Journal of Theoretical and Applied Information Technology
28th February 2023. Vol.101. No 4

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A HYBRID APPROACH FOR REQUIREMENTS PRIORITIZATION IN AN INCREMENTAL DEVELOPMENT MODEL

FATIMA THAHER ABUROMMAN

Al-Balqa Applied University, Princess Rahma University College, Al-Salt, Jordan
e-mail: fatmia_thaher@bau.edu.jo

ABSTRACT

In incremental development model (IDM) stakeholders are involved in the development process to add more emphasis on the system implementation phases rather than the requirements analysis and system design, which in turn may give better results in terms of the delivery of the components of the system which are referred to as increments. Because the development process is shared with stakeholders' experience, interest, positions and other factors, it might not be easy to rely on the stakeholders' opinions on deciding which increment will be the next. This paper provides a solution to this problem by forming a mathematical representation and model that is referred to as the Hybrid Approach (HA) which is a hybridization between the dept-first search (DFS), the value-oriented prioritization (VOP), and the greedy algorithm. The HA model is compared to the analytic hierarchy process (AHP) approach. The results show that the HA model outperforms the AHP approach in terms of runtime.
Keywords: *Increment Prioritization; Incremental Model; Software Engineering; Requirements Prioritization; Hybrid Model; Value-Oriented Prioritization.*

1. INTRODUCTION

Requirement prioritization (RP) is used in managing software development projects to determine the candidate requirements that should be included in a certain release. The aim of RP is to choose the requirements based on their importance with accordance to the stakeholder's understanding or perception [1]. Afterwards, a decision is made on which requirements to implement in each specific release.

Requirements are classified into: (1) functional, (2) non-functional, or (3) domain requirements. Functional requirements describe the behavior of the system, i.e., what the system will do, such as validity checks on the inputs, exact sequence of operations, and error handling and recovery, to name a few.

On the other hand, non-functional requirements (NFRs) are much related to the quality of the software rather than what the software do, such as availability, dependability, and performance [2]. The last type of requirements is domain requirements, which are the requirements that come from the application domain of the system and that reflect characteristics of that domain.

In an incremental development model (IDM), the emphasis is on the development process which mainly comprises the following phases: (1) system increment development, (2) increment validation, (3) increment integration, and (4) system validation.

An increment can be thought of as a subsystem (or a component). Upon the completion of these phases, which incorporate the development of a component, development of another increment can start, and it will follow the very same phases.

However, in IDMs, prioritizing increments is an issue. Prioritizing means to decide which component's development should start before other starting the development of another components.

In this paper, we propose a model in which increments are prioritized based on some factors related to the overall system rather than depending on the stakeholders' needs.

Donald Firesmith in 2004 [3], proposed dimensions to consider when prioritizing, namely, personal preference, business value, harm avoidance, risk, cost, difficulty, time to market, requirements stability, dependencies among

requirements, implementation dependencies, legal mandate, frequency of use, and reuse. The author didn't introduce a formal method to use when prioritizing requirements; the RP process is fully dependent on the consensus between the requirements teams and the stakeholders.

Khari and Kumar [4], compared six techniques for prioritizing software requirements, namely, analytic hierarchy process (AHP), value-oriented prioritization (VOP), cumulative voting (CV), numerical assignment technique (NAT), binary search tree (BST), and planning game (PG). They concluded that VOP is supposed to be the best method for prioritizing software requirements, because of its ease, accuracy, and comfortability to handle.

Furthermore, Azar et al. [5] discussed VOP as a case study on a small development organization. They first identified the core business values of the company and gave each of which a value. Then, they gave each requirement a value of Scale of 1-to-10 against each of the business values. After all, they have a value-oriented prioritization matrix.

There are many researchers who studied RP using algorithmic methods. For instance, Sadiq and Devi [6] present a method for selecting requirements using rough set theory. They tested the applicability of their proposed model using an examination system. The authors concluded that their method captures the exact opinion of the decision makers.

