Journal of Network Communications and Emerging Technologies (JNCET)
Volume 13, Issue 4, October (2025)

Addressing Requirement Starvation in Agile Approach: The IWRS Technique

Rasha A. Bin-Thalab
College of Engineering and Petroleum, Hadhramout University, Mukalla, Hadhramout, Yemen.
r.binthalab@hu.edu.ye

Ahmed Bakodah
Faculty of Computer Science and Engineering, Al-Ahgaff University, Mukalla, Hadhramaout, Yemen.
ahmedbakodah@hotmail.com

Hamzah Alaidaros
Faculty of Computer Science and Engineering, Al-Ahgaff University, Mukalla, Hadhramaout, Yemen.
hamzah@ahgaff.edu

ABSTRACT

Requirement prioritization (RP) is a critical phase in software development, ensuring essential features align with business goals and enhance customer satisfaction. However, traditional RP methods often cause "starvation," where low-priority requirements are indefinitely delayed. To address this issue, this paper introduces the Integrating WhaleRank and Starvation (IWRS) technique, tailored for Agile development, especially within the Scrum framework. IWRS enhances the WhaleRank algorithm by adding a starvation mitigation component that gives increased weight to older, repeatedly postponed requirements. This promotes a fairer distribution of development efforts. The technique uses a multifaceted evaluation framework that includes dictionary-based requirement analysis, pairwise similarity scoring, managerial input, and update frequency. This detailed assessment improves the accuracy and relevance of prioritization. Prioritized requirements are organized into sprints using predefined parameters, streamlining planning and execution. Empirical tests confirm that IWRS effectively reduces starvation while preserving the strengths of WhaleRank. It maintains a prioritization accuracy of 82.6% and a Normalized Disagreement (NDA) of 16.2%, demonstrating its reliability.

Index Terms – Agile, Project Management, Requirement Prioritization, Starvation, Software Development, Iterative Development.

1. INTRODUCTION

Software Development Organizations (SDOs) apply existing software development approaches to manage the development process of software projects. A software development strategy is the division of software development operations into discrete phases that contain tasks aimed at enhancing planning and management [1]. However, some projects are frequently missing of budget and schedule. Because of this, two well-known methods to build software projects have been developed since the beginning of information technology (IT): the traditional approach and the Agile method [2].

The traditional approach—commonly referred to as the Traditional Software Development Life Cycle (SDLC)—follows a structured, sequential process characterized by well-defined phases, comprehensive project plans, thoroughly documented customer requirements, detailed product designs, technical specifications, and systematic testing procedures [2]. The outputs of each process group serve as inputs to subsequent phases, creating strong interdependencies that render requirement modifications increasingly difficult as the project progresses [2].

The Agile methodology, also known as Agile Project Management (APM), on the other hand, uses an iterative process that allows different software components to be designed, developed, and deployed simultaneously. Under the direction of self-organizing teams, this procedure is conducted in a highly collaborative setting [3]. Agile approaches, such as Kanban and Scrum, can produce high-quality software in a cost-effective and timely manner that meets the changing requirements of stakeholders [2], [4], [5]. The Agile approach has gained wide adoption within SDOs and other business organizations due to its flexibility, effectiveness, and ability to provide a shorter development cycle with higher customer satisfaction [6].

Requirement Prioritization (RP) is a critical phase in software development that significantly impacts project success. Despite numerous existing techniques, the field encounters persistent challenges. Primarily, human intervention [7] in the prioritization process is time-consuming and prone to subjectivity. Moreover, traditional methods often struggle with scaling to large requirement

sets [8] and ensuring accurate prioritization order. A further limitation is the tendency to neglect low-priority requirements, a phenomenon known as starvation [9].

A starvation problem typically arises in contexts where iteration cycles are short [9]. In general, starvation occurs when a requirement (such as a feature, bug fix, or patch) fails to receive the necessary resources to be completed within the expected timeframe. This issue is predominantly associated with Agile-based Software Development Organizations (SDOs), owing to the inherently brief duration of Agile iterations [9]. Within such iterations, development teams are compelled to address higher-priority items first, as dictated by the iteration's time constraints. Consequently, lower-priority items are repeatedly deferred and may remain unaddressed until all higher-priority tasks have been completed.

In the realm of operating system scheduling, Silberschatz et al. presented a strategy to mitigate process starvation, particularly for low-priority tasks that may experience prolonged delays. Silberschatz [1][10] stated that aging involves gradually increasing the priority of processes that wait in the system for a long time. Another solution was provided by [11] in the context of real time systems. The solution is to combine round robin with priority algorithm to handle this issue by giving each process a slot of time that occupies it to execute their job. Rahim [9] conducted a case study using the Redmine issue-tracking system dataset to investigate whether starvation occurs within software development processes. The analysis revealed that starvation can indeed arise in software development tasks more generally, not only in isolated cases. Furthermore, Rahim [9] notes that the first remedy proposed by [10] is effective in mitigating this issue. To operationalize this remedy, the study incorporates an aging-in-months factor, assigning it a specific score and weight to help manage and control starvation situations. When the authors in [9] solves the starvation problem in Agile, it depends on aging in month to control this issue, so the aging in month is too large especially in short term or midterm projects and the arrival time is one of better solution as [10] states.

