E-AHP: An Enhanced Analytical Hierarchy Process Algorithm for Priotrizing Large Software Requirements Numbers

Nahla Mohamed[1], Sherif Mazen[2], Waleed Helmy[3]
Department of Information Systems, Faculty of Computers and Artificial Intelligence, Cairo University, Giza, Egypt

***Abstract*—One of the main activities of software requirements analysis is requirements prioritization. The wrong requirements prioritization is risky as it leads to many software failures. The current requirements prioritization techniques can't deal with large requirement numbers efficiently, which is considered one of their main issues. Many researchers have agreed that the analytical hierarchy process (AHP) is one of the best prioritization techniques as it produces highly accurate results. AHP has two main problems: scalability and inconsistency. These problems have motivated us to propose an improved version of AHP for software requirements prioritization, namely Enhanced AHP (E-AHP). A performance evaluation has been done for the conventional AHP, E-AHP, and one of the recent algorithms that also try to solve the AHP scalability problems, namely removing eigenvalues and introducing the dynamic consistency checking algorithm into AHP (ReDCCahp) algorithms The evaluation shows which algorithm takes the least time, uses the least memory, produces the most consistent and accurate results, and has the highest scalability. The three algorithms have been evaluated by running their codes using different numbers of requirements ranging from 10 to 500. The results show that E-AHP is more scalable, takes the least time, uses the least memory, and produces the most consistent and accurate results compared to the other two algorithms. That becomes remarkable when the number of requirements increases. Therefore, E-AHP is suitable to be applied in large software projects, as it can deal efficiently with the large software requirements numbers.**

Keywords*—Requirements engineering; analytical hierarchy process; software engineering; requirements prioritization techniques

I. INTRODUCTION

Requirements engineering is a critical part of software engineering [1]. It is the process of gathering the requirements and understanding them deeply to ensure that they are correct, complete, and consistent [2]. If the requirements engineering process has not taken enough time, it will affect the overall project [3], [4]. The requirements engineering process consists of five activities: requirements elicitation, requirements analysis, requirements specification, requirements validation, and requirements management. The prioritization of requirements is one of the critical activities in the requirements analysis process [5], [6]. When the requirements number increases, analysts must organize them to implement the most important ones in the early stages to avoid the high cost of system transformation and rework and achieve user satisfaction according to a pre-specified budget, time, and resources [7].

There are three requirements prioritization technique types [8], [9]: nominal scale techniques, ordinal scale techniques and ratio scale techniques. In the Nominal scale prioritization techniques [7], [10], the users assign each requirement to a priority group, and all requirements in the same group have the same priority [8]. One of the well-known techniques is the Numerical Assignment technique, which categorizes the requirements by distributing them into groups [11]; each group has a number that describes its rank or order among all groups. And the number of groups equals the scale range [9], [12]. Top Ten Requirements is another well-known nominal scale technique. It has only one group that contains the most ten critical requirements [9]. Another technique is MoSCow, which distributes the requirements into four main groups [8]: Must-Have, Should-Have, Could-Have, and Will-Not-Have [5], [12]. These techniques are simple, easy, and fast [10]. But their results are not accurate in most cases as they don't give specific priority value to each requirement [13] and cannot deal efficiently with large requirements numbers [11].

The Ordinal scale techniques produce an ordered requirements list [10], [12], and each requirement has a specific priority [8]. They are more accurate than nominal scale techniques [9]. One of the well-known ordinal scale techniques is the Priority group [11]. It is like the Nominal scale techniques but has only three groups: High, Medium, and Low. The users prioritize and classify the requirements within the same group into another sub-group; users repeat that looping until each group has only one. Bubble sort is another well-known ordinal scale technique [11]. In this technique, the user should list the requirements and then compare every adjacent two [9]. If the second one has less priority than the first, the user swaps the order of these two requirements. The user should repeat this process for each element in the requirements list until it becomes sorted in ascending order.

Binary Search Tree (BST) is another well-known ordinal scale technique. It depends on node structure. In BST, each node represents a requirement [14] .The root node is in the first level. The last level is the ordered requirements list. BST works as follows: the user first selects one requirement to represent the top node (the root node) [8]. After that, the user iterates on the requirements list; if the requirement in the root node is more important than the requirement in the current node, the user should search in the left sub-tree to place it. Otherwise, the

user searches in the right sub-tree. The user repeats this process until putting each requirement in the right place in the tree based on its priority. The ordinal scale techniques have medium scalability, consume more time, and are less easy to use than nominal scale techniques [9].

The Ratio scale techniques are similar to the Ordinal scale prioritization techniques [6], [10]. In addition, they show relative importance among all the requirements, which means they give the requirements priority values [8]. In these techniques, the users know to what extent each requirement is more important than the others [9]. Cumulative Voting (CV) is a well-known ratio scale technique that depends on the users' voting; each user has 100 points [9] and distributes the points among the requirements based on their priority [15]. Hierarchical Cumulative Voting (HCV) is a new modification of the CV technique [2]. The main difference is that HCV also prioritizes the detailed requirements (prioritizes requirements and their sub-requirements hierarchically).

Analytical Hierarchical Process (AHP) is multi-criterion decision-making and mathematical method used in many fields, including requirements prioritization [6], [9]. It selects the best decision based on pairwise comparisons among all decisions concerning many criteria [8], [13] (note that AHP will be explained in detail in section three; because the proposed algorithm is based mainly on it). It is good to use one of these techniques when the project is critical, and it is necessary to know the exact difference of importance among all the requirements [11].

