ReproTizer: A Fully Implemented Software Requirements Prioritization Tool

Philip Achimugu, Ali Selamat(✉), and Roliana Ibrahim

Faculty of Computing, Universiti Teknologi Malaysia,
81310 Johor Bahru, Johor, Malaysia
check4philo@gmail.com, {aselamat,roliana}@utm.my

Abstract. Before software is developed, requirements are elicited. These requirements could be over-blown or under-estimated in a way that meeting the expectations of stakeholders becomes a challenge. To develop a software that precisely meets the expectations of stakeholders, elicited requirements need to be prioritized. When requirements are prioritized, contract breaches such as budget over-shoot, exceeding delivery time and missing out important requirements during implementation can be totally avoided. A number of techniques have been developed but these techniques do not addresses some of the crucial issues associated with real-time prioritization of software requirements such as computational complexities and high time consumption rate, inaccurate rank results, inability of dealing with uncertainties or missing weights of requirements, scalability problems and rank update issues. To address these problems, a tool known as ReproTizer (Requirements Prioritizer) is proposed to engender real-time prioritization of software requirements. ReproTizer consist of a WS (Weight Scale) which avails project stakeholders the ability to perceive the influence, different requirements weights may have on the final results. The WS combines a single relative weight decision matrices to determine the weight vectors of requirements with an aggregation operator (AO) which computes the global weights of requirements. The tool was tested for scalability, computational complexity, accuracy, time consumption and rank updates. Results of the performance evaluation showed that the tool is highly reliable (98.89 % accuracy), scalable (prioritized over 1000 requirements), less time consumption and complexity ranging from 500–29,804 milliseconds (ms) of total prioritization time and able to automatically update ranks whenever changes occurs. Requirements prioritization, a multi-criteria decision making task is therefore an integral aspect of the requirements engineering phase of the development life cycle phases. It is used for software release planning and leads to the development of software systems based on the preferential requirements of stakeholders.

Keywords: Software · Requirements · Prioritization · Tool · Stakeholders

Submitted to Transactions on Computational Collective Intelligence (TCCI) Journal.

N.T. Nguyen and R. Kowalczyk (Eds.): TCCI XXII, LNCS 9655, pp. 80–105, 2016.
DOI: 10.1007/978-3-662-49619-0_5

1 Introduction

During requirement elicitation, there are more prospective requirements specified for implementation by relevant stakeholders with limited time and resources. Therefore, a meticulously selected set of requirements must be considered for implementation and planning for software releases with respect to available resources. This process is referred to as requirements prioritization. It is considered to be a complex multi-criteria decision making process (Perini et al. 2013).

There are so many advantages of prioritizing requirements before architecture design or coding. Prioritization aids the implementation of a software system with preferential requirements of stakeholders (Ahl 2005; Thakurta 2012). Also, the challenges associated with software development such as limited resources, inadequate budget, insufficient skilled programmers among others makes requirements prioritization really important (Karlsson et al. 2007). It can help in planning for software releases since not all the elicited requirements can be implemented in a single release due to some of these challenges (Berander et al. 2006; Karlsson and Ryan 1997). It also enhances budget control and scheduling (Perini et al. 2013). Therefore, determining which, among a pool of requirements to be implemented first and the order of implementation is necessary to avoid breach of contract or agreement during the development processes. Furthermore, software products that are developed based on prioritized requirements can be expected to have a lower probability of being rejected. To prioritize requirements, stakeholders will have to compare them in order to determine their relative importance through a weight scale which is eventually used to compute the prioritized requirements (Kobayashi and Maekawa 2001). These comparisons becomes complex with increase in the number of requirements (Kassel and Malloy 2003).

Software system's acceptability level is mostly determined by how well the developed system has met or satisfied the specified requirements. Hence, eliciting and prioritizing appropriate requirements and scheduling right releases with the correct functionalities are a critical success factor for building formidable software systems. In other words, when vague or imprecise requirements are implemented, the resulting system will fall short of user's or stakeholder's expectations. Many software development projects have enormous prospective requirements that may be practically impossible to deliver within the expected time frame and budget (Perini et al. 2013; Tonella et al. 2012). It therefore becomes highly necessary to source for appropriate measures for planning and rating requirements in an efficient way.

A number of techniques have been proposed in the literature by authors and scholars, yet many areas of improvement have also been identified to optimize the prioritization processes. With the advent of Internet and quest for software that can service distributed organizations, the number of stakeholders in large-scale projects have drastically increased and requirements are beginning to possess the attributes of evolving due to innovation, technological advancement or business growth. Therefore, prioritization techniques should be able to generate an ordered list of requirements based on the relative weights provided by the project stakeholders at any point during the development life cycle (Perini et al. 2013; Ahl 2005).

The rest of the paper is organized as follows: Sect. 2 discusses the related works while Sect. 3 describes the proposed technique. Section 4 presents an illustrative example of the proposed technique and Sect. 5 describes the attributes of the support tool. Section 6 presents performance evaluation of ReproTizer; Sect. 7 compares the strengths of ReproTizer over existing ones while Sect. 8 concludes the paper and identify areas for future research.

2 Related Work

Many requirements prioritization techniques exist in the literature. All of these techniques utilize a ranking process to prioritize candidate requirements. The ranking process is usually executed by assigning weights across requirements based on pre-defined criteria, such as value of the requirements perceived by relevant stakeholders or the cost of implementing each requirement. From the literature; analytic hierarchy process (AHP) is the most prominently used technique. However, this technique suffers bad scalability. This is due to the fact that, AHP executes ranking by considering the criteria that are defined through an assessment of the relative priorities between pairs of requirements. This becomes impracticable as the number of requirements increases. It also does not support requirements evolution or rank updates but provide efficient or reliable results (Karlsson et al. 1998). Also, all techniques suffer from rank updates issue. This term refers to the inability of a technique to update rank status of ordered requirements whenever a requirement is added or deleted from the list. Prominent techniques that suffer from this limitation are PHandler (Babar et al. 2015), Case base ranking (Perini et al. 2013); Interactive genetic algorithm prioritization technique (Tonella et al. 2012); Binary search tree (Karlsson et al. 1998); Cost value approach (Karlsson and Ryan 1997) and EVOLVE (Greer and Ruhe 2004). Furthermore, existing techniques are prone to computational errors (Ramzan et al. 2011) probably due to lack of robust algorithms. Karlsson et al. (1998) conducted some researches where certain prioritization techniques were empirically evaluated. From their research, they reported that, most of the prioritization techniques apart from AHP and bubble sorts produce unreliable or misleading results while AHP and bubble sorts were also time consuming. The authors then posited that; techniques like hierarchy AHP, spanning tree, binary search tree, priority groups produce unreliable results and are difficult to implement. Babar et al. (2011) were also of the opinion that, techniques like requirement triage, value intelligent prioritization and fuzzy logic based techniques are also error prone due to their reliance on experts and are time consuming too. Planning game has a better variance of numerical computation but suffer from rank updates problem. Wieger's method and requirement triage are relatively acceptable and adoptable by practitioners but these techniques do not support rank updates in the event of requirements evolution as well. Lim and Finkelstein (2012) proposed a method known as StakeRare which stands for Stakeholder Recommender assisted method for requirements elicitation. It is a requirements prioritization method for large projects, where stakeholders can be in different locations and rank requirements based on a 5-point Likert scale. The authors also implemented the concept of StakeRare method into a support tool known as StakeSource2.0 (Lim et al. 2011), which is a web-based

tool that supports the StakeRare method. However, the method and tool were not tested for large scale prioritization of requirements. The focus was more on numbers of stakeholders than requirements. Additionally, the proposed approach and tool was not tested with various requirements and scenarios of different organizations.

Our motivation for proposing an improved method and tool arose from the limitations of existing techniques as enumerated below:

(i) Scalability: Techniques like AHP, pairwise comparisons and bubblesort suffer from scalability problems because, requirements are compared based on possible pairs causing n (n − 1)/2 comparisons (Karlsson et al. 1998). For example, when the number of requirements is doubled in a list, other techniques will only require double the effort or time for prioritization while AHP, pairwise comparisons and bubblesort techniques will require four times the effort or time. This is bad scalability.