To address these shortcomings, a robust and scalable RP approach is necessary. Existing techniques, while offering valuable insights, have not fully resolved these issues. This study identifies a gap in the current research and positions the WhaleRank [12] method as a promising foundation for developing an improved prioritization technique. By overcoming the limitations of WhaleRank and incorporating a starvation mitigation strategy, this study aims to integrate the WhaleRank technique and starvation in order to develop an alternative technique called in short (IWRS) to be explicitly implemented in the Agile approach. This technique prioritizes the product backlog on Agile approach generally and on scrum method particularly. Mainly, the major contribution is to overcome the starvation problem that produced by the Agile nature of the iteration short time.

The proposed technique has been evaluated based on its ability to reduce human intervention, handle large requirement sets, improve prioritization accuracy, and effectively address the starvation problem. By achieving these objectives, this research contributes to the advancement of software development methodologies and enhances the overall quality of software products.

The primary contribution of this research is the development of the Integrating WhaleRank and Starvation (IWRS) technique for effective requirement prioritization within APM. The IWRS technique addresses the challenge of requirement starvation by incorporating existing prioritization aspects. This research has the following two main objectives:

1. RO1: To develop techniques for mitigating starvation in requirements prioritization within Agile sprint iterations.
2. RO2: To evaluate the impact of starvation handling techniques on requirements ranking algorithm performance across Agile sprint iterations.

The subsequent sections of this paper are organized as follows: Section 2 provides a preliminary overview of the foundational work upon which the integration of starvation mitigation is based. Section 3 presents a comprehensive review of existing literature pertaining to requirement prioritization. Section 4 details the methodology employed in this research. Section 5 offers an in-depth analysis of the results and their implications. Finally, Section 6 summarizes the key findings, contributions, and potential directions for future research.

2. PRELIMINARIES

This section describes the background of the Whale Rank algorithm, providing necessary context for the paper to be self-contained.

WhaleRank method served as the foundation for our suggested starvation-aware enhancement. The WhaleRank [12] algorithm is classified as an optimization method designed to prioritize project requirements. It employs four functions for ranking:

2.1. Dictionary Words

This function, as shown in Function rank1, identifies specific keywords in a dictionary to assess the significance of requirements. If a requirement includes these keywords, it receives a higher priority.

Function rank1(All_Req, dictionary)

FOR EACH requirement IN requirements:

score = 0

FOR EACH word IN requirement:

IF word IN dictionary:

score = score + dictionary[word]

rank1[requirement] = score

2.2. Similarity

This function, as shown in Function rank2, calculates the similarity ratio between requirements using the cosine function to measure the distances between them.

Function rank2(All_Req)

FOR EACH r IN All_Req**:**

rank2[r] = 0

FOR EACH r1 IN All_Req**:**

FOR EACH r2 IN All_Req**:**

similarity = COSINE_SIMILARITY(r1,r2)

IF similarity > 0.5**:**

rank2[r1] = rank2[r1] + similarity

rank2[r2] = rank2[r2] + similarity

2.3. Manager's Scale

This subjective scale, as shown in Function rank3, reflects the project manager's perspective on the importance of each requirement.

Function rank3(All_Req, managerPerception)

FOR EACH r IN All_Req:

rank3[r] = managerPerception[r]

2.4. Update Ratio

This function, as shown in Function rank4, considers how frequently a requirement is updated; the more often it is updated, the more critical it is deemed.

Each requirement is associate with update history with urgency level of each update. a fixed scale (e.g., 1-3, minor, moderate, and Critical) and have stakeholders assign an urgency level to each requirement update during the update process.

Function rank4 (All_Req)

FOR EACH r IN All_Req**:**

Rank4[r] = calc_Importance (r, r.up_hist)

Return rank4

Function calc_Importance(r , history)

total_level = 0

FOR EACH u IN All_history**:**

Total_level = Total_level + u.level

avg = total_level/count(history)

Importance = count(history) + avg

RETURN importance

The four ranking functions are combined using ranking constants into a linear rank function as illustrated in Equation (1).

$$R = \alpha\, rank1 + \beta\, rank2 + \gamma\, rank3 + \eta\, rank4 \quad (1)$$

The authors [12] used the Whale Optimize Algorithm (WOA)[13] to optimize ranking weights $\alpha, \beta, \gamma, and\ \eta$. Thereby enhancing requirement prioritization and ultimately contributing to improved product quality. The WOA[13] is a population-based metaheuristic algorithm inspired by the hunting behavior of humpback whales. The algorithm's efficacy stems from its global search capabilities, focused exploration of promising solution spaces, and a balanced approach to exploration and exploitation

Although the whale algorithm demonstrates effectiveness, it encounters the challenge of starvation, wherein certain requirements consistently receive low priority, especially within agile methodologies. To address this problem, we propose the incorporation of a requirement age factor, which will impact the ranking of requirements and help alleviate starvation.