In essence, a software system is decomposed into several subsystems. Recursively, each subsystem is a system by itself. The decomposition is agreed upon between the development team and the stakeholders. In a VOP approach, a VOP matrix is established, such that a subsystem spans along a column of the VOP matrix. The main idea behind VOP is to focus on the core business values that lead to stakeholders' satisfaction while prioritizing the product requirements [5].

In order to enhance the performance of requirements prioritization, the Ahuja et al. [7] proposed a system that uses least-squares-based random genetic algorithm. The system that was proposed by Ahuja et al. uses genetic algorithm which is a metaheuristic approach that is used to solve combinatorial optimization problems [8]. Their main target was to reduce the time consumed in requirements engineering which leads to lower decision-making efforts.

Ali Khan et al. [9] compared different RP techniques, namely, numerical assignment, MoScoW, simple ranking, bubble sort, binary search tree (BST), hundred-dollar method, AHP, Hierarchy AHP, minimal spanning tree, and planning game.

New RP techniques emerge frequently by researchers and scholars, due to the importance of RP in the software development process. Shafiq et al. [10] proposed a natural-language-processing-based (NLP-based) approach for RP. Similarly, Aly et al. [11] devised an approach for safety RP. Also, Roy et al. [12] focused on NFR and presented an approach that takes into consideration the conflicts among NFRs in the RP phase and minimizes the inconsistencies of software development.

In the modern software development arena, Agile methodologies, such as, scrum, extreme programming (XP), and DevOps, are dominating. In fact, incremental development plays the pivotal role in such models [13]. This incremental delivery requires a formal approach for requirements engineering in which RP is based on a well-defined methodology [14]. In this context, this paper is introducing a systematic approach for continuous delivery (CD) of software increments. This approach can be used in DevOps as well as all agile development methodologies. The model that is introduced in this paper is a novel model that hybridizes exhaustive search algorithms, that is depth-first search (DFS), with VOP, and the greedy algorithm in the RP process. The proposed approach fits in DevOps to help in the CD of software components or increments.

One of the main important contributions of this paper is that it intends to prioritize requirements for fast delivery of software components, or increments. Practically, this model incorporates automatic assignment of weights of different system components, rather than using a predetermined, manual weigh assignment that depends on the experience of the development teams as well as the stakeholders, which, in fact, would lead to some inconsistencies.

The rest of this paper is organized as follows: in Sect. 2 the factors of the proposed hybrid approach (HA) are discussed in detail. The HA model is presented in Sect. 3. Section 4 contains the experimental results. Finally, conclusion and future

work are presented in Sect. 5.

2. THE HYBRID APPROACH FACTORS

In the following subsections we discuss the factors that are used to establish the hybrid approach (HA), which is devised in this paper. Basically, HA takes the following factors into consideration: (1) dependencies between components, (2) functional requirements, (3) users, (4) development time, and (5) cost. For the HA to work perfectly, it is assumed that both the requirements of the system are well defined, and the architectural design is set.

2.1. Dependency

The dependency relationship between any two components (or modules) implies that one component, that is the successor, is dependent on another component, that is the predecessor. In fact, dependencies between components (or subsystems) can be determined at early stages of the analysis and design.

The HA model expresses dependencies between different component by means of dependency graphs**.** A dependency graph is a directed graph in which components are represented by vertices, and the dependencies between components are represented by edges [15]. Formally, let A and B be two components of a given subsystem S, then a directed graph $G = (V, E)$ is created to represent the dependency graph of S, and a directed edge e_{AB} exists from vertex A to vertex B, if the component B is dependent on component A, i.e., component A is a predecessor of component B.

In this paper, each component is assigned a weight w, provided that predecessors must be assigned weights higher than successors' weights. In the previous example, this indicates that the weight of component A must be greater than the weight of component B. In this context, a dependency relationship between the two components A and B, in the dependency graph, is expressed as follows: $e_{AB}: A \rightarrow B | w_A > w_B$.

The algorithm shown in Algorithm 1 is a modified DFS algorithm that is used to calculate the weight of each vertex, i.e., component. Algorithm 1 takes one parameter, that is $G = (V, E)$, which is the directed graph which represent the dependency graph of the system S.