(ii) Computational complexity: Most of the existing prioritization techniques are actually time consuming in the real world (Karlsson et al. 1998). Ahl (2005) executed a comprehensive experimental evaluation of five different prioritization techniques namely; AHP, binary search tree, planning game, $100 (cumulative voting) and a new method which combines planning game and AHP (PgcAHP), to determine their ease of use, accuracy and scalability. The author went as far as determining the average time taken to prioritize 13 requirements across 14 stakeholders with these techniques. At the end of the experiment; it was observed that, planning game was the fastest while AHP was the slowest. Planning game prioritized 13 requirements in about 2.5 min while AHP prioritized the same number of requirements in about 10.5 min. In other words, planning game technique took only 11.5 s to compute the priority scores of one requirement across 14 stakeholders while AHP consumed 48.5 s to accomplish the same task due to pair comparisons.

(iii) Rank updates: Perini et al. (2013) defined rank update as 'anytime' prioritization; that is, the ability of a technique to automatically update ranks anytime a requirement is included or excluded from the list. This situation has to do with requirements evolution. Therefore, existing prioritization techniques are incapable of updating or reflecting rank status whenever a requirement is introduced or deleted from the rank list. Therefore, it does not support iterative updates. This is very critical because, decision making and selection processes cannot survive without iterations. Therefore, a good and reliable prioritization technique should be one that supports rank updates. This limitation seems to cut across most existing techniques.

(iv) Error proneness: Existing prioritization techniques are also prone to errors (Ramzan et al. 2011). This could be due to the fact that, the rules governing the requirements prioritization processes in the existing techniques are not robust enough. This has also led to the generation of unreliable prioritization results because; such results do not reflect the true ranking of requirements from stakeholder's point of view or assessment after the ranking process. Therefore robust algorithms are required to generate reliable prioritization results.

(v) Lack of fully implemented support tools: From the literature, it was observed that most existing prioritization techniques have not been really implemented for real-life scenarios probably because of the complexities associated with prioritizations and the time required for generating prioritized requirements. Therefore, there is need to implement algorithms that will improve or support requirements prioritization at commercial or industrial level (Peng 2008; Racheva et al. 2008; Ramzan et al. 2009). Before these algorithms can work efficiently, the methods for capturing requirements in an unambiguous way must be well thought of (Grunbacher et al. 2003) since the output of prioritization processes depend on the input and the aim is to plan for software releases (Barney et al. 2006) as well as the successful development of software products in line with negotiated or prioritized requirements (Olson and Rodgers 2002).

3 Proposed Technique

The proposed technique consist of six steps (Fig. 1). The first step is to input the consensus requirements and the criteria describing the expected functionalities of each requirement into ReproTizer. The second step determines the relative value of requirements by indicating the preference weights against requirements using the weight scale (WS) in Table 1. The third step calculates the requirements priority vector, normalize the respective weights and calculate the global weights of requirements (Weight vector). The fourth step elicits the performance of each requirements with respect to the global weights, using a classical weighted average decision matrix (WADM). The fifth and sixth step aggregates and determine the ranks of requirements respectively.

The WS was designed to handle prioritization in both real time and fuzzy conditions. We consider a finite collection of requirements X = {R_{11}, R_{12}…. R_{1k}} that has to be

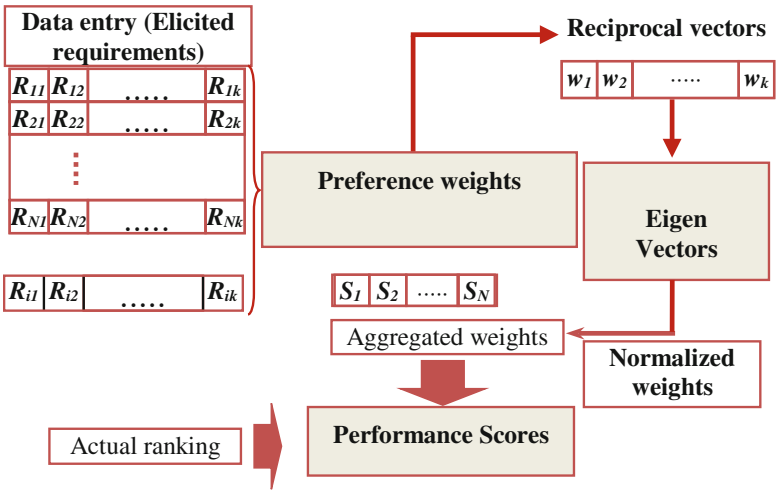

Fig. 1. Proposed technique

ranked against one each other. Our approach consist of set of input R_{11}, R_{12}, …, R_{1k}, associated with their respective weights w_1, w_2, …, w_k that represents stakeholders' preferences and a WADM required to calculate the global scores across requirements. The requirement (R_{11} …, R_{21}…, …, R_{nk}) represent input data that are ranked using the AO and stored in the database. In this approach, we assume that, the stakeholder's preferences are expressed as relative weights, which are values between 5 and 1.

Table 1. Weight scale (WS)

Terms	Numeric rating	Fuzzy weights
Extremely high (EH)	5	(1,1,1)
Very high (VH)	4	(1/2, 1, 1/3)
High (H)	3	(1/5, 1/2, 1/3)
Fair (F)	2	(1/7, 1/3, 1/5)
Low (L)	1	(1/9, 1/4, 1/7)

The data required for the prioritization process comes from the preference weights of stakeholders which could be imprecise, uncertain and vague due to incomplete information, time limitations, lack of knowledge, or understanding about the system under development. The harmonic mean (HM) is determined to replace requirements with missing weights. This is meant to cater for vagueness associated with requirements. It is a multi-criteria decision making approach for analyzing the hierarchy of the decision-making process. The proposed approach is used to model the interaction, dependence and feedback within groups of elements and between groups. The groups and elements can be considered as project stakeholders and requirements respectively. Thereafter, the relationships and values between these elements are constructed using a decision matrix. The elements within a group can have a mutual impact on members of the group and the other groups with respect to each of several characteristics. The stakeholder's judgments on the assessment of requirements in the decision-making process always involve incomplete, imprecise, uncertain, intangible and tangible information. Therefore, the conventional approaches seems inadequate to handle the stakeholder's judgments explicitly. To model the uncertainty of stakeholder's relative weights of requirements, harmonic mean computation is integrated into the relative weight scoring process which makes the proposed approach avoid missing weights. The judgment is described through weight numbers where the harmonic mean is used to determine the weights of requirements that were not scored by the stakeholders. Hence, the decision-making process described by the proposed approach is more realistic and capable of generating accurate results.

3.1 Algorithmic Steps of the Computational Process

Step 1: Given a prioritization event E with Requirements R_1, R_2, R_3, …, R_n (i.e. n – Requirements) and Stakeholders S_1, S_2, S_3, …, S_u (i.e. u – number of Stakeholders), the

Table 2. Preference weights of requirements

R_1	5	EH	R_{n-1}	R_n
R_2	5	EH		
R_3	5	EH		
R_4	4	VH		

relative or preference weights of requirements are indicated by the project stakeholders as follows:

The weights in Table 2 is for one stakeholders across 4 requirements as an example. For each stakeholder, the proposed approach computes a decision matrix of all the requirements by applying Eq. 1.

$$rank_{j.s_i} \tag{1}$$

Where $1 \leq j \leq NoOfRequirements$ and
$1 \leq i \leq NoOfStakeholders$

Step 2: The sum of the ranks of each requirement is computed across the project stakeholders using Eq. 2.

$$rankSum_j = \sum_{i=1}^{u} rank_j.s_i \tag{2}$$

Step 3: The reciprocals of the relative weights are determined to minimize the discrepancies of the final ranks by using Eq. 3 and decision matrix is formed as shown in Table 3.

$$reciprocalSum_j = \frac{1}{n}\sum_{i=1}^{u} rank_j.s_i \tag{3}$$

n stands for the number of requirements undergoing prioritization.