3. RELATED WORK

Beyond to the base work [12], this section discusses other requirements prioritization techniques that have been proposed and used in both traditional and Agile software development approaches. One of the widely adopted techniques is the Analytic Hierarchy Process (AHP) [9], [14], which is a generic decision-making technique. AHP performs pairwise comparisons on hierarchically classified requirements to determine their priority. While AHP is customizable and uses multiple aspects, it has been criticized for being time-consuming and lacking scalability [15].

Another technique is the Case-Based Rank (CBRank)[16], [17], which is based on machine learning theory. CBRank is capable of inferring requirement priorities from incomplete preference information supplied by stakeholders. While the method has demonstrated strong accuracy, its performance has been shown to decline in complex, real-world environments. Drank [17], by contrast, is a semi-automated prioritization technique that incorporates both stakeholder preferences and inter-requirement dependencies. It uses a modified version of the PageRank algorithm to calculate the dependency between requirements. DRank supports contribution and business dependencies, but is limited in the types of dependencies it can handle.

The RIZE technique [9] was developed to address the starvation problem in Agile software development. RIZE considers aspects such as business value, cost, risk, volatility, and age to prioritize requirements. While RIZE is customizable, easy to use, and has low time complexity, it requires significant human interaction and does not use any heuristics or intelligence, resulting in the same prioritization output for each run. Model-Based Requirements Prioritization (MBRP) technique [18] also utilizes a modified PageRank algorithm to prioritize requirements based on their interdependencies and conflict-free functional relationships. MBRP considers aspects like manager perception, risk, cost, and business value.

RP remains an active area of research, as evidenced by recent studies [19], [20], [21], [22]. Machine learning and explainable AI have been employed to analyze user requirements in popular video conferencing applications and subsequently prioritize them, offering valuable insights for application enhancement [22]. Moreover, AI-driven approaches, specifically those leveraging agents and prompt engineering, have been introduced to automate the RP process within Agile frameworks [20]. The dynamic nature of prioritization criteria across development stages has also been investigated, highlighting the need for adaptive prioritization strategies [21]. Table I provides a comparative analysis of existing approaches and the proposed technique.

4. METHOGOLOGY

This paper employs the Design Science Research Methodology DSRM [23]. DSRM that consists of five phases, which are (1) Problem Awareness, (2) Suggestion, (3) Development, (4) Evaluation, and (5) Conclusion. While section 1 explains the first phase, this section will describe the second and third phases of the study. The subsequent two sections will then explain the evaluation and conclusion, respectively. As illustrated in Figure. 1, the proposed technique consists of five key aspects. Four of these aspects are adopted from the base work, including (1) dictionary words, (2) similarity measure, (3) perception of the manager, and (4) newly updated requirements. The fifth aspect is the proposed 'aging' technique, which addresses the starvation problem and is based on the arrival time of the requirements.

Figure 1 shows that the proposed technique consists of a "Requirement set" that stores all the requirements in a database table, including attributes like manager perception, number of updates, dictionary words, similarity, and arrival time. These requirements are then processed to prioritize and construct the product backlog and sprint backlog.

Table 1 A Comparison of Related Works

N.	Prioritization technique	Criteria							Involved Aspects
		Easy to Use	Customizable	Scalability	Time Complexity	Dependency Resolving	Accuracy	Starvation Prevention	
1	[14]	Yes	Yes	No	High	No	Yes	No	Use multiple aspects
2	[16]	Yes	No	No	Low	No	Yes	No	Manager perception and requirements ordering approximations
3	[17]	Yes	No	Yes	Low	Yes	Yes	No	Manager perception
4	[12]	Yes	No	Yes	Low	No	Yes	No	Dictionary word, cosine similarity, Manger perception and newly updated requirement
5	[9]	Ye	Yes	Yes	Low	No	Yes	Yes	Priority score, aging score, aging weight. Cost, risk and business value
6	[18]	Yes	No	No	Low	Yes	No	No	Uses manager perception, risk, cost and business value
7	[12]	Yes	Yes	Yes	Low	No	Yes	No	Dictionary word, cosine similarity, manager perception, new requirement updates
#	IWRS	Yes	Yes	Yes	Low	No	Yes	Yes	Dictionary word, cosine similarity, manager perception, new requirement updates, aging (arrival time).