In the beginning of Algorithm 1, all vertices are set to -1, which is an equal weight. The output of Algorithm 1 is the dependency graph with all its vertices are assigned weights based on the dependencies between the components. In fact, one main contribution of this paper is the automatic assignment of weights to the system components.

Algorithm 1: Component Dependency Algorithm

Input: $G = (V, E)$ a directed graph of the components of the system
Output: Weights of components

```
1  Dependency(G){
2      //such that G =< V, E > is a directed
       graph
3      //initialize all component weights to -1
4      //V is the set of vertices of graph G
5      for each vertex v ∈ V {
6          weight[v]= -1;
7          parent[v] = NULL;
8          finished[v] = false;
9      }
10     for each vertex v ∈ V
11         if (NOT finished[v])
12             Weight(v);
13     Adjust_roots();
14 }
```

Algorithm 2 shows the logic of computing the weights of different vertices in the directed graph.

While traversing the graph, there may arise some cases in which some vertices do not have predecessors neither do they have any successors, in this case, they will have the following state at the end of traversal: finished = true, parent = NULL and weight = 0.

Algorithm 2: Weight Computation Procedure

Input: The vertex v
Output: Weights of components that are dependent on v

```
1  Weight(v){
2      for each vertex u ∈ adjacent[v]
3          if (NOT finished[u]){
4              parent[u]= v;
5              Weight(u);
6          }
7          Else
8              weight[v] = weight[u];
9          weight[v] = weight[v] + 1;
10         finished[v] = true;
11         if (weight[parent[v]] ≤ weight[v])
12             weight[parent[v]] = weight[v];
13 }
```

Line 13 of Algorithm 1 calls the `Adjust_roots()` procedure which is shown in Algorithm 3. The `Adjust_roots()` procedure finds the vertex which has the maximum weight between root nodes in the graph and sets the weights of all root nodes to the maximum weight that has just been found, thus giving all root components an equal priority.

Algorithm 3: Root Adjustment Procedure

Input: $G = (V, E)$ a directed graph of the components of the system

Output: Adjusted weights of root components

```
1  Adjust_roots(G){
2      max = find_max_weight(V);
3      for each vertex v ∈ V
4          if (parent[v] is null)
5              weight[v]= max;
6  }
```

2.2. Requirements Grouping

A system S can be decomposed into a set of Components $C = \{c_i \mid i = 1, 2, \dots, m\}$. At an earlier stage, a set of functional requirements $R = \{r_i \mid i = 1, 2, \dots, n\}$ were specified. Thus, a component-requirement matrix (CRM), which is an $m \times n$ matrix, is constructed as a relationship between C in the columns and R in the rows, such that the values that are allowed in the CRM are only 1 if the requirement r_i corresponds to the component c_j, or 0 otherwise.

The summation of each column in the CRM is computed. The result is the set $T = \{t_i \mid i = 1, 2, \dots, n\}$ which has a size of n, such that each element of the set T corresponds to one requirement of the set of requirements R.

2.3. Users

Another important contribution to this paper is that it gives weight to the domains of application, i.e., the domain to which the system is to provide services to. The intuition here is that domains are given values with respect to how much they are offered services by the system.

Formally, a system S can offer services or functions to one or multiple domains. Domains of applications are expressed according to their relevancy of the system S as a scale, e.g., from 0 to 5 or from 0 to 10. A domain relevancy matrix $D = \{d_{ij} | i = 1,2, \dots m, j = 1, 2, \dots, k\}$, is an $m \times k$ matrix, that is constructed as a relationship between the components in the rows and the domains in the columns. Each instance of D represents the value that is assigned to the component according to its relevancy with the component. In other words, d_{ij} can range from zero, if the domain is not relevant to the component (or vice versa); the upper bound is the value given to that domain based on the predetermined scale.