Most of the prioritization techniques can't deal with large requirements numbers and produce accurate results at the same time [2], [5], [9], [11], [13], [14], [15]. Researchers [2], [7], [9], [11], [12], [14], [15] agreed that AHP is the most accurate prioritization technique, as it is a mathematical-based method and produces highly accurate results. But they found that AHP is suitable to be used only if the requirements number is small; otherwise, it is not good as it is not scalable and sometimes suffers from inconsistency problems. Scalability means the ability of a technique to deal with a large number of requirements efficiently, and inconsistency means it sometimes produces elements that are semantically conflicting and not compatible with each other. The limitations of AHP can be summarized as follows:

- Sometimes, the results of AHP may be inconsistent because of the high human involvement [13], especially with large requirements numbers [5].
- AHP is not fast; it takes much time to work [8], [13].
- It is not easy for users to use as it needs an excellent mathematical base. It also needs time to understand how it works [13], [4].
- AHP performs n*(n-1)/2 comparisons [4], [9]where n is the number of requirements ; that means when the requirements numbers increases, the pairwise comparisons number will increase exponentially [11], which indicates it does not work well with a large number of requirements and is not scalable [3], [13].

The previous AHP limitations have motivated us to search for new ways to enhance it to deal efficiently with large requirements numbers.

The main contributions of this paper can be summarized as follows:

- Proposing a new algorithm that tries to solve the scalability problem that faces the conventional AHP and minimizes inconsistent and inaccurate results.
- An experiment that compares the proposed algorithm against the AHP and one of the best recent algorithms introduced to solve the scalability problem of AHP, namely removing eigenvalues and introducing the dynamic consistency checking algorithm into AHP (ReDCCahp), is conducted. The experiment aims to test the three algorithms' scalability, complexity, results' accuracy, and consistency.

The rest of the paper is structured as follows: Section II presents the related works on the recent techniques introduced to solve the scalability and inconsistency problems of the conventional AHP and other prioritization techniques. Section III is the research background; it explains the conventional AHP (as the proposed algorithm is a modification of AHP). Section IV presents the proposed requirements prioritization algorithm. Section V presents an experiment that compares the proposed algorithm against the AHP and ReDCCahp algorithms. Section VI presents the experimental results and discussion. Section VII is the conclusion of the paper. Section VIII is the limitations and future works.

II. Related Work

Many researchers introduced several approaches and techniques to prioritize a large number of requirements efficiently. This section will briefly explain most of them. Market-Driven Requirement Prioritization Model (MDRM) [16] is an AHP modification model introduced to deal with large requirements numbers by reducing the number of pairwise comparisons. The main idea of MDRM is to divide all the requirements into bins and prioritize all these bins by AHP. The main limitations of this technique are that it can't consider the dependencies and conflicts among the requirements [3] and cannot deal with large requirements number efficiently [13].

NAcAHP is another technique introduced to prioritize a large number of requirements [17]; it combines the AHP technique with the Numerical Assignment technique to reduce the time that results from the pairwise comparisons [12], [11]. There are three main priority groups: Optional, Standard, and Critical. AHP works only on ones in the Critical group. One of the main limitations of this technique is that it works well only if at least 80 % of all requirements are critical (because users can't know their priorities until completing the prioritization process) [18], [19]. Other limitations [2], [3] are that it does not do consistency checking for the results, and it has not been evaluated on large data sets [17].

Fuzzy AHP [20] is another approach introduced to solve the scalability issue that faces AHP [11]. The main idea of Fuzzy AHP is to use fuzzy scales [21], and the pairwise comparison matrix consists of fuzzy triangle numbers. It

provides flexibility and efficiency to get benefits from the decision-maker's preferences. This approach also addresses the uncertainty in human judgment that AHP cannot address [3], [18]. Fuzzy AHP has many limitations. One of them is that it is not reliable [8], couldn't solve the scalability problem as it can't deal with large requirements numbers and takes much time to work [13]. Another limitation [22], [23] is that it doesn't consider requirements dependencies. And also, fuzzy systems are highly dependent on human expertise, have no systematic problem-solving approach, and need a lot of validation and testing. Researchers [22] proposed a goal-based requirements prioritization technique. It depends on giving weights to the requirements based on the different project's goals [3]. But this technique is not scalable [13], suffers from the data vagueness and uncertainty problems as it heavily relies on user involvement, and does not consider the dependency relationships among the requirements [22].

The Interactive Genetic Algorithm-based (IGA) technique [24] was introduced to solve the scalability problem by combining pairwise comparisons IGA [18]. This technique uses the IGA to reduce the pairwise comparisons number [3], [14], it's working on extracting from the user the relevant knowledge, and each user provides his preference values. IGA algorithms don't require much information about the problem. But they have many limitations [2], [5]. Un-Scalability is a major one, as the search space increases exponentially when the number of problem elements increases [25]. Another is that the experts choose the best solution only after comparing it to the others, has no stopping criteria, and designing the objective function and getting the correct operators and representation needs effort [9].

Researchers [9] introduced an expert system, namely the Priority Handler (PHandler), to solve the scalability issue. It combines three approaches, Value-based Intelligent Requirement Prioritization (VIRP), the Back Propagation Neural Network (BPNN), and AHP. PHandler predicts the values of the requirement priorities by applying the BPNN, and then AHP. It can deal with large requirements numbers [13]. The main challenge of this system is choosing professional business analysts because a strong analyst's knowledge is necessary to estimate accurate values of requirements classification factors. One of the main limitations [13] of this system is that it neglects the dependency relationship among requirements. And the expert systems do not explain the logic behind taking a decision, cannot easily automate complex processes, and have no common sense when making a decision.

Fuzzy AHP ANN [21] is an artificial intelligence decision support system proposed to deal with large requirements numbers [18]. It integrates the Artificial intelligence Neural Network (ANN) with AHP to select the best alternatives. It determines the priority weights for the requirements using a program, namely PECAR. After that, a supervised ANN is trained (by applying a feed-forward back-propagation algorithm) using results from the PECAR program. And the decision-makers can apply different scenarios using the PECAR program by entering several input parameters into it and then observing the difference among the results. The main challenge of this system is that it needs high experts' involvement in the prioritization process. One of the main limitations of this system [13] is that it does not produce consistent results. Another limitation is that large neural networks consume a high processing time, need a lot of data to work, cannot specify a single solution for the problem, and not scalable [2].

