At the lower end, 2.04% of all JPs have an arrival rate more than ten times faster than the service rate. For every PBI delivered by the people working on such a system, ten new PBIs arrived at the same time, on average. At the upper end of the scale, 4.39% of JPs result in workers involved there having nothing to do on the JP more than half of the time, on average.

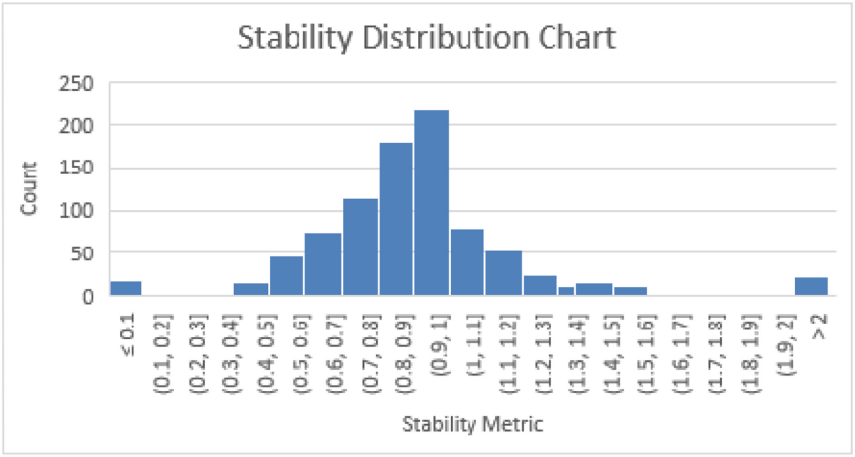

Fig. 2. Distribution of the Stability Metric across 926 JPs

Table 3 shows the proportion of stable and marginally stable JPs as a share of all JPs. This shows that none of organizations have consistently stable systems. There does not appear to be a relationship between the number of JPs per organization/repo and the percentage of stable systems. However, 86% of organizations have at least one stable JP.

4.2 Stability and Backlog Size

When Montgomery *et al.* captured the Public Jira Dataset in 2022, 338,164 PBIs were unresolved across 887 JPs. Table 4 shows the relationship between each level of stability

Table 3. Stable and marginally stable JPs for all Repos.

Repos	Number of stable/marginally stable JPs	Count of JPs	Percentage
JFrog	0	6	0.0%
Sonatype	0	2	0.0%
Jira	2	26	7.7%
Mojang	1	8	12.5%
Qt	3	17	17.7%
JiraEcosystem	10	55	18.2%
Sakai	4	20	20.0%
MariaDB	2	9	22.2%
Apache	125	486	25.7%
MongoDB	8	27	29.6%
RedHat	65	198	32.8%
Hyperledger	7	18	38.9%
Spring	22	52	42.3%
IntelDAOS	1	2	50.0%
All JPs	**250**	**926**	**27.0%**

and the corresponding backlog size for all 926 JPs. It shows that the biggest backlog size appears when the system is unstable – this makes sense as, by definition, the system is unable to meet the demands being placed on it. The smallest backlog sizes occur when the system is marginally stable. Again, this is logical, since for those systems, the time spent waiting for a new piece of work to arrive is eliminated – there is minimal waste, but the system is not overloaded. Systems with high stability may wait to accumulate PBIs before addressing at high speed.

Table 4. Stability metrics and backlog sizes of PBIs.

Stability	Mean stability metric, ψ	Mean backlog size	JP count
Unstable	0.77	455.39	675
Marginal	1.00	80.25	20
Stable	1.80	126.55	231

Figure 3 expands on Table 2 by illustrating the relationship between stability and backlog sizes. A point to note is the significant ranges in backlog sizes which vary between 0 and 18,956. Of these, the largest backlogs are an order of magnitude larger than marginally stable or stable queues. 71% of all stable and marginally stable systems have backlogs between 0 and 100 items in them. By contrast, 49% of all unstable systems have backlogs of one hundred or more.

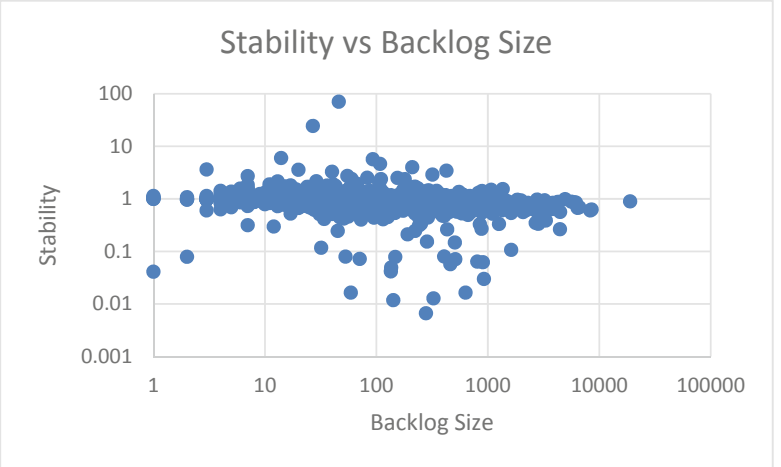

Fig. 3. Backlog size vs stability of all JPs, plotted on logarithmic axes.