Step 4: The Square of the matrix is computed using Eq. 4 and the sum of each row of the matrix is calculated using Eq. 5 which will yield a result of $(n \times 1)$ matrix, known as the Eigenvector. It represents the global weights of requirements.

$$SquareM = \left(\prod_{i=1}^{n} a_k^2\right) \tag{4}$$

$$SumM = \left(\sum_{i=1}^{n} a_k\right) \tag{5}$$

Table 3. Reciprocals of the preference weights

	S_1	S_2	S_3	…	S_n
R_1	$\frac{1}{n}\sum_{i=1}^{u} rank_1.s_1$	$\frac{1}{n}\sum_{i=1}^{u} rank_1.s_2$	$\frac{1}{n}\sum_{i=1}^{u} rank_1.s_3$	…	$\frac{1}{n}\sum_{i=1}^{u} rank_1.s_{n-1}$
R_2	$\frac{1}{n}\sum_{i=1}^{u} rank_2.s_{2i}$	$\frac{1}{n}\sum_{i=1}^{u} rank_2.s_2$	$\frac{1}{n}\sum_{i=1}^{u} rank_2.s_3$	…	$\frac{1}{n}\sum_{i=1}^{u} rank_1.s_{n-2}$
R_3	$\frac{1}{n}\sum_{i=1}^{u} rank_3.s_3$	$\frac{1}{n}\sum_{i=1}^{u} rank_3.s_2$	$\frac{1}{n}\sum_{i=1}^{u} rank_3.s_3$	…	$\frac{1}{n}\sum_{i=1}^{u} rank_1.s_{n-3}$
…	…	…	…	…	…
R_n	$\frac{1}{n}\sum_{i=1}^{u} rank_j.s_i$	$\frac{1}{n}\sum_{i=1}^{u} rank_j.s_i$	$\frac{1}{n}\sum_{i=1}^{u} rank_j.s_i$	…	$rankSum_{n\ -\ k}$

Step 5: The Eigenvectors are normalized using Eq. 6. Meaning, the sum of all the values in the Eigenvector is calculated and used to divide each of the values in the Eigenvector. This places all the values on a scale of 1 and the sum of all the values to 1.

$$\aleph_j = \frac{w_j}{\sum_{j=1}^{n} w_j = 1} i = 1, \ldots n; j = 1, \ldots m \tag{6}$$

Step 6: This obtains the performance scores for the requirements by summing the relative normalized weights (wj) of each requirement across the stakeholders using Eq. 7.

$$p_i = \sum_{i=1}^{n} w_j \tag{7}$$

4 Illustrative Example

This section presents an illustrative example for prioritizing software requirements with the proposed approach. For the sake of clarity, let us consider 4 requirements to be prioritized by 3 stakeholders. The requirements are *usability*, *scalability*, *security* and *modularity*. Since this is an example, the elicited weights for step 1 is just illustrative and represent opinions of stakeholders. In the implemented tool, the user dialog is achieved with a simplified interface weights scale, shown in Fig. 2 but at the back end, the calculations are performed using the computational processes described in Sect. 3.

It is important to note that in Step 1, stakeholders are only required to provide the preference weights of requirements and the proposed technique automatically perform relevant calculations in order to display the prioritized requirements. Table 4 presents the illustrative preference weights of stakeholders while Table 5 shows the rank sum of weights for the 3 stakeholders using Eq. 2. Table 6 shows the reciprocal values for the rank sum using Eq. 3 and Eq. 4 was used to compute the square matrix of requirements

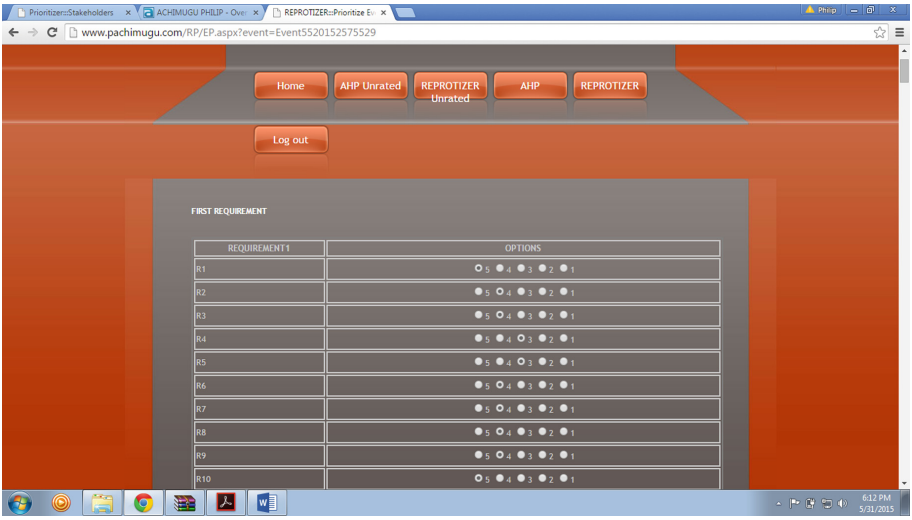

Fig. 2. Simplified interface weights scale of the proposed technique

as displayed in Table 7. The relative normalized decision matrix shown in Table 8 was computed using Eqs. 5 and 6 respectively while final scores for the requirements displayed in Table 9 were computed using Eq. 7. From the final scores of requirements, it can be easily seen that the stakeholders ranked usability and security as the most valued requirements followed by scalability and then modularity. In terms of the accuracy of the proposed approach, it can be seen that the original weights provided by the stakeholders in Table 4 is in agreement with the final scores in Table 9.

Table 4. Preference weights of requirements

	Usability	Scalability	Security	Modularity
Stakeholder 1	5	5	5	5
Stakeholder 2	5	4	5	4
Stakeholder 3	5	4	5	3

Table 5. Rank sum of requirements

	Usability	Scalability	Security	Modularity
Stakeholder 1	15	15	15	15
Stakeholder 2	15	12	15	12
Stakeholder 3	15	12	15	9

Table 6. Reciprocal values of the requirements' sum

	Usability	Scalability	Security	Modularity
Stakeholder 1	3.75	3.75	3.75	3.75
Stakeholder 2	3.75	3.00	3.75	3.00
Stakeholder 3	3.75	3.00	3.75	2.25

Table 7. Square matrix of the requirements' sum

	Usability	Scalability	Security	Modularity
Stakeholder 1	14.06	14.06	14.06	14.06
Stakeholder 2	14.06	9.00	14.06	9.00
Stakeholder 3	14.06	9.00	14.06	5.06

Table 8. Normalized weights

	Usability	Scalability	Security	Modularity
Stakeholder 1	0.304	0.304	0.304	0.304
Stakeholder 2	0.305	0.195	0.305	0.195
Stakeholder 3	0.333	0.213	0.333	0.120

Table 9. Performance scores

Requirements	Final Scores
Usability	0.942
Scalability	0.712
Security	0.942
Modularity	0.619

5 Tool Support

The tool was implemented in C#, very similar to Java platform standard edition 7. It takes relative weights of requirements provided by the stakeholders as input and processes them to generate list of prioritized requirements. The tool is deployed at http://www.pachimugu.com/. It provides a convenient way of accessing various menus of the tool from the HTML of the page. Additionally, a pattern matching was utilized to aid the re-weighting and re-computation of ranks whenever requirements evolves. The tool also provides an avenue for inclusion or exclusion of stakeholders if need be using three step process; (1) New stakeholders are added by the administrator as soon as they get registered as users. The proposed tool can cater for as much stakeholders as required for a particular software project. (2) The consensus requirements automatically appears against their names so as to initiate the scoring process. (3) The relative weights are then processed or computed to display the final ranks of requirements. However, deleting a stakeholder also applies to the relative weights of that stakeholder where the tool automatically re-compute the new ranks of each requirement based on the new number of stakeholders. The tool's main window is displayed in Fig. 3.

Considering the top-most part of the window, it can be observed that the name of the tool is known as Requirements Prioritizer, consisting of five tabs namely; *Home*, *Events*, *Login*, *Sign Up* and *Contact*. To use this tool, prospective project stakeholders would have to first register by clicking the *sign up* tab to fill the required details. Once this is done, the tools' administrator can now view all the registered stakeholders and

Fig. 3. Proposed tool main window

add them up. It is also the duty of the administrator to input the elicited requirements to undergo prioritization into ReproTizer. Requirements are inputted into ReproTizer by clicking the tab *add event* where the name of the project is used to save the inputted requirements. It can be observed from the window that the tool is flexible enough to cater for addition or deletion of requirements or stakeholders at any point in time where ReproTizer simply updates the ranks status of requirements by displaying new ordered list of requirements that has occurred either by adding or deleting a requirement or stakeholder. A concept Perini et al. described as "anytime prioritization" (Perini et al. 2013). Once, all the requirements have been inputted into ReproTizer and all the registered project stakeholders have been accepted by the administrator, the scoring of requirements can be initiated by stakeholders who logs into ReproTizer with their respective username and password. Figure 4 shows a window of the database where the registered stakeholders are stored.