Researchers [26] introduced a graph-based approach to prioritize a large number of requirements. It represents the requirements as a directed graph; each node represents a requirement and can be a pre-request for or dependent on other nodes. The dependency relationships among nodes are represented as directed arrows. After that, all spanning trees are generated from the graph. In the end, requirements priorities values will be calculated based on the number of requirements dependent on them (the dependent requirements will have lower priority than the pre-requisite requirements). The main limitation of this approach [27] is the large memory consumption. It is also hard to be implemented by users; its representation is not structured, and has no specific spanning concept [26]. Researchers [4] introduced an iteration model for implementing large numbers of requirements. The main idea is to implement the requirements in phases and not all at one time. It uses the graph-based approach. One of the main limitations of this model is it does not implement all the requirements as it implements the critical ones only [4]. Another limitation is that system architecture issues will appear as all the requirements have not been gathered together, it needs more resources, and it doesn't consider the dependency among the requirements.

Researchers [19] introduced another technique based on AHP, namely, ReDCCahp. The main idea of ReDCCahp is to put every pair of adjunct requirements from the requirements list in one group and make the pairwise comparison among these groups to reduce the number of pairwise comparisons and matrix size. It is fast, simple, and does not need a strong background in math, data science, or data structure; to understand it. So it is easy to use and understand by users and more effective than AHP when dealing with a medium number of requirements. But this technique is not highly accurate as it randomly groups the adjacent requirements in the list, which means it does not have a specific requirements grouping method, which is considered its main limitation [19]. It also can deal with only small and medium requirements numbers [13] and doesn't consider the requirements dependencies. Because ReDCCahp is one of the easiest and best new requirements prioritization techniques, the proposed algorithm tries to solve its limitation besides AHP limitations by introducing an efficient method for requirements grouping. A comparison has been made among the proposed algorithm, the conventional AHP, and the ReDCCahp algorithms.

III. Background: The Conventional AHP

This section explains the conventional AHP; because the proposed algorithm is based on it. AHP is a multi-criterion decision-making method developed by Saaty (1980) [28], [29] to solve social science domain problems. It had been used in many other fields, one of which is requirements engineering. AHP is a mathematical technique based mainly on pairwise comparisons; it selects the best decision based on comparisons

among all decisions concerning many criteria. It is one of the efficient and best techniques for dealing with complex decisions. AHP consists of two phases: 1. Construct the reciprocal matrix and get the priority vector (PV). 2. Checking the consistency of the results.

In the first phase, an n × n reciprocal matrix is constructed (where n is the number of requirements) from the pairwise comparisons among the requirements by letting the user choose a value from a specified scale in AHP (each value in the scale refers to a specific importance degree) between each pair of requirements as follows: the user will put 1 in the cell(i, j) and cell(j, i) in the matrix when the requirements i and j have the same importance, and if i has more priority than j, the user should put the value he chooses from the scale in cell (i, j), and put the reciprocal of this value in cell (j, i) and vice versa. Table I shows the pairwise comparison scale in AHP.

TABLE I. PAIRWISE COMPARISON SCALE IN AHP

Intensity of importance	Description	Reciprocal value
1	Equal importance	1
2	Equal to moderate difference in importance	1/2
3	Moderate difference in importance	1/3
4	Moderate to strong difference in importance	1/4
5	Strong difference in importance	1/5
6	Strong to very strong difference in importance	1/6
7	Very strong difference in importance	1/7
8	Very strong to extremely difference in importance	1/8
9	Extreme difference in importance	1/9

Then, each element in the matrix should be divided by the sum of its columns to get a normalized matrix. After that, the user should sum elements of each row in the matrix to get the eigenvector. The last step is normalizing the eigenvector by dividing each cell by the requirements number. The normalized eigenvector is the PV, which describes the relative weights among the requirements, and the summation of all values in PV should equal one. After finishing these steps, user goes to the second phase, the Consistency checking. It means that if there are three requirements: R1, R2, and R3. R1 is more important than R2, and R2 is more important than R3, then R1 should be more important than R3. That check is called the Transitivity check. So the user should ensure that all the elements in the matrix are transitive (the matrix is a correct reciprocal matrix). The percentage of inconsistent results is high as the matrix produced by AHP is made by humans.

Professor Saaty defined a measure for consistency checking called Consistency Index (CI), which is calculated using the formula (λmax-n) / (n-1), where λmax means the maximum eigenvalue of the matrix [30], and n is the number of pairwise comparisons. To ensure that the matrix is consistent, CI should equal zero (λmax should equal n), which means there is no deviation or difference between the excepted reciprocal matrix and the resulting one. But in real-life ideal cases rarely happen, so; how much inconsistency is acceptable? Professor Saaty put a specified percentage; if the error didn't exceed it, then the matrix is consistent (there is a minimum acceptable ratio for the inconsistency).

Saaty defined this ratio as Consistency Ratio (CR), which is a guide to checking whether the matrix is consistent or not. If CR is more than 10%, then the matrix is inconsistent, and users should repeat the process from the beginning, but if the CR value is equal to or less than 10 %, then the matrix is consistent. CR value is the value of CI divided by Random Index (RI), where RI is the average CI value of several comparison matrices sizes.

IV. E-AHP: THE PROPOSED ALGORITHM

Ma [31] has found that the user's effort in the prioritization process should be reduced to solve the AHP's scalability problem. And that can happen by reducing the time pairwise comparisons take. This section introduces an AHP-based algorithm namely, Enhanced Analytical Hierarchal Process (E-AHP). Which increases the scalability of AHP by reducing the time AHP takes to construct a reciprocal matrix; its main idea is to group similar requirements using a specific method. It also decreases the inconsistent results by giving scores to the requirements groups. E-AHP consists of five main steps that will be explained in the following subsections.