4.3 Stability and Inter-service Times

Figure 4 shows the distribution of average inter-service times for all resolved PBIs. These were calculated using Equation 7. 64% of PBIs are resolved within eleven calendar days and that 88% are resolved within 31 calendar days. This indicates that some PBI's may have been used in an Agile system.

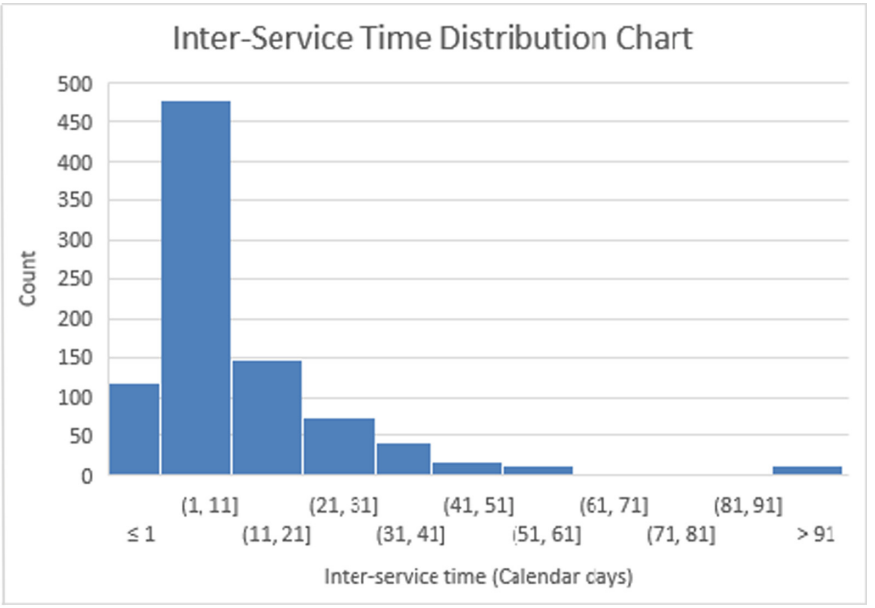

Fig. 4. Distribution of inter-service time

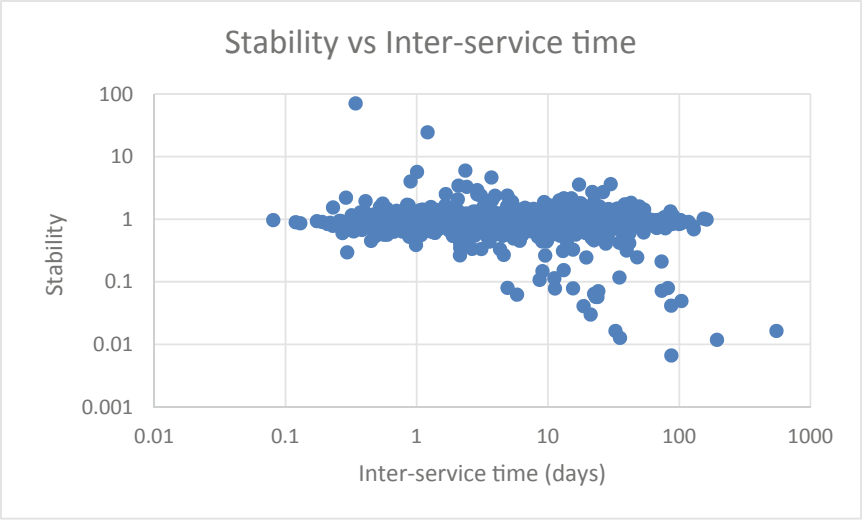

Fig. 5. Inter-service time vs stability of all JPs, plotted on logarithmic axes.

Figure 5 shows the relationship between inter-service time and stability. The outliers on the lower right in this plot demonstrate that very unstable systems tend to have high service times while the outliers on the upper left show that systems with very high stability tend to have very fast service times. However, perhaps the most interesting information on the plot is the broad range of inter-service times over which systems tend to cluster around marginal stability. Over 33% of all JPs have stability in the range of 0.9 to 1.1. Of these, the service rates range from 0.08 days between PBIs resolved on average to 161.58 days between PBIs resolved on average. This suggests that a wide array of systems with different arrival rates, people, tools, technologies are all independently all adapting processes to attempt to stabilize their systems. These results will be discussed further in the Discussion section.