Once the stakeholders log into ReproTizer, they can now view the consensus requirements in order to rank or score them. Figure 5 presents the window that shows the individual weights of stakeholders. The assessment of these requirements lead to the construction of a decision matrix. This is where the tradeoffs between the requirements are displayed. ReproTizer displays both the individual and overall ratings of each requirements. The individual weights signifies the ranks of the requirements by one stakeholder. ReproTizer automatically calculates the overall weights of requirements by aggregating the scores across all project stakeholders in chronological order (Fig. 6). If the requirements weights are inconsistent or missing, a message pops-up warning the user.

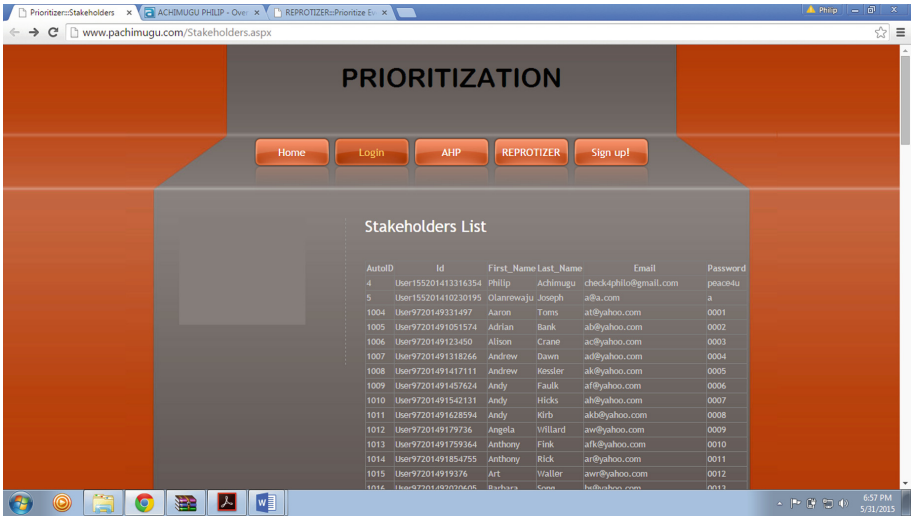

Fig. 4. Registered stakeholders

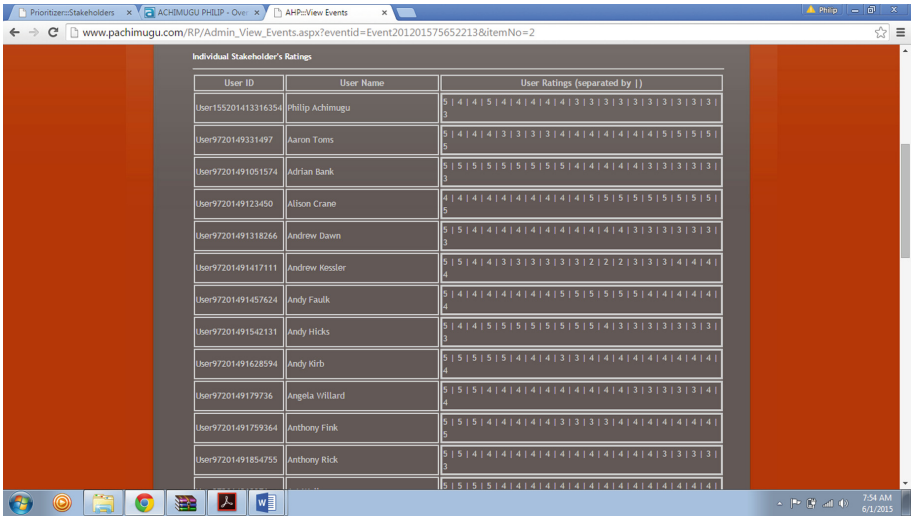

Fig. 5. Individual weights of requirements

6 Performance Evaluation

The motivation for developing ReproTizer was as a result of the following limitations of existing techniques as described in Sect. 2 (scalability, computational complexity, rank updates, error proneness and lack of fully implemented support tools). Therefore, the evaluation of ReproTizer is based on these parameters.

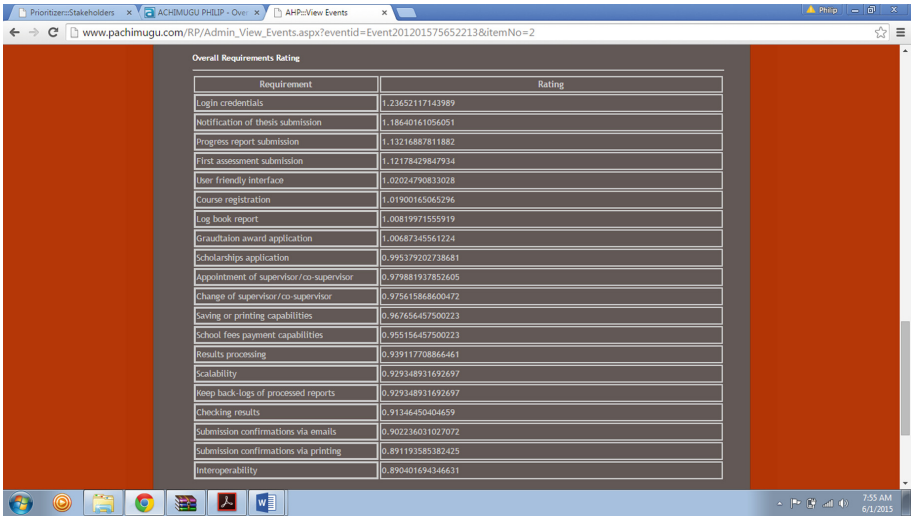

Fig. 6. Overall weights of requirements (Final ranks)

Various authors have executed a comparative analysis of the different software requirements prioritization techniques in order to measure the performance of these techniques. In this section, some well-known requirements prioritization techniques are considered and compared with ReproTizer based on the five evaluation criteria mentioned above. Consequently, *scalability* is measured in terms of the number of requirements ReproTizer can accommodate at runtime. *Computational complexity* measures the time consumed in executing the computational processes or calculations of the weighted requirements to generate the prioritized list. *Rank updates* has to do with the ability of ReproTizer to effect or generate new ranks whenever a requirement or stakeholder is included or excluded from the list. *Error proneness* measures the accuracy of the ranked results while lack of fully implemented support tools has to do with the absence of tool capable of supporting real-time prioritization of software requirements.

In order to evaluate the performance of ReproTizer, 4 experiments were conducted with different requirements datasets. The first experiment was conducted with 20 requirements from GSMS project (A web-based Graduate Students' information Management System in Universiti Teknologi Malaysia) and 100 requirements from a health information system (HIS) software. The second experiment was conducted with 200 requirements from RALIC project (an access/identity card software for university staff and students in University College London). The third and fourth experiment were conducted with 500 and then, 1000 requirements of an enterprise resource planning (ERP) software package. These experiments were meant to prove the contributions of ReproTizer with respect to the limitations highlighted in Table 10. As it can be seen, a lot of techniques suffer scalability problems. Most techniques are only suitable for small to medium sized software projects. To address scalability issues, Babar and colleagues proposed an expert system known as PHandler which was able to prioritize

up to 500 requirements; the highest so far in the literature (Babar et al. 2015). However, if requirements run up to thousands, it is not certain that PHandler can provide desired results on that scale. PHandler was not also tested for computational complexities, rank updates and time consumption; although, their system was only meant to address scalability issue inherent in existing techniques. In terms of time consumption, a