A. Gathering the Requirements

First, the analysts should gather all the functional and non-functional requirements, ensure they are consistent, discover any dependencies among them [2] and make sure they are clear and specific.

B. Scoring and Sorting the Requirements

In this step, each user assigns a score to each requirement, and the score scale will equal the total requirements number. More than one requirement can have the same score if they have the same priority to the user. After finishing the scoring process, the algorithm sorts the requirements in descending order by their scores. For example, if there are three requirements: R1, R2, and R3, in this case, the score scale will be from 1 to 3. If the user assigns scores 1 to R1, 3 to R2, and 2 to R3, then the algorithm will sort them in descending order, and the sorted requirements list will be 1. R2, 2. R3, 3. R1.

To sort the requirements' scores list, a hybrid algorithm of insertion sort algorithm [19] and merge sort algorithm [32] is applied. The main idea of the hybrid algorithm is to divide the list of requirements scores into chunks. The chunk is an ascending or descending sorted sub-list that has the following patterns: $a_i > a_{i+1} > \ldots > a_n$ or $a_i < a_{i+1} < \ldots < a_n$ where a is the score of the requirement in position i, and n is requirements number. For example, if there is a list {1, 5, 6, 4, 3, 2} then the first chuck will be {1, 5, 6} (ascending order), and the second chunk will be {4, 3, 2} (descending order). After that, the algorithm reverses the first chunks to be sorted in descending order. A minimum size for each chunk is defined. For example, if the list is {2, 3, 1, 4, 5, 6} and the minimum pre-specified chunk size is 3, then the first chunk should be {2, 3, 1} not {2, 3}, although the element 1 breaks the chunk's pattern. After that, the algorithm does an insertion sort in descending order to this chunk to be {3, 2, and 1}. That algorithm fastens the

sorting process (the complexity of the best case is O (n)) as it benefits from the already existing sorted sub-lists. In the end, the merge sort is applied to all these sub-lists to make a final sorted list.

C. Grouping the Similar Requirements

After sorting the requirements list, the algorithm will group all requirements that have the same score, or the difference among their scores is <= MaxDRS. Where the MaxDRS variable is the Maximum Difference of Requirements Scores in the same group; and it is pre-specified by the user. The user also specifies the value of MaxNR, which is the Maximum Number of Requirements in one group. For example, if there are five requirements: R1, R2, R3, R4, and R5. R1 and R2 have a score of 1, R3 has a score of 2, R4 has a score of 5, R5 has a score of 6, and MaxDRS = 1. Then the algorithm will put R1, R2, and R3 in one group and R4; and R5 in another group. And if MaxNR = 5 and seven of them have a score <= the specified MaxDRS, in this case, only five of them will be in the same group, and the rest two will be in another new group. To assign a score for one group, the algorithm calculates the average score of all its requirements. For example, if a group has three requirements: R1, R2, and R3, and the score of R1, R2 is 1, and the score of R3 is 2, then the score of their group will be (1+1+2)/3 =1.3.

The different values of MaxNR and MaxDRS influence the results, because there are negative relationships between the values of MaxNR, MaxDRS, and time, and between them and accuracy. If users aim to decrease the time taken, the MaxNR and MaxDRS values should be increased, which reduces the accuracy and vice versa. The best choice is to choose the values of MaxNR and MaxDRS based on the total number of requirements in the project. If the requirements number is large, it's better to choose large values for them and vice versa.

D. Constructing the Reciprocal Matrix

In these steps, E-AHP constructs the reciprocal matrix like AHP, but in E-AHP, the rows and columns of the matrix are the requirements groups, and the matrix's elements are the difference between the scores of the requirements groups. For example, if the score of G1 is 4; and the score of G2 is 2, then the element in the intersection of row G1, column G2 will be 2. And will be -2 between row G2 and column G1. After that, E-AHP normalizes the elements in the matrix and does the same mathematical calculations in AHP to get the PV. The flow chart in Fig. 1 and Algorithm I explain the steps to get the PV in E-AHP.

Algorithm I. Construct PC matrix and get the PV

```
Input: N - number of requirements
Input: RN - the requirements names list
Input: RS - the requirements scores list
Input: MaxDRS - maximum difference of requirements scores in one group
Input: MaxNR - maximum numbers of requirements in one group
Output: List of prioritized requirements
Initialize: R        // list of requirements objects
Initialize: G        // One group of requirements
Initialize: GL       // One group length
Initialize: AG       // List of all groups
Initialize: NG       // Number of all groups
Initialize: Sum      // Counter
Initialize: M        // The Reciprocal matrix
Initialize: CS       // List of column sum
Begin
for i in N do
    requirement ← new Requirement()
    requirement.name ←RN[i]
    requirement.score ← RS[i]
    requirement.ingroup ← false
    R.append(requirement)
end for
R ← sort(R by names, reverse=true)
for i in R.length do
    if R[i].ingroup == true then
        skip to next iteration
    G.reqList.append(R[i])
    R[i].ingroup ← true // R[i] is assigned to a group
    for j in R.length do
        if R[j].ingroup == true
        or (R[j].score - R[i].score) > MaxDRS
        or G.length > MaxNR then
            skip to next iteration
        else
            G.reqList.append(R[i])
            R[i].ingroup ← true
        end if
        G.reqList ← []
        G.avScore ← 0.0
    end for
    AG.append(G)
end for
NG ← AG.length
for i in NG do
    if AG[i].length == 1 then
        AG[i].avScore←AG[i][0].score
    else
        GL ← AG[i].length
        for j in GL do
            AG[i].avScore ←AG[i].avScore +
            AG[i].reqList[j].score
        end for
        AG[i].avScore←AG[i].avScore / GL
    end if
end for
for i in NG do
    for j in NG do
        if AG[i].avScore == AG[j].avScore then
            M[i][j] ← 1
        else if AG[i].avScore > AG[j].avScore then
            M[i][j]←AG[i].avScore - AG[j].avScore
        else
            M[i][j] ←1 / (AG[i].avScore -
            AG[j].avScore)
        end if
    end for
end for
Sum ←0
for j in NG do
    for i in NG do
        Sum ← Sum + M[i][j]
    end for
```

```
        CS[j] ←Sum
        Sum ← 0
    end for
    for j in NG do
        for i in NG do
            M[i][j] ← M[i][j] / CS[j]
        end for
        Sum ←0
    end for
    for i in NG do
        PV[i] ←PV[i] / NG
    end for
    for i in NG do
        for j in AG[i]
            print(AG[i].reqList[j].name "has relative weight
            : " + PV[i])
        end for
    end for
    End
```