5 Discussion

The analysis of the 926 JPs of the Public Jira Dataset illustrates that it is possible to extract anonymous data from Jira projects and process it using a novel measure, the Stability Metric to be able to infer the stability of the system. We illustrated the relationship to backlog size for a large historic dataset with over 1.6 million items over a 20-year period. The results show a strong tendency for the system to be unstable with 72.86% showing this. These unstable systems have, by definition, growing backlogs and we also find that they tend to have larger backlogs overall. Our results show that there is a discrepancy between the assumed stability of Agile systems and the reality of how systems operate in practice and seems to indicate that the assumption of sustainability associated with the Agile process may not hold good in practice due to the systems' lack of stability.

However, over half of all JPs are nearly stable, with 57.6% of all JPs are in the range 0.8–1.2 so future research may investigate whether it is possible, or advisable, for

systems to be adapted so they become more stable in nature. Of the 14 organizations whose data was analyzed, all but two had some systems either stable or marginally stable, ($\psi \geq 1$). This means that although all organizations had some unstable systems, most could make some systems stable. Further investigation of real teams using the stability metric can help identify which techniques help teams achieve stable ways of working, and whether these correlate with other metrics or measures of satisfaction.

An interesting result is the percentage of stable systems in the dataset – 27% of all JPs analyzed were stable or marginally stable. From a practical perspective the tendency towards a Stability Metric value of close to 1 makes sense as organizations that achieve this can deliver work as needed, without having frequent idle resources. This finding was supported by the analysis of the relationship between average backlog sizes and the stability metric, with a tendency for significantly larger backlog sizes to correspond to unstable systems. Whilst this is suggestive of a causal relationship between backlog size and stability, this also will need to be tested with actual teams.

Another effect worth considering in further research is the impact the statistical necessity of having no backlog has on system stability. A team with nothing to do is likely to be assigned to other work by the management since their concern of using the available manpower optimally conflicts with the conditions of having a mathematically stable system where a team having no work is expected, and even required, to happen periodically. In Scrum, a high-performing team with no carryover of work achieves this to a degree every sprint but a Kanban team who have a continuous flow of work may never eliminate work-in-progress and achieve a stable system. This calls into question the applicability of Little's Law which requires a stable system to be applicable. Further research is needed to analyze real organizations to assess whether high utilization rates is a higher priority than system performance, because this may suggest a business incentive for not striving to achieve sustainable work systems.

Despite most JPs appearing to be close to stable where $\psi \approx 1$, significant outliers do exist where JPs had high levels of instability $\psi < 0.1$ and stability $\psi > 2$. Both outliers are worth considering as neither are sustainable for the people performing the work nor the organizations in which these systems exist. Where $\psi < 0.1$, the average arrival rate is more than 10 times the average service rate. This is likely to be a stressful situation, as demand outstrips supply and difficult choices will need to be continually made as to which is the next most important piece of work. For systems where $\psi > 2$ this means that the people involved in performing the work have nothing to do for more than half the time on average. Neither outlier looks sustainable from a business or human perspective – the business would under-perform compared to competitors and the developers might suffer from burnout or boredom. Future research is needed to determine if these stability metrics exist in Agile systems, as it is not possible to determine the precise conditions from the data, but if substantiated, it would strongly reject a hypothesis of sustainable ways of working as neither sponsor, developer nor user could be reasonably expected to sustain unstable ways of working indefinitely.

5.1 Potential Application of this Metric

The Stability Metric is still under investigation and, as Sect. 6 describes, there remain several limitations in the investigations to date. It may be useful to discuss potential

applications. This metric is adapted from existing measures from queueing theory but is novel to analysis of Agile systems. It is intended as a diagnostic tool to help predictability. For all Agile systems it can help diagnose if the team is being under-loaded or over-loaded and the degree to which this is occurring. This, in turn may help in overall organizational design and establishing the appropriate number of teams for the volume of work required.

On a more operational level, by shifting the duration over which the data is used a comparison can easily be made between stability over a longer period and stability in the recent past. This will provide information to the team and stakeholders on the impact of continuous improvement initiatives. In Kanban systems, the Stability Metric can be used to determine if the system is stable and if applying Little's Law and limiting WIP is a valid approach. It could be used on an overall system process flow and on any sub-system within that flow to quantify and control process bottlenecks. In Scrum, it could be an improvement on the current practice of "yesterday's weather" to help teams identify the target velocity required per sprint to achieve a stable product backlog based on longer term "climatic" patterns of arrival and service rates. However, before we get too excited about potential applications, we must recognize this research is ongoing and list the limitations of the work to date.