Table 10. Limitations of existing techniques

Techniques and references	Limitations
AHP (Saaty 1980; Karlsson et al. 1998), Binary tree (Beg et al. 2009; Aasem et al. 2010), Case based ranking (Perini et al. 2013), Interactive requirements prioritization (Tonella et al. 2013), Cost-Value Ranking (Karlsson and Ryan 1997), StakeSource2.0 (Lim et al. 2011), Fuzzy AHP (Lima et al. 2011), Quality Functional Deployment (QFD) (Edwin 1992), Ranking, Requirement uncertainty prioritization approach (RUPA) (Voola and Babu 2012), Round-the-Group Prioritization (Hatton 2008; Karlsson and Ryan 1997), $100 Allocation or Cumulative Voting (Berander and Andrews 2005; Regnell et al. 2001)	Not scalable, 10–100 requirements only
Cost-Value Ranking (Karlsson and Ryan 1997), AHP (Saaty 1980; Karlsson et al. 1998), Binary search tree (Duan et al. 2009)	Time consuming
EVOLVE (Thakurta 2013, Greer and Ruhe 2004), Wiegers' matrix approach (Duan et al. 2009)	Computationally complex
Hierarchy AHP (Karlsson et al. 1998), Minimal spanning tree (Karlsson et al. 1998), Multi-criteria Preference Analysis Requirements Negotiation (MPARN) (In and Olson 2002), Pair Wise Analysis (Karlsson and Ryan 1997), Quality Functional Deployment (QFD) (QFD) (Edwin 1992), Simple multi-criteria rating technique by swing (SMARTS) (Avesani et al. 2005), Top ten requirements (Berander 2004), Value based requirements prioritization (Kukreja et al. 2012), WinWin (Gruenbacher 2000)	Error prone
TOPSIS (Kukreja 2013; Kukreja et al. 2012), Requirements triage (Karlsson et al. 2004), PHandler (Babar et al. 2015)	Lack of implemented tool, do not recall or update ranks and time consumption rate was not measured

number of techniques are also limited in this area. Some studies confirmed that most techniques are time consuming (Ramzan et al. 2011; Soni 2014; Kyosev 2014; Dabbagh and Lee 2014). Specifically, AHP, cumulative voting, numerical assignment, ranking, top-ten, Theory-W, planning game, requirements triage, Wieger's method and value based requirement prioritization techniques consumes a lot of time during the prioritization process (Ramzan et al. 2011). Furthermore, the systematic literature review executed by Achimugu et al. (2014) have it that most techniques suffer from rank inaccuracies, computational complexities, rank updates, scalability, requirements dependencies among others.

Requirements for software projects 1–4 were inputted into ReproTizer. This was followed by the indication of preference weights against each requirements where ReproTizer was automatically able to display prioritized requirements based on the individual and overall weights of requirements. We have observed that results of prioritization often get faulty when requirements increases due to computational complexities and lack of efficient algorithms. However, in the case of ReproTizer, Figs. 7a and 7b show the average accuracy and time consumed for prioritizing 20 requirements while Fig. 7c shows that ReproTizer is automatically able to update rank status when requirements evolves. Similarly, Figs. 8a, 8b; 9a, 9b and 10a, 10b show the average accuracy and time consumed by ReproTizer for prioritizing 200, 500 and 1000 requirements respectively while Figs. 8c, 9c and 10c confirmed that ReproTizer is capable of updating rank status when requirements changes on a large scale. For the time consumption, it took ReproTizer 0.39 min (23.4 s) to prioritize 500 requirements (Fig. 9b) while 0.49 min (29.4 s) was exhausted in prioritizing 1000 requirement (Fig. 10b). The time difference between prioritizing 500 and 1000 requirements is 1 min which is expected because the requirements are doubled. This would almost mean that, for every 500 requirements; 1 additional minute is consumed by ReproTizer to produce the desired results. This is good response time achieved by implementing improved formulas and algorithms Therefore, we conclude that, a fully implemented support tool with high accuracy, good response time and user-friendlier interface for software requirements prioritization has been developed. Also, using a six-step approach, ReproTizer is able to automatically calculate the weights of requirements and perform trade-offs in all steps with minimized divergence in prioritized requirements.

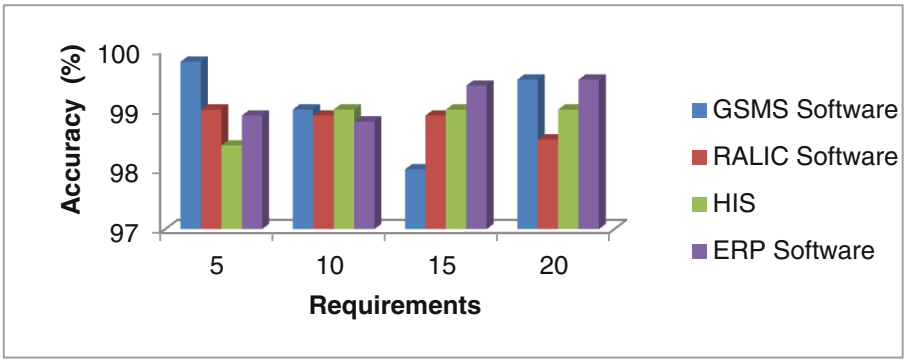

Fig. 7a. Prioritization accuracy for 20 requirements (Color figure online)

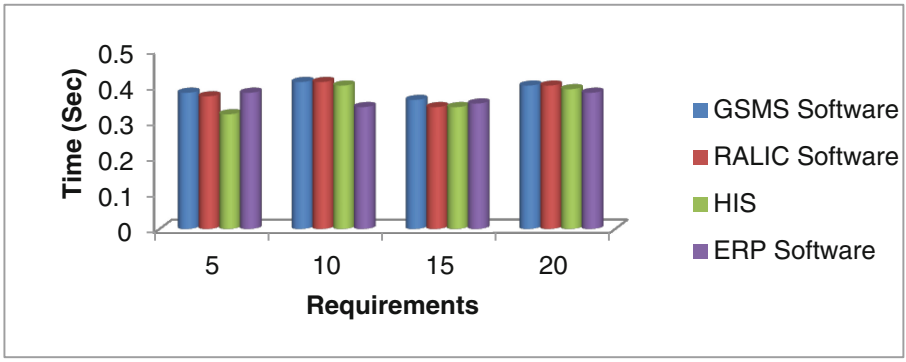

Fig. 7b. Time taken for prioritizing 20 requirements (Color figure online)

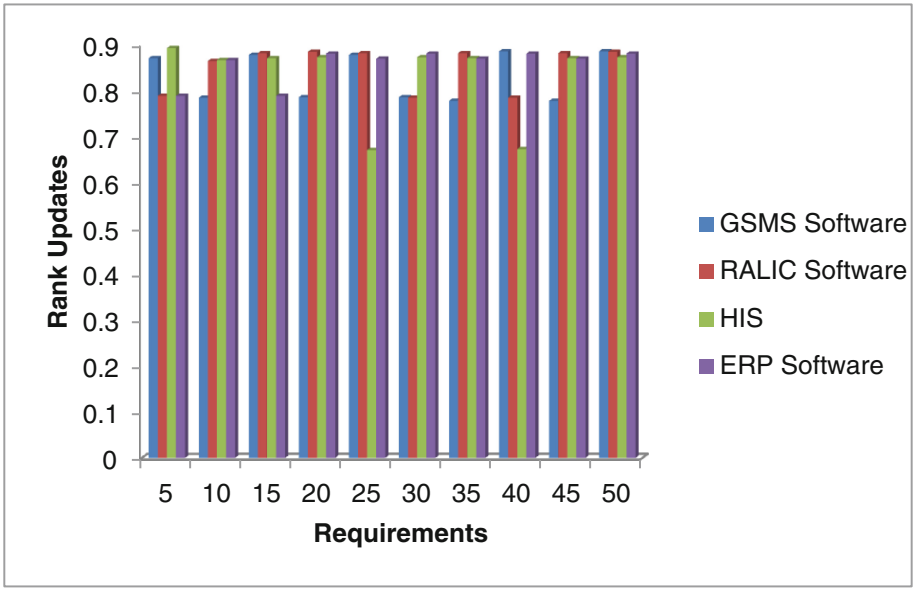

Fig. 7c. Automatic rank updates for 50 requirements (Color figure online)