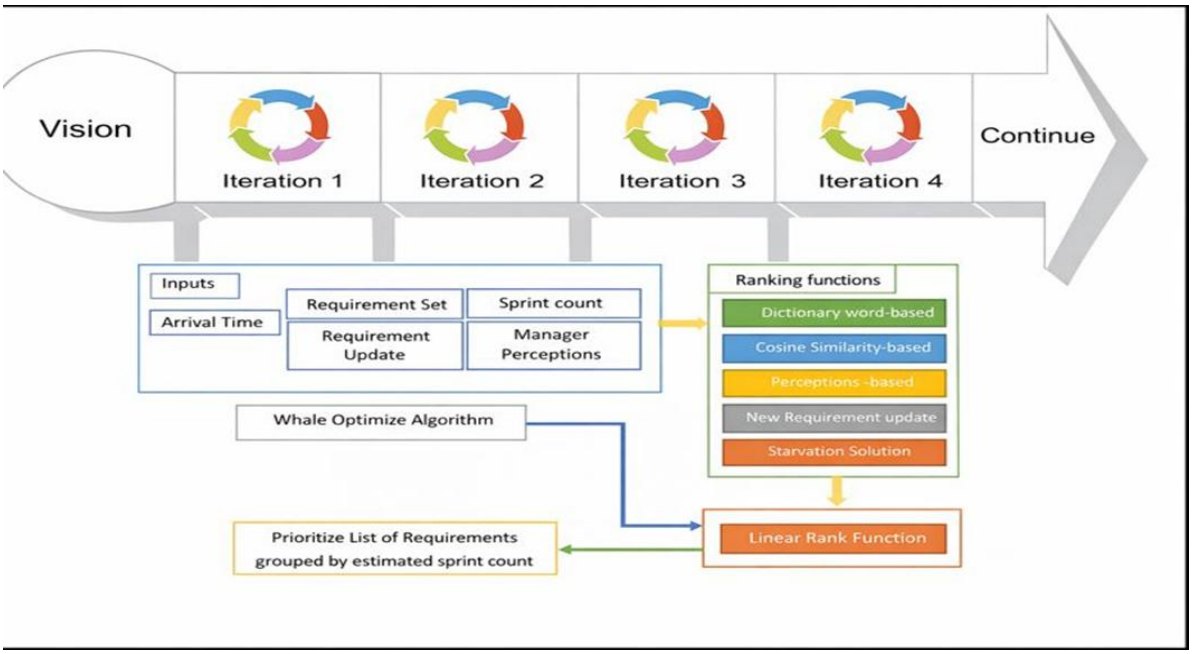

Figure 1 Integrating WhaleRank and Starvation (IWRS) Technique

The technique uses five ranking functions to prioritize the requirements. The first four functions are adopted from the WhaleRank method [12] - dictionary-based ranking, pair-to-pair similarity measure, manager perception ranking, and ranking based on the number of updates.

The fifth ranking function (R5) - illustrated in Equation (2) - is a new "Starvation Solution" that prioritizes requirements based on their arrival time to address the starvation problem.

$$R5(r_i) = current\ time(r_i) - arrival\ time(r_i) \quad (2)$$

The final function is given illustrated in Equation (3):

$$R = \alpha R1 + \beta R2 + \gamma R3 + \eta R4 + R5 \quad (3)$$

If two requirements have the same priority, the one with the earlier arrival time is treated first, unless the product owner decides otherwise. The prioritization process is run at the beginning of each sprint to generate a prioritized list for the current sprint and an estimated list for the next sprints.

To prevent requirement starvation, the proposed system prioritizes requirements with identical total ranks based on their submission time, with older requirements given precedence.

Starvation can occur either within a single sprint or across multiple sprints. The latter is generally easier to mitigate, as Scrum treats each sprint as an independent unit that requires all selected requirements to be completed before the sprint ends, unless the product owner decides otherwise. At the start of every sprint, the product backlog is updated, and the sprint backlog for the upcoming iteration is created.

As illustrated in Figures 1, the proposed technique is applied at the start of each sprint to refine the product backlog by elevating the highest-ranked requirements to the top. Following this refinement, the project team selects the most appropriate requirements from the updated backlog to construct the sprint backlog for the upcoming iteration. The product owner retains responsibility for product backlog organization; hence, the proposed technique offers supplementary support.

Algorithm 1 presents the steps of the proposed method for prioritizing software requirements across multiple sprints within an iterative Agile development process. The algorithm is depicted also visually in Figure 3.

Inputs:

- All_Req: List of all requirements
- dictionary: Words with associated weights
- managerPerceptions: Manager perceptions per sprint
- n: Total number of sprints

Initialization:

1. Set i = 1
2. Set previousRankings = []

Process:

While i ≤ n do:

1. SR[i].Req ← Get_Req_Sprints(i, All_Req)

2. SR[i].Rank ← GetRank(SR[i].Req, dictionary, managerPerceptions[i])

3. i ← i + 1

End While

Output:

Return SR.Rank: A list of sprint rankings.

Function GetRank(sprintRequirements, dictionary, managerPerception)

Initialize variables r1= r2= r3= r4= r5 = []

For each q in sprintRequirements:

Compute r1(q), r2(q), r3(q), and r4(q) (see Section 2).

Compute r5(q) = *current time(q) – arrival time(q)*

Apply WOA to determine the value of α, β, γ, η

Rank(q) =α r1[q] + β r2[q] + γ r3[q] + ηr4[q] + r5[q]