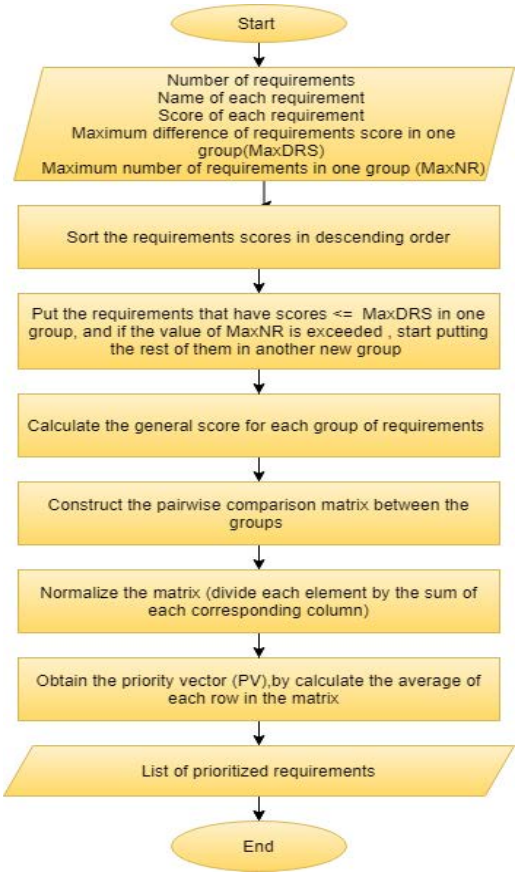

Fig. 1. A Flowchart shows Steps to Get the PV in the E-AHP Algorithm.

E. Consistency Check of the Results

In this step, a new consistency check algorithm is applied to ensure that the results are consistent. This step has three inputs: X, Y, and Z. X is a list of the most critical requirements to the user, and Y is the number of first groups from the resulting PV that will be checked. And Z is the minimum acceptable number of requirements in the list X that must be in the Y groups. The user fills the list (X), and to prove consistency, the algorithm checks that at least (Z) of them are in the (Y) groups from the resulting PV; otherwise, the result is inconsistent. And users should repeat the prioritization process from the beginning and re-evaluate their preference judgments. For example, if the size of the list X =5, Y=6 and Z=3, which means the user will choose the most critical five requirements for him, and then to prove the consistency of the results, the algorithm must find at least three of these requirements in the first sex groups from the resulting PV. These variables influence the results as the more the values of X, Z and the less the value of Y, the more accurate the results will be.

The flow chart in Fig. 2 and Algorithm II explain the consistency check step in E-AHP.

Algorithm II. Consistency checking of the results

```
Input: X - list of most important requirements
Input: Y - number of requirements for check
Input: Z – minimum accepted number of found requirements
Input: PV - priority vector result from phase 1
Output: print whether the results are consistent or not
Initialize: XL // length of most important requirements list
Initialize: nXF // number of important requirements found
Begin
nXF ← 0
for i in XL do
    for j in Y do
        if X[i] == PV[j] then
            nXF ← nXF + 1
        end if
    end for
end for
if nXF >= Z
    print ( "Results are consistent")
else print ("Results are not consistent")
end if
End
```

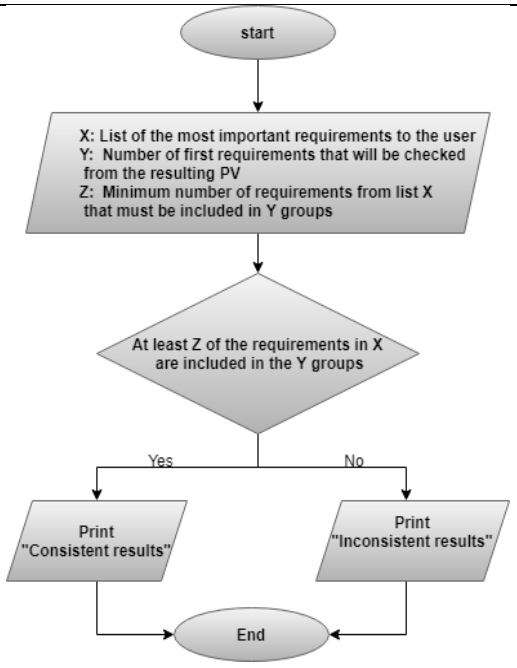

Fig. 2. A Flowchart shows the Consistency Check Step in the E-AHP Algorithm.

V. An Experiment Comparing the Proposed Algorithm (E-AHP) Against AHP and ReDCCahp Algorithms

This section will explain the experiment that compares the E-AHP against the AHP and ReDCCahp. It will mention the experiment objectives, variables, and setup.

A. Experimental Objective

This experiment aims to validate that E-AHP (the proposed algorithm) is better than AHP and ReDCCahp; by proving that it solves the scalability, inconsistency, and accuracy problems (as AHP suffers from scalability and inconsistency problems, and ReDCCahp suffers from accuracy problems). The experiment has compared the three algorithms (AHP, ReDCCahp, and E-AHP) against each other to test which one can deal with large numbers of requirements efficiently.