Because resulting system is expected to provide services to a predetermined of users, a rank between 1 and 4 is given to each user based on the managerial position. In this paper, users are categorized into four levels: (1) top-level management who ranked 4, (2) mid-level management ranked 3, (3), low-level management with rank 2, and (4) employees who are ranked as 1.

Based on this categorization, the set of users $U = \{u_i | i = 1, 2, \dots, l\}$ contains the users of the system such that each element of the set represents the weight of the user which is computed with respect to domain of application and the desired component, i.e., the domain relevancy matrix, as follows: $w_i = p \times rank_i$, such that p is the average of domain relevancy matrix instances that correspond to the specific user, or in other words, the domains and components that the specific user will be using in the system.

It is worth to mention that categorizing the users based on their managerial positions is another contribution to add to this paper. The idea is that users with higher managerial positions have greater influence on the system development and the requirements engineering process, and as a result RP will be affected by the managerial position of the user.

2.4. Development Time

Each component has an estimated length. Lengths are not necessarily equal. Based on the number of the development team members, the nature of the

software system, its size, availability of resources, and some other factors that might be relevant to one team or another, each development team can preset the maximum time required for each component to finish its development. The overall development time of the system (DT) is the sum of all development times of all the components, $DT = \sum_{i=1}^{m} dt_i$, such that dt_i is the development time of component i.

2.5. Development Cost

The development cost of a component depends on the development time, number of team members, size of the component, and hardware and software resources, to name a few. The total cost of system development is the summation of the costs of development of each component of the system and is computed as follows: $DC = \sum_{i=1}^{m} dc_i$, such that dc_i is the development cost of component i.

3. THE HYBRID APPROACH MODEL

The HA model is a greedy algorithm [16] that uses the greedy approach to select the requirements based on a well-defined model of factor computation as illustrated in detail in this section.

To calculate the component weight, the set W is established as a set of component weights $W = \{w_i \mid i = 1, 2, \dots, m\}$, such that w_i is the initial weight of component i, and m is the number of components (or increments).

Algorithm 4 contains the hybrid approach that is devised by this paper for RP. The algorithm starts by establishing the component dependency graph (CDG) that shows the dependencies between the modules. Then, the dependency algorithm, that is explained in Algorithm 1, is started to compute module weights.

Algorithm 4 establishes the CRM and computes the total requirements of each component T.

Afterwards, Algorithm 4 needs to find out the user value of each module. First, weights of domains of applications are determined by means of the domain relevancy matrix D. The algorithm needs to be fed with the list of users to compute their weights based on the domains of applications and managerial positions weights.

The algorithm computes the total system development time based on the development times of each individual component of the system. and each module development time. It also computes the total system development cost as the summation of the cost of development of each individual component.

When all the factors are computed, Algorithm 4 computes the weight of each component according to Eq. 1.

$$c_i = w_i + t_i + u_i + \frac{dt_i}{DT} + \frac{dc_i}{DC} \qquad \text{Eq. 1}$$

Such that c_i is the component (or increment) weight, w_i is the initial weight as computed from the dependency matrix, t_i is the total requirements of component i, $u_i$is the weight of user i, dt_i is the development time of component i, DT is the total system development time, c_i is the cost of development of component i, and DC is the total cost of development of the system.

Algorithm 4: The HA Algorithm

Input: $G = (V, E)$: a directed graph of the components of the system,
U: the set of users of the system.

Output: Weights of components

```
1   HA(G){
2       Dependency(G);
3       CRM = Create_CRM();
4       T = Compute_total_requirements(CRM);
5       D =
        Create_domain_relevancy_matrix(U);
6       DT = Σ_{i=1}^{m} dt_i ;
7       DC = Σ_{i=1}^{m} dc_i ;
8       for each component c_i ∈ C
9           c_i = w_i + t_i + u_i + dt_i/DT + dc_i/DC ;
10      sort (non-decreasing order) by weights;
11      decide on RP (greedy);
12  }
```

As shown in Eq. 1, the effect of both development time and cost on the RP decision is mitigated by dividing on the total development time and total development cost. This mitigates the influence of predetermined estimation that is set by the development teams, and thus reducing the chances of lack of accuracy of the HA model.