APPEND (Finalrank, Rank(q)))

return FinalRank

Algorithm 1 IWRS Method

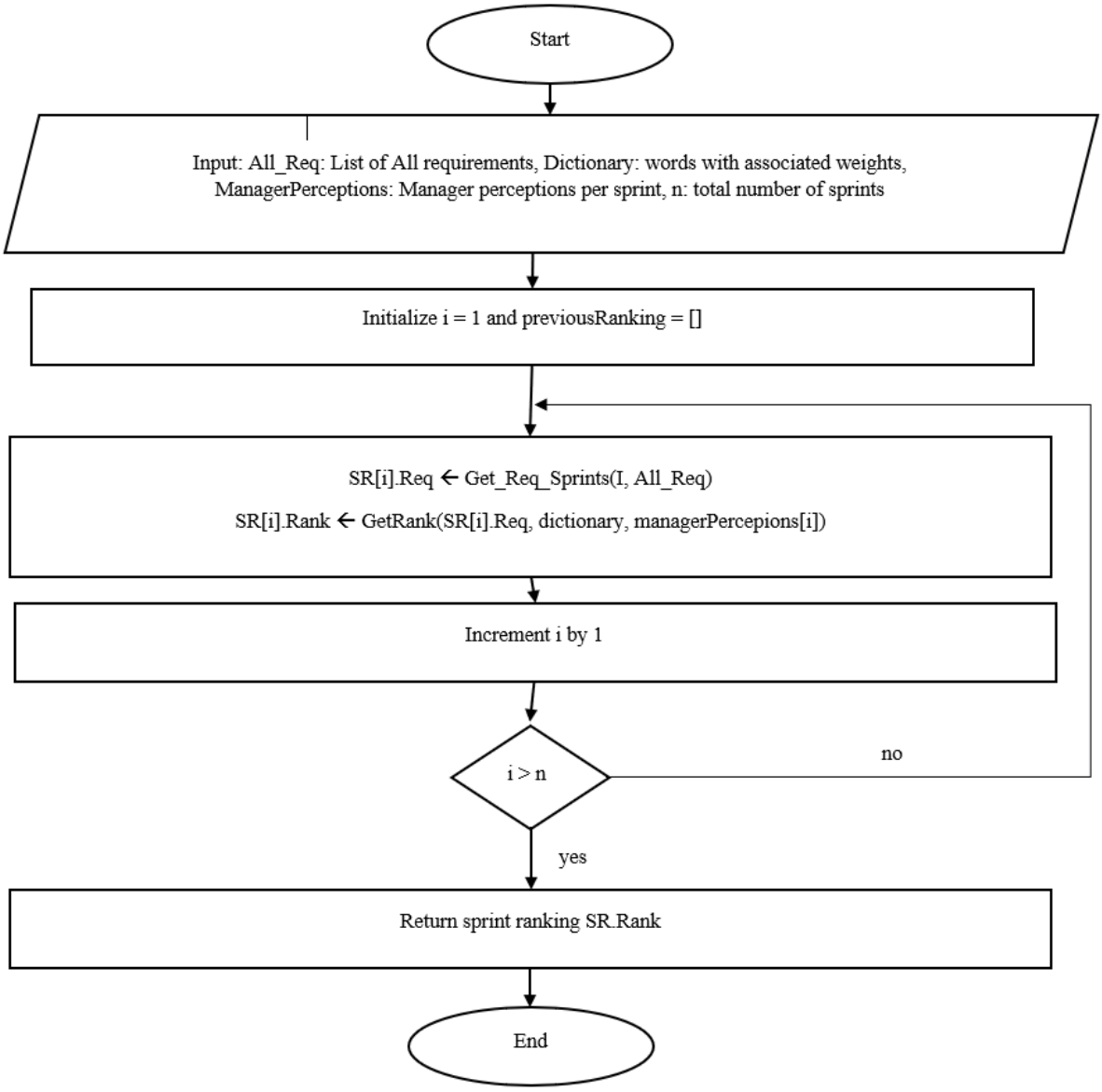

Figure 2 Flowchart of the Algorithm 1 Steps

5. EXPERIMENT

In this section, we address the experiments that are used to assess the ranking performance of the proposed IWRS. The experimental evaluations were performed using the .NET framework on a personal workstation operating under Windows 10, featuring an Intel Core i7 CPU and 8 GB of memory. The CM1 data are the dataset in use.

The dataset utilized in this study is the WhaleRank dataset, an accessible dataset for software requirement specifications of the Data Processing Unit (DPU) for the X-Ray Detector Explorer, made available by X-gamma Company (http://promise. site. uottawa. ca/SERepository/). This dataset, which consists of 235 requirements and their revised versions, was selected as a standard to appraise the efficacy of our proposed technique, referred to as IWRS. We leveraged the dataset's intrinsic structure, which features the categorization of requirements into 24 sprints, to simulate a realistic agile development setting. To evaluate the performance of IWRS, we executed the following assessment: First, we employed IWRS to rank the requirements within each sprint, considering the arrival time of each requirement to tackle potential starvation concerns. We subsequently contrasted the rankings produced by IWRS with the initial sprint allocations provided in the WhaleRank dataset. This comparison enabled to gauge the precision of IWRS technique in aligning with the prioritization established in the dataset. The categorization of requirements into sprints significantly influenced our evaluation by offering a framework to examine the algorithm's performance in a time-sensitive, iterative development scenario.

5.1. Evaluation

This paper evaluates three main metrics:

5.1.1. Requirement Advancement

This metric measures the change in a requirement's position within the product backlog after applying the proposed IWRS technique, compared to its original position. It is calculated as the difference between the old and new positions.