The research aims to answer the following questions:

- Which algorithm among the three algorithms takes the least time?
- Which algorithm among the three algorithms uses the least memory?
- Which algorithm among the three algorithms produces the most consistent results (produces results with minimum CR value)?
- Which algorithm among the three algorithms is the most scalable?
- Which algorithm among the three algorithms produces the most accurate results?

B. Experimental Variables

The experiment has three independent variables: AHP, ReDCCahp, and E-AHP, and three dependent variables: time, memory, and CR value. A brief definition for these dependent variables is given below.

- Time: representing the time each algorithm takes to prioritize the requirements (completion time of each algorithm), and it is measured in minutes.
- Memory: representing the memory needed for each algorithm to prioritize the requirements, and it is measured in megabytes (MB).
- CR: is the main criterion to check whether the results are consistent. It is calculated in the experiment by the same equation used in AHP (CI / RI) [28], [29].

C. Experimental Setup

The time consumption, memory usage, and CR values are evaluated by implementing and running the algorithms' codes and comparing their results. They have been written in the JAVA programming language and run on a machine with a Processor: Intel(R) Core(TM) i7-6500U CPU @ 2.50 GHz 2.60 GHz, and Installed Memory (RAM): 8.00 GB and System type: 64-bit operating system, x64-based processor.

Researchers [9], [31] considered the requirements numbers small when they are less than 20; medium when they are between 20 and 50; Otherwise, large. Several data set sizes (small, medium, and large) that range from 10 to 500 requirements are used in the experiment to prove the efficiency of E-AHP over AHP and ReDCCahp when dealing with various requirements numbers. The input will be a list of requirements objects (R1, R2, and R3…Rn), where n is the requirements number. Each requirement object has the name and score of the requirement. The requirements names are according to their order in the list. To consider the different scores for the requirements, the codes of the three algorithms have run ten times, each with different scores values, and then the average results are taken. Only the requirements numbers and their scores are important in the experiment, and it doesn't matter about their meaning.

VI. Discussion and Results

A. Discussion

This section presents and discusses the experiment results; it compares the performance of AHP, ReDCCahp, and E-AHP. It will show the time taken, memory used, and CR values of the algorithms after running their Java code with different numbers of requirements ranging from 10 to 500. Tables II and III present the average time taken and memory used by AHP, ReDDCahp, and E-AHP, respectively. Table IV presents the CR values of the AHP, ReDCCahp, and E-AHP results.

TABLE II. The Average Time Taken (in Minutes) by the AHP, ReDDCahp and E-AHP Algorithms

Number of requirements	AHP	ReDDCahp	E-AHP
10	2.501	1.391	0.297
25	9.647	3.675	0.972
50	17.533	6.988	2.326
100	27.091	13.962	4.955
150	49.081	22.491	6.883
200	74.718	28.541	13.481
300	109.981	46.846	25.236
400	136.345	61.932	36.766
500	161.709	95.961	45.152

TABLE III. The Average Memory used (in MB) by the AHP, ReDDCahp and E-AHP Algorithms

Number of requirements	AHP	ReDDCahp	E-AHP
10	0.0681	0.0344	0.0293
25	0.2133	0.0939	0.0845
50	0.3312	0.1997	0.1497
100	0.5508	0.3138	0.2349
150	0.7019	0.4609	0.2911
200	0.9402	0.5011	0.3278
300	1.3082	0.7443	0.4026
400	1.8708	0.9952	0.5133
500	2.8937	1.1960	0.6479

TABLE IV. THE AVERAGE CR VALUE OF THE AHP, REDCCAHP AND E-AHP ALGORITHMS' RESULTS

Number of requirements	AHP	ReDDCahp	E-AHP
10	0.0791	0.0504	0.0193
25	0.1944	0.0905	0.0337
50	0.2908	0.0981	0.0617
100	0.3873	0.1649	0.0729
150	0.4521	0.2591	0.0801
200	0.5091	0.3287	0.0896
300	0.6099	0.4926	0.11091
400	0.7011	0.6608	0.1482
500	0.8019	0.7492	0.1615

Charts in Fig. 3 and Fig. 4 visualize results in Table II and Table III, respectively. Chart in Fig. 5 visualizes results in Table IV. Tables II, III, Fig.3, and Fig. 4 show that ReDCCahp and E-AHP take less time and memory than AHP. When the requirements number is from 10 to 50, the difference between the time taken and memory used by the algorithms is small, larger when the number of requirements becomes > 50, and significant when > 150. That happens because they apply the grouping method, which decreases their matrix size (when the matrix size decreases, the memory and time needed to complete the operations to get the final PV decreases).

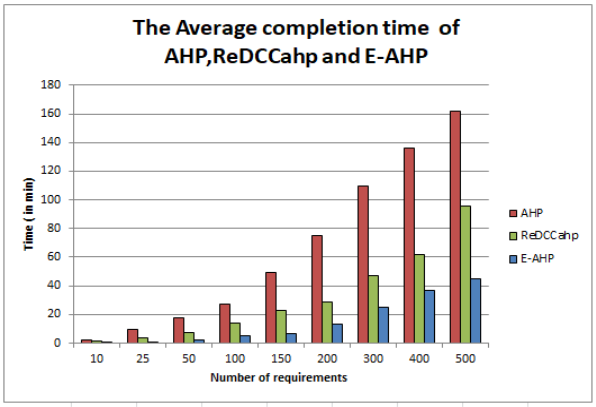

Fig. 3. The Average Time taken by the AHP, ReDCCahp and E-AHP Algorithms.