Finally, Algorithm 4 orders the components in a non-increasing order according to their weights. The greedy algorithm is then set to choose items based on their weights.

As shown in Algorithm 4, the HA algorithm is a hybrid approach that uses: (1) DFS, (2) greedy

algorithm, and (3) a customization of the VOP based on customized factors.

Figure 1 depicts a graphical representation of the steps that are incurred by the HA algorithm.

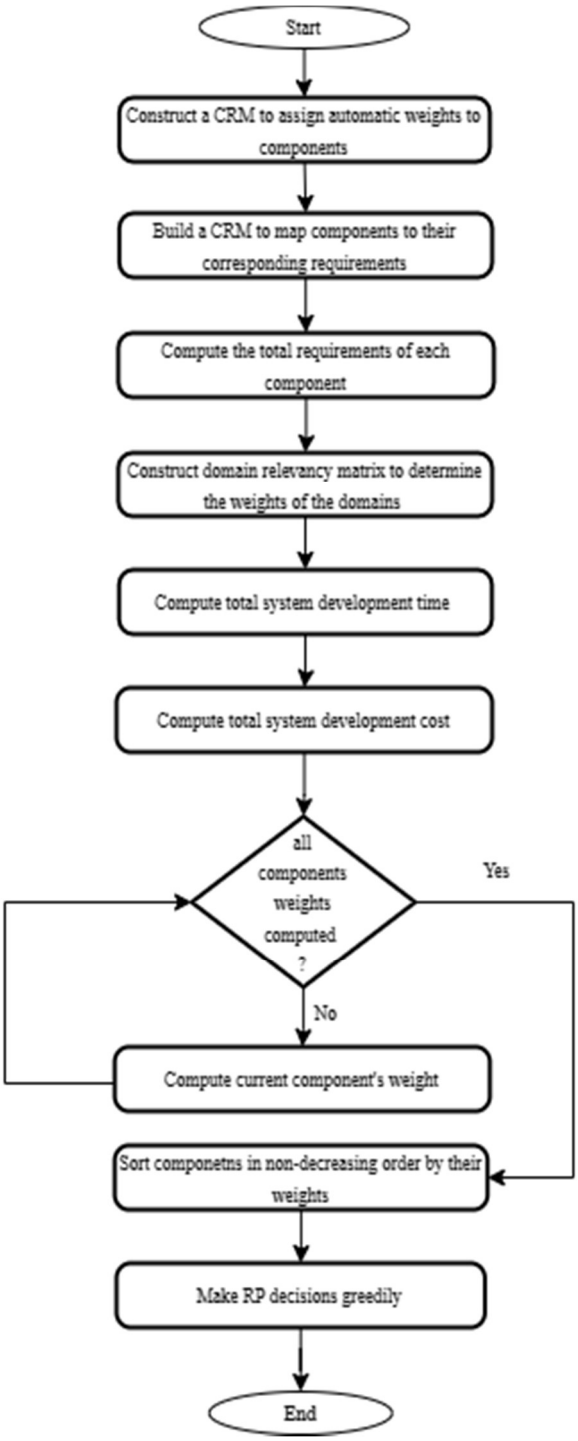

Figure 1. The HA Algorithm Depicted As A Flowchart

4. EXPERIMENTAL RESULTS

A library information system is considered in this context to prove the correctness of the approach presented earlier. This information system is presented by Al-Shaikh et al. [17].

The system is composed of the following components: Book, Borrower, and Borrowing, as shown in Figure 2. The functional requirements of the system are [18]:

R1. Issue a book to a borrower.
R2. Receive a book returned by a borrower.
R3. Create information about a newly acquired book.
R4. Display a list of the books on loan to a particular borrower.