All our experiments were carried out on a computer with a 2.4 GHz processor and 4 GB RAM. We have observed that ReproTizer consumed an average run time ranging from 500–29,804 milliseconds (ms) to prioritize requirements on a large scale. Table 11 shows the average runtime of three major components that constitute ReproTizer. The average runtime for the decision matrix includes time taken for the construction of preference weights of requirements. Similarly, average runtime for computing the normalized decision matrix includes time taken for constructing a new matrix which subjects the summation of all the preference weights of a requirement to 1 while the global decision matrix stands for the average time of computing the final

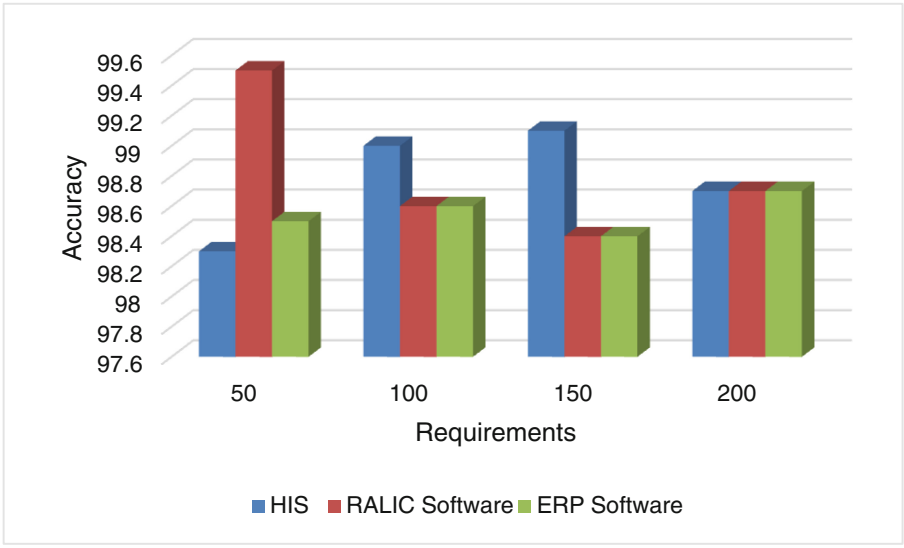

Fig. 8a. Prioritization accuracy for 200 requirements (Color figure online)

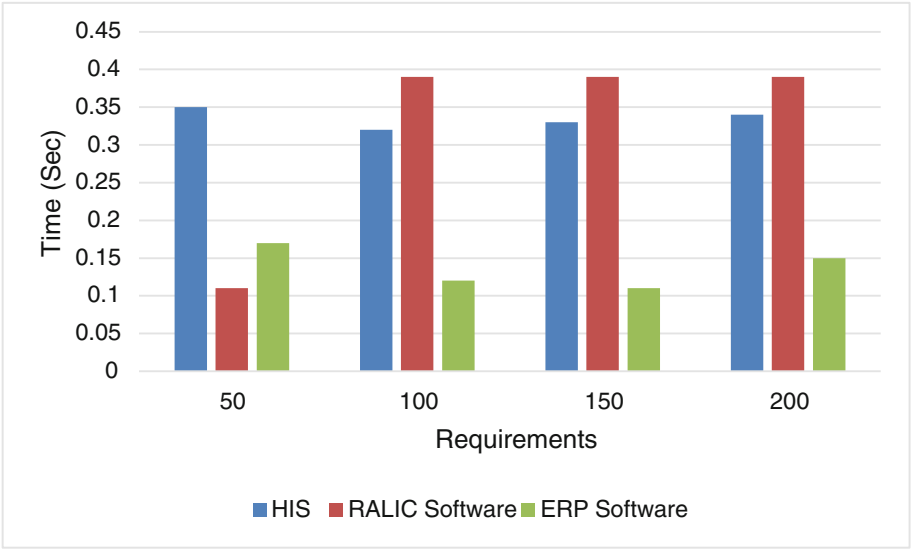

Fig. 8b. Time taken for prioritizing 200 requirements (Color figure online)

weights of requirements. Among these three modules, the discrepancy rate is highly minimal with high correlation between the relative and final weights. Therefore, ReproTizer produces good response time with reduced complexities.

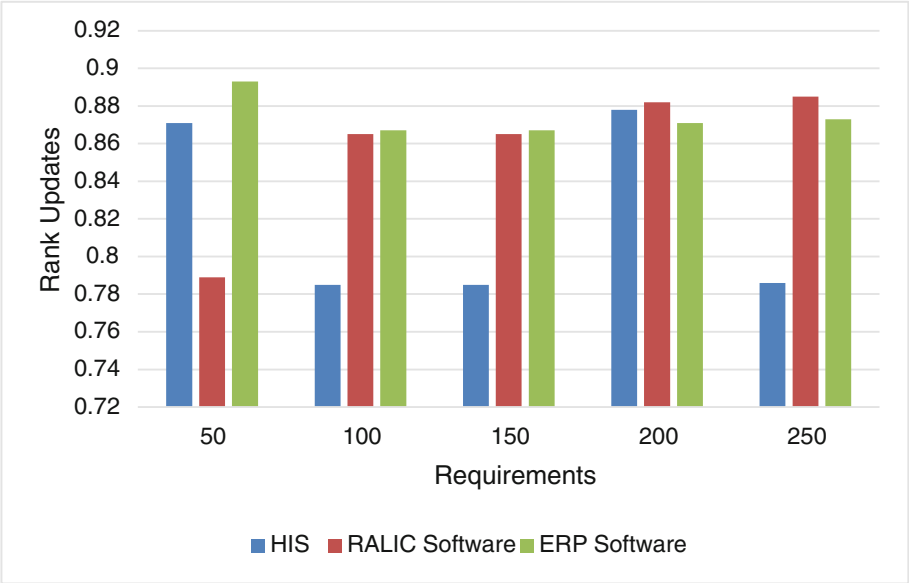

Fig. 8c. Automatic rank updates for 250 requirements (Color figure online)