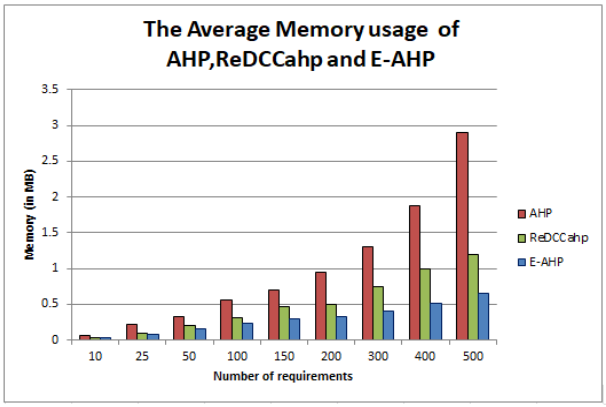

Fig. 4. The Average Memory used by the AHP, ReDCCahp and E-AHP Algorithms.

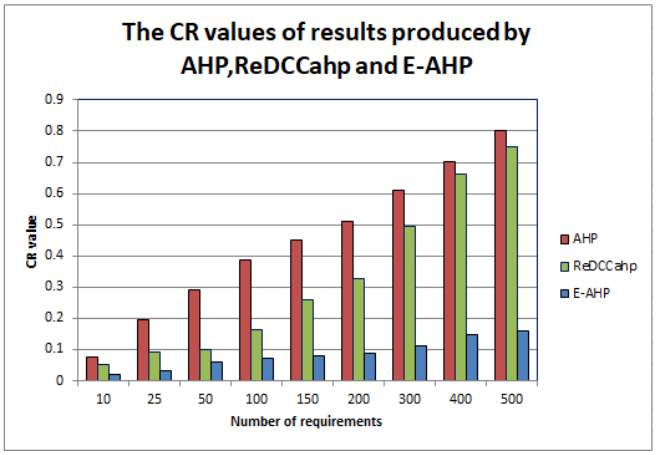

Fig. 5. The Average CR Value of the AHP, ReDCCahp and E-AHP Algorithms' Result.

It also can be noticed that E-AHP takes less time and memory than ReDCCahp, especially when the requirements number is > 100 (large). Because, in ReDCCahp, the group size is fixed (each group has only two requirements), but in E-AHP, the group size is variable (one group can have any requirements number). So in most cases, E-AHP produces less number of groups than ReDCCahp, which makes its pairwise comparison matrix size smaller than ReDCCahp.

Table IV and Fig. 5 show that the results of AHP becomes inconsistent (CR > .1) when the number of requirements is > = 25. That happens because it requires the users to make pairwise comparisons among all requirements; not among requirements groups, which increases the human error proportion; and hence decreases the results consistency. ReDCCahp produces consistent results when the number of requirements is < =50, and E-AHP produces consistent results when the requirement number is < 300. Although E-AHP and ReDCCahp both group the requirements, E-AHP produces more consistent results than ReDCCahp. That happens because E-AHP uses the scoring method instead of the pairwise comparisons, which decreases the human error proportion and increases results consistency. Moreover, the scoring method takes less time than the pairwise comparisons method, as the pairwise comparisons increase the time exponentially when the requirements number increases.

B. Results

The results that can be concluded from the experiment are as follows:

- AHP consumes more time than ReDCCahp and E-AHP, and E-AHP uses the least time among the three algorithms.
- AHP uses more memory than ReDCCahp and E-AHP, and E-AHP uses the least among the three algorithms.
- AHP produces less consistent results than ReDCCahp and E-AHP, and E-AHP produces the most consistent results among the three algorithms.
- AHP has less scalability than ReDCCahp and E-AHP, and E-AHP is the most scalable algorithm among the three algorithms.

- AHP produces less accurate results than ReDCCahp and E-AHP, and E-AHP produces the most accurate results among the three algorithms.

So among the three algorithms, E-AHP is the best one as it takes the least time, uses the least memory, has the highest scalability, and produces the most consistent and accurate results. All of that becomes remarkable when the number of requirements increases.

VII. Conclusion

This research proposes a new software requirements prioritization algorithm to solve the scalability and inconsistency problems faced by the AHP, namely E-AHP. A performance evaluation of the E-AHP algorithm against the AHP and ReDCCahp algorithms (ReDCCahp is one of the best recent algorithms that try to solve the AHP problems) was done to prove the effectiveness and efficiency of E-AHP. The java codes of the three algorithms have been implemented and run on the same machine with various requirements numbers ranging from 10 to 500 (small, medium, and large). The time taken, the memory used, and CR values of the results are measured. Results show that E-AHP takes much less time, uses less memory, and produces more consistent and accurate results than AHP and ReDCCahp, especially with large requirements numbers, which means that E-AHP has high scalability. So E-AHP is better than AHP and ReDCCahp as it can deal efficiently with large numbers of requirements.

VIII. Limitations and Future Work

Some cases will reduce the speed and accuracy of the E-AHP algorithm: first, if the difference among most of the requirements scores is more than the MaxDRS value, in this case, most of the groups will have one requirement, and the number of groups will increase (will almost equal to the requirements number), which will cause a scalability problem. Second, if the difference among most requirements scores is the same, then each group will have many requirements, which will decrease the accuracy of the results. The negative effect is reduced in these cases by choosing small values for MaxDRS and MaxNR. So in the future, we plan to enhance the E-AHP algorithm to deal with the previous cases efficiently. We also plan to conduct an experiment using a large number of software analysts as participants to validate the applicability and usability of the proposed algorithm on large real-life software projects.

References

[1] Rashidah Kasauli, Eric Knauss, Jennifer Horkoff, Grischa Liebel, Francisco Gomes de Oliveira Neto,"Requirements engineering challenges and practices in large-scale agile system development", Journal of Systems and Software, 2021.

[2] Naila Jan, Irum Inayat, and Muhammad Abbas, "An Empirical Evaluation of Requirements Prioritization Techniques." , Marketing and Branding Research,2020.

[3] Faiza Allah Bukhs , Zaharah Allah Bukhsh , and Maya Daneva, "A systematic literature review on requirement prioritization techniques and their empirical evaluation", Computer Standards,2020.