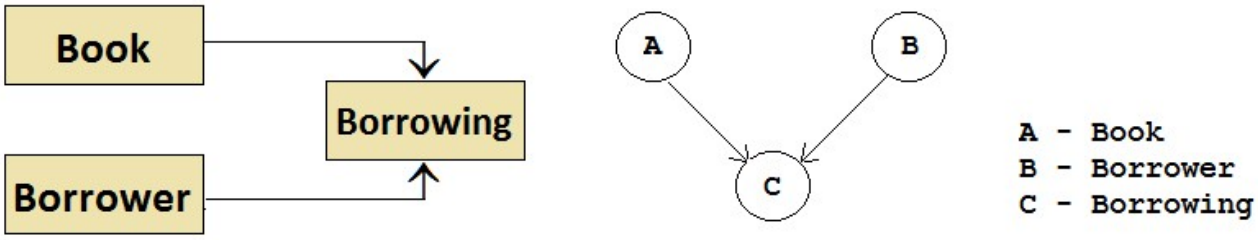

Figure 2. System Decomposition And Representation Using The Module Dependency Graph

In Figure 2, all component weights are initially set to -1. After applying Algorithm 4, the weights of the components will be as follows: Book: 1, Borrower: 1, and Borrowing: 0.

Then, CRM is CRM by Algorithm 4, as shown in Table 1.

Table 1 CRM

	Book	Borrower	Borrowing
R1	1	1	1
R2	1	1	1
R3	1	0	0
R4	1	1	1
Total	**4**	**3**	**3**

The domain of application of this system is the library, and it is assumed that there are no further domains of applications to this system. This implies that all values of domain of application would be set to 1. Thus, the value of domain of application will be mitigated and will not have any effect on any component.

Let the users of the system are as listed in Table 2, assume users of the system as follows:

Table 2 User Weights

User	Domain		Position		p
	Name	d	Title	value	
User 1	Library	1	Director	4	4
User 2			Supervisor	3	3
User 3			Data Entry	1	1
User 4			Data Entry	1	1
User 5			Borrower	1	1
User 6			Borrower	1	1

Afterwards, the user-value matrix is constructed as follows:

Table 3 User-Value Matrix

	Book	Borrower	Borrowing
User 1 (4)	1	1	1
User 2 (3)	1	1	1
User 3 (1)	1	1	0
User 4 (1)	1	1	0
User 5 (1)	0	1	1
User 6 (1)	0	1	1
Total	**9**	**11**	**9**

Assuming that the development time and development cost for each component is given as follows:

Table 4 Components Development Times And Costs

Component	DT	dt_i/DT	DC	dc_i/DC
Book	2	0.25	250	0.16
Borrower	2	0.25	350	0.22
Borrowing	4	0.5	1000	0.62
Total	**8**		**1600**	

Finally, we group them together as shown in Table 5.

Table 5 Final Calculations

Component	W	T	U	DT	DC	C
Book	1	4	9	0.25	0.16	**14.41**
Borrower	1	3	11	0.25	0.22	**15.47**
Borrowing	0	3	9	0.5	0.62	**13.12**

The final step of Algorithm 4 is to sort the modules in a non-decreasing order to obtain the result: (1) Borrower, (2) Book, and (3) Borrowing. Using the greedy approach of HA, the first component to start with is Borrower, then Book, and at the end we finish up with the Borrowing component.

Intuitively, the borrowing component cannot be the first component to start with, from a logical explanation of the function of the system. Also, not matter which component to start with, i.e., Book or Borrower, as shown in Figure 2. However, we cannot start with the Borrowing module until we

finish the two other modules, and this was reflected by the devised HA model.

Experimentally, the HA model was tested against the AHP method. The tests are set to run on an Intel Core(TM) i5-3230M CPU with 2.60 GHz and 3 MB cache with 4 cores. Memory size of the computer in use is 4 GB of RAM (2.87GB is only usable). The PC runs windows 7 Enterprise edition 32-bit.

The datasets with different sizes between 100 and 10000 requirements were generated randomly. For each algorithm, the algorithm was set to run 30 times on each dataset. The factors that are related with each component and requirement were set randomly by the program, too.