And the requirement advance is calculated as shown in Equation (4):

$$R_{advanc}(R_i) = P_o(R_i) - P_n(R_i) \quad (4)$$

Where:

$R_{advanc}(R_i)$ is requirement advance.

$P_o(R_i)$ is the old position of requirement in the previous ranked list

$P_n(R_i)$ is the old position of requirement in the previous ranked list

The $R_{advanc}(R_i)$ metric provides a measure of requirement performance. A positive value indicates advancement, a zero value indicates no change, and a negative value indicates a decline in priority, which is considered a performance deficit

5.1.2. Accuracy

This metric, as shown in Equation (5), measures how accurately the computed rank order of requirements matches the original rank order. It returns a value of 1 if the orders match, and 0 if they don't, averaged across all requirements.

$$Accuracy = \frac{1}{q} \sum_{i=1}^{q} \sigma_i \quad (5)$$

$$\sigma_i = \begin{cases} 1, & R(P_i) = R(Q_i) \\ 0, & R(P_i) \neq R(Q_i) \end{cases}$$

Where:

q is the total number of requirements,

R(Pi) and *R(Qi)* are the rank of the requirement relations *P* and *Q* respectively.

5.1.3. Normalized Disagreement (NDA)

This metric [16], as shown in Equation (6), measures the degree of disagreement between the computed and original order relations for the same set of requirements. It ranges from 0 (no disagreement) to 1 (maximum disagreement). So, when measure NDA, the less value is better than greater. For example, when we reach the value 18% is considered as reasonable NDA value.

$$NDA = \frac{1}{U_d} \sum_{j=1}^{q} \sum_{k=1}^{q} D_{P,Q}(r_j, r_k) \quad (6)$$

Where

NDA is normalized disagreement and its value $0 \leq NDA \leq 1$

U_d is the original rank set.

P and Q are the order relation.

j, *k* and *q* are the number requirements in the input

$D_{P,Q}(r_j, r_k)$ is the disagreement between requirement pair as illustrated in Equation (7)

$$D_{P,Q}(r_j, r_k) = \begin{cases} 1 \; if \; P(r_j) < P(r_k) \text{ and } Q(r_j) > Q(r_k) \\ \quad or \\ \quad P(r_j) > P(r_k) \text{ and } Q(r_j) < Q(r_k) \\ 0 \quad otherwise \end{cases} \quad (7)$$

6. RESULTS AND DISCUSSIONS

We depict the effectiveness of proposed IWRS in how it handles the starvation problem and preserve the accuracy and disagreement of the WhaleRank.

6.1. Requirements Advance Analysis

Figure 3 shows the requirements advance and how position of requirement has changed. The most requirements changes were in requirement 8 and 64 where it advance by 5 positions. The requirements 2, 6 and 66 were advanced by 4 positions. Whereas requirements 62, 63, 68, 79, 100 have changed by 3 positions. Finally, requirements 7, 10, 25, 30, 39, 44, 47, 69 and 119 changed by only 2 positions.

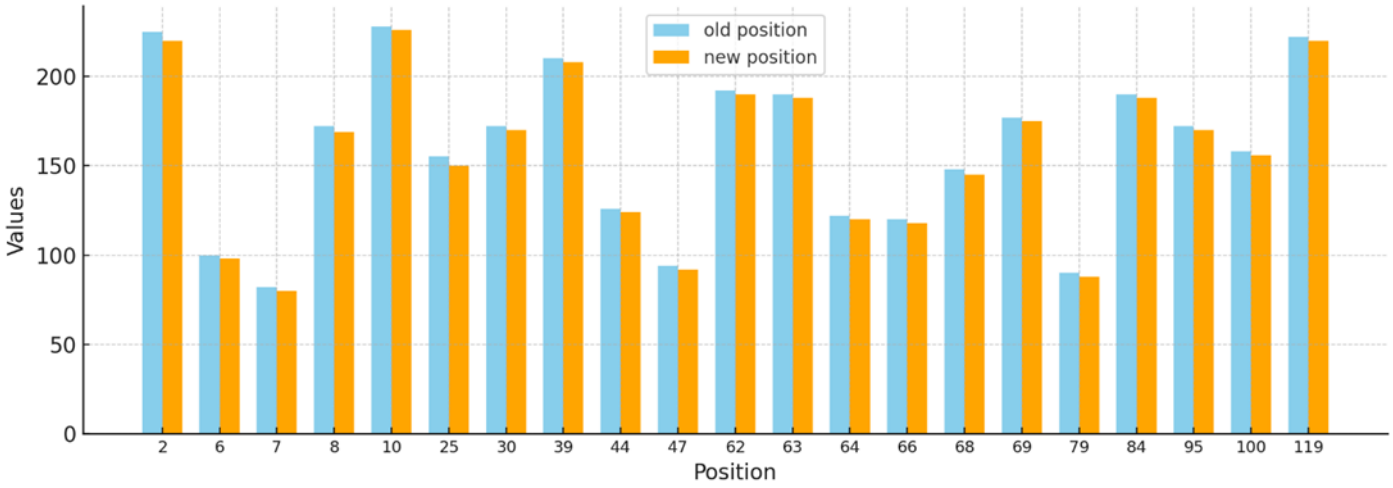

Figure 3 Requirements Advance Per Sprint

Table 2 provides a quantitative assessment of requirement improvements achieved through the IWRS technique. The table utilizes a structured layout to specify particular alterations in requirement positioning and scheduling.
The information is arranged into six columns:

1) Requirement: This column offers a distinct identifier for each requirement, aiding clear reference and tracking.
2) Advance: This column measures the extent of each requirement's positional progression, indicated as the number of positions advanced. This metric serves as a direct gauge of the effect of the IWRS technique on requirement prioritization or scheduling.
3) Old Position: This column specifies the initial location of each requirement before the implementation of the IWRS technique. This acts as a reference point for assessing positional change.

4) New Position: This column shows the updated location of each requirement following the implementation of the IWRS technique. Comparing it with the "Old Position" column allows for accurate identification of the positional shift.

5) Old Sprint: This column specifies the initial sprint allocation for each requirement. This aspect is essential for grasping changes in project scheduling.

6) New Sprint: This column indicates the updated sprint allocation for each requirement after the implementation of the IWRS technique. Differences between the "Old Sprint" and "New Sprint" columns reflect the technique's effect on sprint planning.

Table 2 IWRS Impact on Requirements Advancements

R#	Advance	Old position	New position	Old sprint	New sprint
2	4	226	222	23	23
6	4	102	98	11	10
7	2	83	81	9	9
8	5	172	167	18	17
10	2	229	227	23	23
25	2	155	153	16	16
30	2	175	173	18	18
39	2	214	212	22	22
44	2	127	125	13	13
47	2	95	93	10	10
62	3	193	190	20	19
63	3	194	191	20	20
64	5	124	120	13	12
66	4	123	119	13	12
68	3	145	142	15	15
69	2	180	178	18	18
79	3	90	87	9	9
95	1	171	170	18	17
100	3	159	156	16	16
119	2	225	223	23	23

The information within Table 2 illustrates the ability of the IWRS technique to bring about both positional and temporal transformations in project requirements. Significantly, the "Advance" column emphasizes differences in the degree of positional advancements among various requirements. Additionally, the comparative examination of "Old Sprint" and "New Sprint" assignments uncover the technique's effect on project timeline management.

Of the 224 requirements evaluated, 20 (around 9%) were influenced by the implementation of the IWRS technique. Among this group, 14 requirements saw changes in their positions within their corresponding sprints, while the other 6 requirements had

alterations in both their position and sprint assignment. In APM, these adjustments to requirements are deemed essential for reducing starvation.

6.2. Accuracy and NDA Analysis

Table 3 shows the results of the suggested method's accuracy and NDA, which are 83% and 16%, respectively, preserving the same value as the base work [12]. Table III also shows how the values of accuracy and NDA vary slightly with population size.

Table 3 Accuracy and NDA Results

Method	Population Size				Population Size			
	4	8	12	16	4	8	12	16
	NDA (%)				Accuracy (%)			
WhaleRank	16	16	16	16	83	83	83	83
IWRS	16	16	16	16	83	83	83	83

7. CONCLUSION

This paper developed a system to handle starvation in Agile project management by combining the strengths of the WhaleRank algorithm with a novel starvation mitigation strategy. This development process influenced 9% of the whole requirements, necessitating changes to their positions and sprint assignments. Furthermore, the starvation handling mechanisms implemented did not negatively impact the performance of requirements prioritization techniques, as evidenced by the comparable accuracy and NDA values to the baseline work. To further refine the IWRS technique, future research should aim to broaden the dataset to encompass diverse project contexts and undertake comprehensive case studies to evaluate its effectiveness across different organizational environments. Additionally, exploring alternative prioritization criteria and optimization algorithms could lead to further improvements.