[4] Muhammad Yaseen, Noraini Ibrahim, and Aida Mustapha, "Requirements prioritization and using iteration model for successful implementation of requirements", International Journal of Advanced Computer Science and Applications, 2019.

[5] Khaled AbdElazim, Ramadan Moawad, and Essam Elfakharany, "A framework for requirements prioritization process in agile software development", Journal of Physics: Conference Series, 2020.

[6] Emmanuel OC Mkpojiogu, and Nor Laily Hashim, "Quality based prioritization: An approach for prioritizing software requirements", Journal of Telecommunication, Electronic and Computer Engineering, 2018.

[7] Hanny Tufail, Iqra Qasim, Muhammad Faisal Masood, Sara Tanvir, Wasi Haider Butt, "Towards the selection of Optimum Requirements Prioritization Technique: A Comparative Analysis.", 2019 5th International Conference on Information Management (ICIM). IEEE, 2019.

[8] Iroju Olaronke, Rhoda Ikono, Ishaya Gambo, "An Appraisal of Software Requirement Prioritization Techniques", Asian Journal of Research in Computer Science, 2018.

[9] Muhammad Imran Babar, Masitah Ghazali, Dayang N.A. Jawawi, Siti Maryam Shamsuddin, and Noraini Ibrahim, "PHandler: an expert system for a scalable software requirements prioritization process", Knowledge-Based Systems, 2015.

[10] Jamilah Din, Muhammed Basheer Jasser, "Software Requirements Prioritization Tool using a Hybrid Technique ", International Journal of Engineering and Advanced Technology (IJEAT), 2019.

[11] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Nazri Mahrin, "A systematic literature review of software requirements prioritization research", Information and software technology, 2014.

[12] Nayak, Soumen, Chiranjeev Kumar, and Sachin Tripathi. "Analytic hierarchy process-based regression test case prioritization technique enhancing the fault detection rate.", Soft Computing 26.15, 2022.

[13] Fadhl Hujainah, Rohani Binti Abu Bakar, Mansoor Abdullateef Abdulgabber,and Kamal Z. Zamli, "Software requirements prioritisation: a systematic literature review on significance, stakeholders, techniques and challenges.", IEEE Access ,2018.

[14] Noor Hazlini Borhan, Hazura Zulzalil, Sa'adah Hassan, Norhayati Mohd Ali,"Requirements Prioritization Techniques Focusing on Agile Software Development:A Systematic Literature review ", International Journal of Scientific and Technology Research, 2019.

[15] Amjad Hudaib, Raja M.T Masadeh, Mais Qasem, and Abdullah Issa Alzaqebah, "Requirements prioritization techniques comparison." Modern Applied Science, 2018.

[16] Muhammad Atif Iqbal, Athar Mohsin Zaidi, and Saeed Murtaza, "A new requirement prioritization model for market driven products using analytical hierarchical process", 2010 International Conference on Data Storage and Data Engineering .IEEE, 2010.

[17] Srinivas Nidhra, Likith Poovanna, Kelapanda Satish, and Vinay Sudha Ethiraj, "Analytical Hierarchy Process issues and mitigation strategy for large number of requirements", 2012 CSI Sixth International Conference on Software Engineering (CONSEG). IEEE, 2012.

[18] Naila Jan, Irum Inayat,and Muhammad Abbas, ", An Empirical Evaluation of Requirements Prioritization Techniques ,Marketing and Branding Research", 2020.

[19] Iyas Ibriwesh, Sin-Ban Ho, and Ian Chai, "Overcoming scalability issues in analytic hierarchy process with ReDCCahp: An empirical investigation", Arabian Journal for Science and Engineering, 2018.

[20] Xiaojun Wang, Hing Kai Chan, Rachel W.Y. Yee , and Ivan Diaz-Rainey, "A two-stage fuzzy-AHP model for risk assessment of implementing green initiatives in the fashion supply chain", International Journal of Production Economics, 2012.

[21] Yash Veer Singh, Bijendra Kumar, Satish Chand, and Jitendra Kumar ,"A comparative analysis and proposing 'ANN fuzzy AHP model'for requirements prioritization.", I.J. Information Technology and Computer Science, 2018.

[22] Mukhtar Elsood, A. Abo, Hesham A. Hefny, and Eman S. Nasr, "A goal-based technique for requirements prioritization", 2014 9th International Conference on Informatics and Systems. IEEE, 2014.

[23] YanLiu, Claudia M.Eckert,and ChristopherEarl. "A review of fuzzy AHP methods for decision-making with subjective judgements", Expert Systems with Applications, 2020.

[24] Paolo Tonella, Angelo Susi, and Francis Palma, "Using interactive GA for requirements prioritization", 2nd International Symposium on Search Based Software Engineering. IEEE, 2013.

[25] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar, "A review on genetic algorithm: past, present, and future", Multimedia Tools and Applications, 2021.

[26] Muhammad Yaseen , Noraini Ibrahim , Aida Mustapha, "Prioritization of Software Functional Requirements: Spanning Tree based Approach ",International Journal of Advanced Computer Science and Applications,2019.

[27] Lafore, Robert. Data structures and algorithms in Java. Sams publishing, 2017.

[28] Thomas L. Saaty, "What is the analytic hierarchy process? ", Mathematical models for decision support.Springer, Berlin, Heidelberg, 1988.

[29] Bruce L. Golden, Edward A. Wasil, and Patrick T. Harker, "The analytic hierarchy process", Applications and Studies, Berlin, Heidelberg, 1989.

[30] Chatelin, Françoise, ed. Eigenvalues of matrices: revised edition. Society for Industrial and Applied Mathematics, 2012.

[31] Qiao. Ma, The effectiveness of requirements prioritization techniques for a medium to large number of requirements: a systematic literature review", Diss. Auckland University of Technology, 2009.

[32] Kurt. Mehlhorn, "Data structures and algorithms 1: Sorting and searching", Springer Science & Business Media, 2013.