Table 6 shows a comparison between the runtimes of the AHP method and HA model. The results show that the HA model outperforms the AHP in terms of runtime for most of dataset sizes as well as in the average runtime of the algorithm.

The comparison results are depicted in Figure 3. It is obvious from Figure 3 that HA outperforms AHP in terms of runtime. There might be some cases in which runtime values for AHP are better than those of HA. However, this behavior has no effect on the average runtime. Also, the reason of this behavior is the random nature of the datasets because the datasets are generated randomly.

Table 6 Comparison Between AHP And HA In Terms Of Runtime In Seconds.

Dataset size	AHP	HA
100	0.206165	0.045097
200	0.237553	0.047081
300	0.26894	0.139259
400	0.300328	0.231437
500	0.331716	0.323615
600	0.363104	0.415793
700	0.394491	0.507971
800	0.425879	0.600149
900	0.457267	0.692327
1000	2.160758	0.784505
2000	4.146739	1.706284
3000	6.132719	2.628064
4000	8.1187	3.549843
5000	10.10468	4.471622
6000	12.09066	5.393402
7000	14.07664	6.315181
8000	16.06262	7.23696
9000	18.0486	8.15874
10000	20.03458	9.080519
Average	**5.998008**	**2.754097**

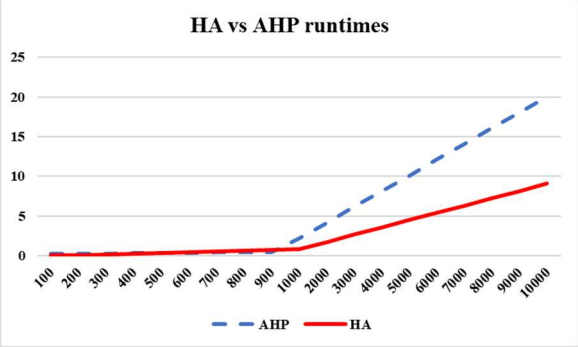

Figure 3. Comparison between HA and AHP runtimes

Using runtime analysis and comparison between the proposed HA and the AHP is of a great importance. In fact, agile methodologies, such as DevOps, requires swift methods in dealing with increments. This implies that time is a major factor in determining appropriateness of the proposed model with agile and DevOps. A slow RP process will delay will slow down the CD of software components, or increments. Consequently, the appropriateness of the proposed model to fit in DevOps is proved.

In a nutshell, having HA to outperform AHP in terms of runtime indicates that the proposed approach is a fast approach that can be used with agile development methodologies and in DevOps. It also shows preference on the methods or approaches that are found in the literature, in the sense that this approach uses multiple techniques to prioritize the components as well as it is a fast approach that fits in modern software development approaches.

5. CONCLUSION AND FUTURE WORK

This work presented a model of evaluating increments weights based on some factors, like the dependency between components, requirements that each component aggregate, users of each component and their values, in addition to the relevancy between different domains of application and the overall software system, plus some actuators which are the development time and the development cost of each

component. The model computes the values of the factors and then applies a greedy algorithm to prioritize the components, such that the higher weight is selected first, followed by the one with less weight and thus until reaching the last component (or increment) to deliver, and a result the whole system in total is delivered. Most of the analysis done in this context was conducted using value-oriented prioritization (VOP) matrices.

The resulted HA algorithm was compared to the AHP algorithm in terms of runtime. results show that the HA algorithm outperformed AHP in terms of runtime.

As a future work, some NFRs can be considered and added to this model, such as availability of resources, staffing, team knowledge, experience, and fields of interest, and working hours, to name a few. Also, another metaheuristic algorithms can be used instead of the greedy algorithm, such as the most-valuable player (MVP), duelist algorithm (DA), or others.

REFERENCES:

[1] F. Hujainah, R. B. A. Bakar, A. B. Nasser, B. Al-haimi and K. Z. Zamli, "SRPTackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects," Information and Software Technology, vol. 131, 2021. https://doi.org/10.1016/j.infsof.2020.106501

[2] D. Ameller, C. Ayala, J. Cabot and X. Franch, "How do software architects consider non-functional requirements: An exploratory study," in 2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, 2012.