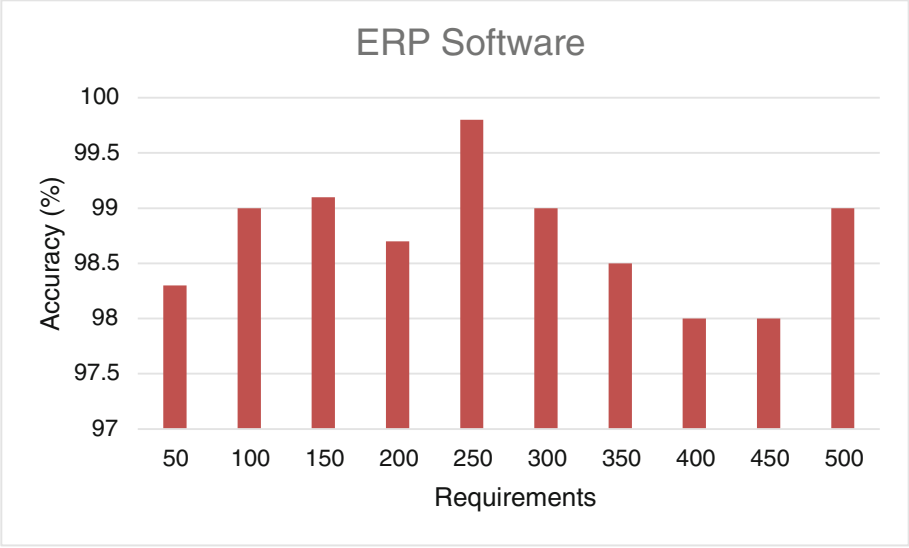

Fig. 9a. Prioritization accuracy for 500 requirements

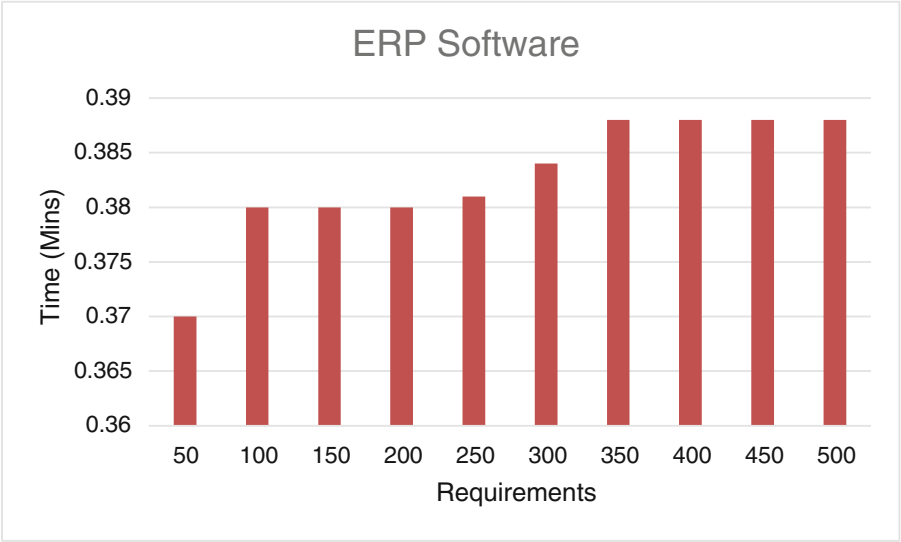

Fig. 9b. Time taken for prioritizing 500 requirements

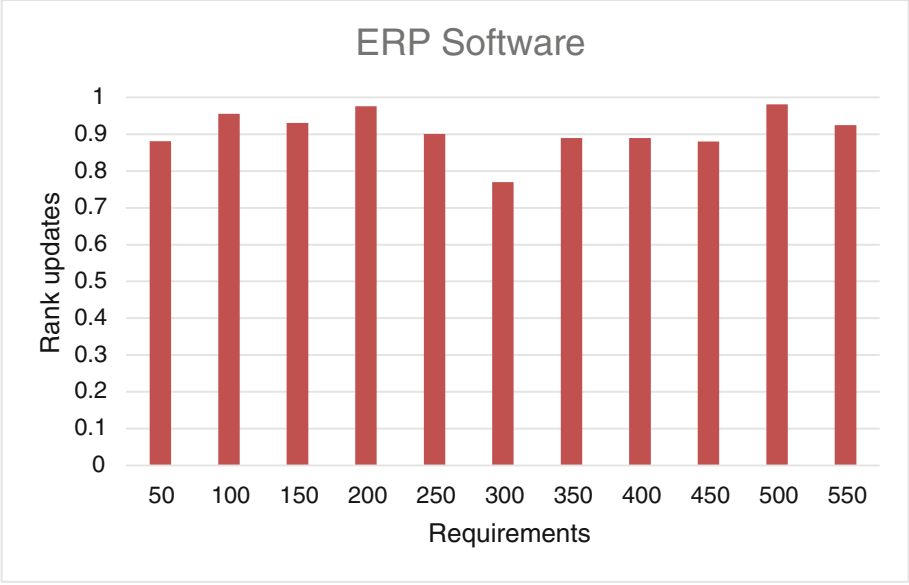

Fig. 9c. Automatic rank updates for 550 requirements

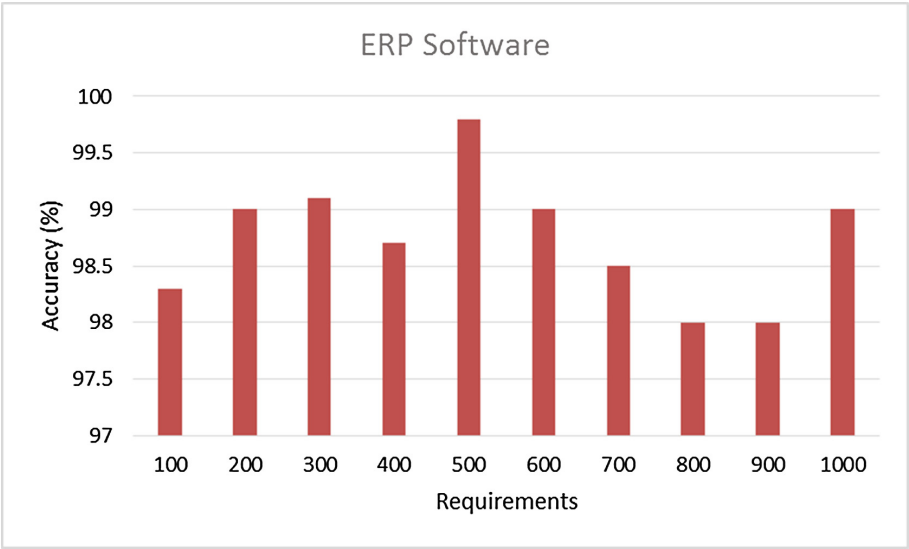

Fig. 10a. Prioritization accuracy for 1000 requirements

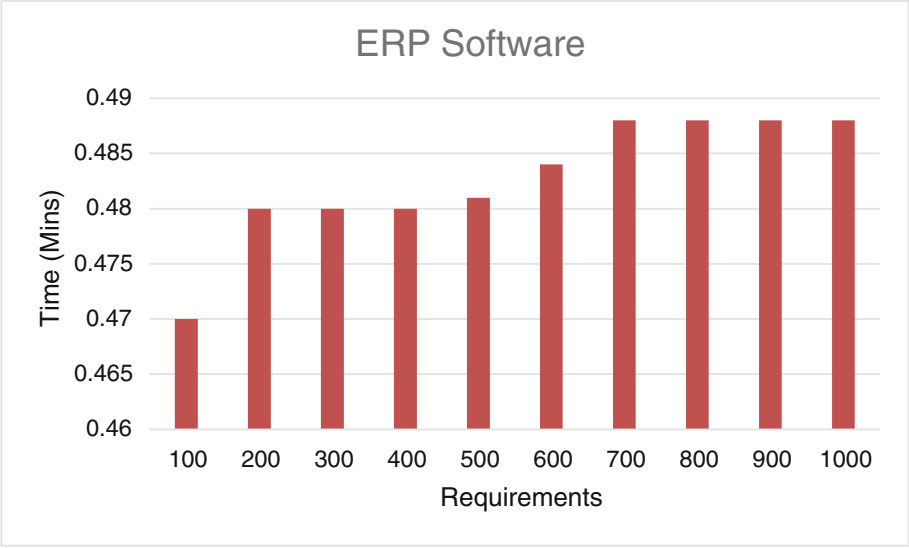

Fig. 10b. Time taken for prioritizing 1000 requirements

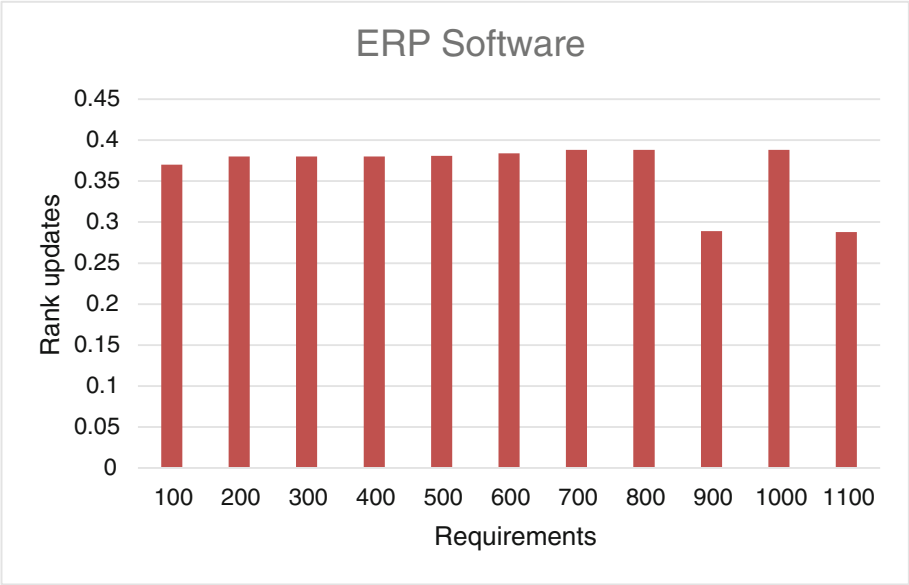

Fig. 10c. Automatic rank updates for 1100 requirements

Table 11. Average runtime behaviour of the modules

Components	Average time (Milliseconds)
Decision matrix	3192–3443
Normalized decision matrix	2290–894
Aggregate decision matrix	500–29,804