6 Limitations

The use of a large dataset that crosses many organizations for a prolonged period offers very good generalizability of findings, but it is at the payoff of contextual realism. The main limitation of this work is that it is not known with 100% certainty if an organization we studied used an Agile framework. The analysis of service rates shown in Fig. 4 suggests that the majority PBIs were discretized to fit within short cycles used in Scrum and Kanban. Also, the use of Jira itself is suggestive of Scrum or Kanban approaches as these are standard frameworks within the tool [16].

Another limitation of this study was the use of Jira Projects, JPs, as a collection of PBIs that represent a queuing system. Jira can use a JP for this purpose but may also slice the collections of PBIs within and across JPs to create queues for teams. Without the specific details of what queue or queues are in use in a given organization it is not possible to be certain of the stability of each. We have relied on Jackson's Theorem for queueing networks which dictates that a network of queues can only be stable if all sub-queues in the network are stable [4]. This suggests that for JPs that displayed stability, any sub-queues in that JP must have also been stable. The situation is more complicated for unstable queueing networks; it is possible for a network to be unstable as long as one or more of the sub-queues are unstable. Further investigation of real teams will be needed to determine the prevalence of instability.

Methodologically, one limitation may arise in the fact that we used averages across the entire JP. This potentially ignores temporal effects. A queue is a dynamic system and may be temporarily stable or unstable. The degree to which stability fluctuates and root causes should be investigated further, probably with real teams.

Finally, the study assumed that the data provided were accurate. Since the creation and update of PBIs in Jira is a human activity there is likely to be variation in the accuracy

of the data. For example, we assumed the work arrived in the queue at the point where the PBI was created but it is plausible that sometimes a piece of work could be discussed long before it is logged in Jira. Similarly, a ticket can be resolved only to find out the resolution was insufficient, and more work is needed. Jira captures only the date of first resolution [14]. We removed subtasks, Epics, and certain resolution types but these may have been misclassified by the original user who was more interested in getting their work done than data integrity. Further investigations will be required into the Public Jira Dataset to analyze the accuracy of the data captured.

7 Conclusion

The research question considered whether Agile systems are stable from queueing perspective. A novel metric was developed to test this hypothesis and a large historic dataset was used to test the Stability Metric. While it is not known how many of the Jira Projects sampled used Agile frameworks such as Scrum and Kanban, it is likely that many did. Based on this assumption, the data presented show that systems are often unstable with large and growing product backlogs. A potential cause of this is likely service rates that are too slow for the individual queue. Further research is required to investigate in more detail but based on this analysis it appears that Agile software development systems are neither inherently stable nor sustainable from a human or business perspective but can, under certain conditions, be made so.

References

1. United Nations:Sustainability. https://www.un.org/en/academic-impact/sustainability. Accessed 5 Apr 2023
2. Measey, P., et al.: Agile Foundations: Principles, Practices and Frameworks, pp. 125–162. BCS, Swindon (2015)
3. Fowler, M., Highsmith, J.: The agile manifesto. Softw. Dev. 9(8), 28–35 (2001)
4. Bose, S.: An Introduction to Queueing Systems. pp. 17–22. Kluwer Academic/Plenum, New York (2002)
5. Montgomery, L., Luders, C., Maalej, W.: An alternative issue tracking dataset of public Jira repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories, pp. 73–77 (2022). https://doi.org/10.1145/3524842.3528486
6. Kleinrock, L.: Queueing Systems, vol. 1, pp. 17–19. John Wiley & Sons, New York (1975)
7. Kupiainen, E., Mäntylä, M., Itkonen, J.: Using metrics in agile and lean software development – a systematic literature review of industrial studies. Inf. Softw. Technol. **62**, 143–163 (2015). https://doi.org/10.1016/j.infsof.2015.02.005
8. Vacanti, D.: Actionable Agile Metrics for Predictability, pp. 41–54. Actionable Agile Press, LeanPub (2015)
9. Fuior, F.: Key elements for the success of the most popular Agile methods. Revista Română de Informatică şi Automatică 29(4), 7–16, (2019). https://doi.org/10.33436/v29i4y201901
10. Conboy, K., Fitzgerald, B.: Toward a conceptual framework of agile methods: a study of agility in different disciplines. In: Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research, pp. 37–44 (2004). https://doi.org/10.1145/1029997.1030005
11. What is a Jira Software Project. https://support.atlassian.com/jira-software-cloud/docs/what-is-a-jira-software-project/. Accessed 5 Apr 2023

12. What is an epic. https://support.atlassian.com/jira-software-cloud/docs/what-is-an-epic/. Accessed 5 Apr 2023
13. Create an issue and a sub-task. https://support.atlassian.com/jira-software-cloud/docs/create-an-issue-and-a-sub-task/. Accessed 5 Apr 2023
14. Defining resolution field values. https://confluence.atlassian.com/adminjiraserver/defining-resolution-field-values-938847105.html/. Accessed 5 Apr 2023