[3] D. Firesmith, "Prioritizing Requirements," Journal of Object Technology, vol. 3, no. 8, pp. 35-47, 2004.

[4] M. Khari and N. Kumar, "COMPARISON OF SIX PRIORITIZATION TECHNIQUES FOR SOFTWARE REQUIREMENTS," Journal of Global Research in Computer Science, vol. 4, no. 1, pp. 38-43, 2013.

[5] J. Azar, R. K. Smith and D. Cordes, "Value-Oriented Requirements Prioritization in a Small Development Organization," IEEE Software, vol. 24, no. 1, 2007. https://doi.org/10.1109/MS.2007.30

[6] M. Sadiq and V. S. Devi, "A rough-set based approach for the prioritization of software requirements," International Journal of Information Technology, vol. 447–457, p. 447–457, 2022. https://doi.org/10.1007/s41870-021-00749-0

[7] H. Ahuja, Sujata and U. Batra, "Performance Enhancement in Requirement Prioritization by Using Least-Squares-Based Random Genetic Algorithm," in Innovations in Computational Intelligence, Singapore, Springer, 2018, p. 251–263.

[8] A. Al-Shaikh, B. A. Mahafzah and M. Alshraideh, "Hybrid harmony search algorithm for social network contact tracing of COVID-19," Soft Computing, 2021. https://doi.org/10.1007/s00500-021-05948-2

[9] J. A. Khan, I. U. Rehman, Y. H. Khan, I. J. Khan and S. Rashid, "mparison of Requirement Prioritization Techniques to Find Best Prioritization Technique," International Journal of Modern Education and Computer Science, vol. 7, no. 11, pp. 53-59, 2015. https://doi.org/10.5815/ijmecs.2015.11.06

[10] S. Shafiq, A. Mashkoor, C. Mayr-Dorn and A. Egyed, "NLP4IP: Natural Language Processing-based Recommendation Approach for Issues Prioritization," in 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Palermo, Italy, 2021.

[11] S. Aly, J. Tyrychtr, R. Kvasnicka and I. Vrana, "Novel methodology for developing a safety standard based on clustering of experts' assessments of safety requirements," Safety Science, vol. 140, 2021. https://doi.org/10.1016/j.ssci.2021.105292

[12] M. Roy, N. Deb, A. Cortesi, R. Chaki and N. Chaki, "NFR-aware prioritization of software requirements," Systems Engineering, vol. 24, pp. 158-176, 2021. https://doi.org/10.1002/sys.21572

[13] K. Pal and B. Karakostas, "Software Testing Under Agile, Scrum, and DevOps," in Agile Scrum Implementation and Its Long-Term Impact on Organizations, 2021.

[14] C. Ebert, G. Gallardo, J. Hernantes and N. Serrano, "DevOps," IEEE Software, vol. 33, no. 3, pp. 94-100, 2016. https://doi.org/10.1109/MS.2016.68

[15] A. Al-Shaikh, B. A. Mahafzah and M. A. Alshraideh, "Metaheuristic approach using grey wolf optimizer for finding strongly connected components in digraphs," Journal of Theoretical and Applied Information Technology, vol. 97, no. 16, pp. 4439-4452, 2019.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

http://www.jatit.org/volumes/Vol97No16/17Vol97No16.pdf

[16] A. Al-Shaikh, H. Khattab, A. Sharieh and A. Sleit, "Resource Utilization in Cloud Computing as an Optimization Problem," International Journal of Advanced Computer Science and Applications, vol. 7, no. 6, pp. 336-342, 2016. http://doi.org/10.14569/IJACSA.2016.070643

[17] A. Al-Shaikh, H. Khattab, A. Moubaiddin and N. Obeid, "A defeasible description logic for representing bibliographic data," in Social Media Shaping e-Publishing and Academia, Springer, Cham, 2017, pp. 90-105.

[18] D. Bell, Software Engineering for Students – A Programming Approach, vol. 4th, Addison-Wesley, 2005.