REFERENCES

[1] S. Pargaonkar, “A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages, Disadvantages, and Application Suitability in Software Quality Engineering,” International Journal of Scientific and Research Publications (IJSRP), vol. 13, no. 08, pp. 345–358, 2023.
[2] I. Sommerville, Software Engineering, 10th ed. United Kingdom: University of Lancaste, 2016.
[3] H. Alaidaros and M. Omar, “Software project management approaches for monitoring work-in-progress: A review,” Journal of Engineering and Applied Sciences, vol. 12, no. 15, pp. 3851–3857, 2017.
[4] H. Alaidaros, M. Omar, and R. Romli, “An improved model of Agile Kanban method: verification process through experts’ review,” International Journal of Agile Systems and Management, vol. 13, no. 4, pp. 390–416, 2020.
[5] S. G. Tetteh, “Empirical Study of Agile Software Development Methodologies: A Comparative Analysis,” Asian Journal of Research in Computer Science, vol. 17, no. 5, pp. 30–42, 2024.
[6] C. V. Ibeh, K. F. Awonuga, U. I. Okoli, C. U. Ike, N. L. Ndubuisi, and A. Obaigbena, “A review of agile methodologies in product lifecycle management: bridging theory and practice for enhanced digital technology integration,” Engineering Science & Technology Journal, vol. 5, no. 2, pp. 448–459, 2024.
[7] S. U. R. Khan, S. P. Lee, M. Dabbagh, M. Tahir, M. Khan, and M. Arif, “RePizer: a framework for prioritization of software requirements,” Frontiers of Information Technology & Electronic Engineering, vol. 17, no. 8, pp. 750–765, 2016.
[8] M. I. Babar, M. Ramzan, and S. A. K. Ghayyur, “Challenges and future trends in software requirements prioritization,” in International conference on computer networks and information technology, IEEE, 2011, pp. 319–324.
[9] M. S. Rahim, A. Z. M. E. Chowdhury, and S. Das, “Rize: A proposed requirements prioritization technique for agile development,” in 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), IEEE, 2017, pp. 634–637.
[10] A. Silberschatz, P. B. Galvin, and G. Gagne, Operating system principles. John Wiley & Sons, 2006.
[11] I. S. Rajput and D. Gupta, “A priority based round robin CPU scheduling algorithm for real time systems,” International Journal of Innovations in Engineering and Technology, vol. 1, no. 3, pp. 1–11, 2012.
[12] R. V. Anand and M. Dinakaran, “WhaleRank: an optimisation based ranking approach for software requirements prioritisation,” Int. J. Environ. Waste Manag., vol. 21, no. 1, pp. 1–21, 2018.
[13] S. Mirjalili and A. Lewis, “The whale optimization algorithm,” Advances in engineering software, vol. 95, pp. 51–67, 2016.
[14] M. Sadiq, S. Ghafir, and M. Shahid, “An approach for eliciting software requirements and its prioritization using analytic hierarchy process,” in 2009 international conference on advances in recent technologies in communication and computing, IEEE, 2009, pp. 790–795.
[15] N. H. Borhan, H. Zulzalil, and N. M. A. Sa’adah Hassan, “Requirements prioritization techniques focusing on agile software development: a systematic literature,” International Journal of Scientific and Technology Research, vol. 8, no. 11, pp. 2118–2125, 2019.
[16] A. Perini, A. Susi, and P. Avesani, “A machine learning approach to software requirements prioritization,” IEEE Transactions on Software Engineering, vol. 39, no. 4, pp. 445–461, 2012.

[17] F. Shao, R. Peng, H. Lai, and B. Wang, "DRank: A semi-automated requirements prioritization method based on preferences and dependencies," Journal of Systems and Software, vol. 126, pp. 141–156, 2017.
[18] M. Abbas, I. Inayat, N. Jan, M. Saadatmand, E. P. Enoiu, and D. Sundmark, "MBRP: model-based requirements prioritization using PageRank algorithm," in 2019 26th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2019, pp. 31–38.
[19] E. Alhenawi, S. Awawdeh, R. A. Khurma, M. García-Arenas, P. A. Castillo, and A. Hudaib, "Choosing a Suitable Requirement Prioritization Method: A Survey," arXiv preprint arXiv:2402.13149, 2024.
[20] M. A. Sami, Z. Rasheed, M. Waseem, Z. Zhang, T. Herda, and P. Abrahamsson, "Prioritizing Software Requirements Using Large Language Models," arXiv preprint arXiv:2405.01564, 2024.
[21] R. B. Svensson and R. Torkar, "Not all requirements prioritization criteria are equal at all times: A quantitative analysis," Journal of Systems and Software, vol. 209, p. 111909, 2024.
[22] S. Bai, S. Shi, C. Han, M. Yang, B. B. Gupta, and V. Arya, "Prioritizing user requirements for digital products using explainable artificial intelligence: A data-driven analysis on video conferencing apps," Future Generation Computer Systems, vol. 158, pp. 167–182, 2024
[23] V. K. Vaishnavi, Design science research methods and patterns: innovating information and communication technology. Auerbach Publications, 2007.

Authors

Rasha A. Bin-Thalab is an associate professor in the Department of Computer Engineering at Hadhramout University, Hadhramout, Yemen. She received her Bachelor's degree from Hadhramout University (2000), her Master's degree from Alexandria University (2007), and her Ph.D. from Cairo University (2014). She has been working in the field since 2014. Dr. Bin-Thalab interests on software engineering, machine learning, and information retrieval.

Ahmed Bakodah is a web developer. He received his bachelor degree in Computer Engineering from the Department of Computer Engineering, Faculty of Engineering, Hadhramout University, Hadhramout, Yemen in 2011. Ahmed holds a master degree in Science (Information Technology) from Al-Ahqaff University, Hadhramout, Yemen in 2023.

Hamzah Alaidaros, PhD, is an assistant Professor at the Faculty of Computer Science and Engineering, Al-Ahgaff University, Mukalla, Hadramaout, Yemen. He received the PhD in Computer Science from the Awang Had Salleh Graduate School (AHSGS), Universiti Utara Malaysia (UUM) in 2020. Hamzah holds a master degree in Science (Information Technology) from UUM since 2015. He received his bachelor degree in Computer Science from Al-Ahgaff University, Hadhramout, Yemen in 2007. His main research interests on software engineering, software project management, and Agile methodology.