7 Comparison with Existing Techniques

The relative performance of the proposed tool with respect to other techniques is shown in Table 12. The relative performance is measured based on the number of requirements, accuracy and time consumed by the techniques during requirements prioritization. From the table, PHandler is seen to be the most scalable technique in literature. The expert system is capable of prioritizing up to 500 requirements at runtime with average accuracy of 93.89 %. This makes PHandler system about 80 % better than existing techniques in terms of the number of requirements it is capable of accommodating. However, PHandler was not tested for rank updates, time consumption and complexity. Meaning, if requirements scale up to thousands, it is not sure if PHandler would produce the desired results at that scale. This forms the rationale for developing a support tool capable of prioritizing more numbers of requirements. Hence, ReproTizer was developed and evaluated with 5 different software projects requirements ranging from small, medium and to large scale. The average accuracy of ReproTizer was 98.89 % even on a large scale. It was also able to accommodate and prioritize over

1000 requirements with less complexity between 500–29,804 ms thereby producing good response time. When compared to other techniques in literature, the capabilities of ReproTizer are eminent. Besides, the computational processes, formulas and algorithm are simple but robust enough to be used in practice. The tool has been fully implemented and deployed online, available for use by software practitioners on real-life basis. The performance of the proposed tool was generally evaluated based on number of requirements, time consumption and computational complexities and rank updates. Based on these evaluation parameters, it is clear that ReproTizer is much better and would be beneficial in practice.

Table 12. Comparative analysis of prioritization techniques.

Source	Technique	No of requirements	Accuracy	Time consumption	Support tool
(Ramzan et al. 2011)	Intelligent requirement prioritization	Not indicated	90 %	90 Work hours	×
(Ramzan et al. 2011)	Theory W	Not indicated	80 %	160 Work hours	×
(Perini et al. 2009)	AHP	20	85 %	37 min	√
(Ramzan et al. 2011)	Cumulative voting	Not indicated	85 %	120 Work hours	×
(Ramzan et al. 2011)	Wieger's method	Not indicated	85 %	100 Work hours	×
(Perini et al. 2009)	Case-Based Ranking	20	Not indicated	10 min	√
(Perini et al. 2013)	Case-Based Ranking	25, 50, 100	80 %	Not measured	×
(Tonella et al. 2012)	Interactive GA-Based Prioritization	26, 23, 21 and 49	97.20 %	Not measured	×
(Lim and Finkelstein 2012; Lim et al. 2011)	StakeRare, StakeSource 2.0	<50	80 %	Not measured	√
(Babar et al. 2015)	PHandler	14, 25, 50, 100, 200, 400, 500	93.89	Not measured	×
This Study	ReproTizer	20, 50, 100, 200, 500, 1000	98.89 %	500–29,804 ms (0.5–29.804 s)	√

8 Conclusion/Future Work

The aim of this research was to identify the limitations of existing prioritization techniques so as to address them. It was eventually discovered that existing techniques actually suffer from mainly scalability problems, large disparity or disagreement between ranked weights, rank reversals, as well as unreliable results. These were all

taken into cognizance during the course of developing ReproTizer. The method utilized in this research consisted of intelligent algorithms implemented with C# and MicrosoftSQL server 2012. Efficient models were formulated in order to enhance the reliability of the proposed approach. The developed tool was designed and implemented to cater large requirements and stakeholders. It is easy to use with friendlier user interface, reduced computational complexities and has addressed rank reversals issues. For the future work, we hope to validate the tool in a real-life setting with large numbers of stakeholders and requirements alike. Finally, the developed tool is able to classify ranked requirements in chronological order with an accompanied graph to visualize the prioritized results at a glance. For dependency issues, requirements are thoroughly analyzed using factor analysis to track redundant, conflicting, independent and dependent requirements before inputting the requirements into ReproTizer.

Acknowledgement. The Universiti Teknologi Malaysia (UTM) under Research University funding vot number 02G31 and Ministry of Higher Education (MOHE) Malaysia under vot number 4F550 are hereby sincerely acknowledged for providing the research funds to complete this research.

References

Perini, A., Ricca, F., Susi, A., Bazzanella, C.: An empirical study to compare the accuracy of AHP and CBRanking techniques for requirements prioritization. In: Proceedings of the Fifth International Workshop on Comparative Evaluation in Requirements Engineering, pp. 23–35. IEEE (2007)

Ruhe, G., Eberlein, A., Pfahl, D.: Trade-off analysis for requirements selection. Int. J. Softw. Eng. Knowl. Eng. **13**(4), 345–366 (2003)

Ahl, V.: An experimental comparison of five prioritization methods–investigating ease of use, accuracy and scalability. Master's thesis, School of Engineering, Blekinge Institute of Technology, Sweden, August 2005

Berander, P., Khan, K.A., Lehtola, L.: Towards a research framework on requirements prioritization. In: Proceedings of Sixth Conference on Software Engineering Research and Practice in Sweden (SERPS 2006), October 2006

Kobayashi, M., Maekawa, M.: Need-based requirements change management. In: Proceedings of ECBS 2001 Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 171–178 (2001)

Kassel, N.W., Malloy, B.A.: An approach to automate requirements elicitation and specification. In: Proceedings of the 7th IASTED International Conference on Software Engineering and Applications, Marina Del Rey, CA, USA, 3–5 November 2003

Perini, A., Susi, A., Avesani, P.: A machine learning approach to software requirements prioritization. IEEE Trans. Softw. Eng. **39**(4), 445–460 (2013)

Tonella, P., Susi, A., Palma, F.: Interactive requirements prioritization using a genetic algorithm. Inf. Softw. Technol. Inf. Softw. Technol. **55**, 173–187 (2012)

Babar, M.I., Ghazali, M., Jawawi, D.N., Shamsuddin, S.M., Ibrahim, N.: PHandler: an expert system for a scalable software requirements prioritization process. Knowl.-Based Syst. **84**, 179–202 (2015)

Kaur, G., Bawa, S.: A survey of requirement prioritization methods. Int. J. Eng. Res. Technol. **2**(5), 958–962 (2013)

Voola, P., Babu, A.: Requirements uncertainty prioritization approach: a novel approach for requirements prioritization. Softw. Eng. Int. J. (SEIJ) **2**(2), 37–49 (2012)

Thakurta, R.: A framework for prioritization of quality requirements for inclusion in a software project. Softw. Qual. J. **21**, 573–597 (2012)

Ramzan, M., Jaffar, A., Shahid, A.: Value based intelligent requirement prioritization (VIRP): expert driven fuzzy logic based prioritization technique. Int. J. Innovative Comput. **7**(3), 1017–1038 (2011)

Perini, A., Ricca, F., Susi, A.: Tool-supported requirements prioritization: comparing the AHP and CBRank method. Inf. Softw. Technol. **51**, 1021–1032 (2009)

Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Inf. Softw. Technol. **46**(4), 243–253 (2004)

Franceschini, F., Rupil, A.: Rating scales and prioritization in QFD. Int. J. Qual. Reliab. Manage. **16**(1), 85–97 (1999)

Karlsson, J., Wohlin, C., Regnell, B.: An evaluation of methods for prioritizing software requirements. Inf. Softw. Technol. **39**(14), 939–947 (1998)

Kukreja, N., Payyavula, S., Boehm, B., Padmanabhuni, S.: Value-based requirements prioritization: usage experiences. Procedia Comput. Sci. **16**, 806–813 (2012)

Kukreja, N.: Decision theoretic requirements prioritization: a two-step approach for sliding towards value realization. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 1465–1467. IEEE Press (2013)

Dabbagh, M., Lee, S.: An approach for integrating the prioritization of functional and nonfunctional requirements. Sci. World J. (2014)

Voola, P., Vinaya Babu, A.: Interval evidential reasoning algorithm for requirements prioritization. In: Satapathy, S.C., Avadhani, P.S., Abraham, A. (eds.) Proceedings of the InConINDIA 2012. AISC, vol. 132, pp. 915–922. Springer, Heidelberg (2012)

Aasem, M., Ramzan, M., Jaffar, A.: Analysis and optimization of software requirements prioritization techniques. In: 2010 International Conference on Information and Emerging Technologies (ICIET), pp. 1–6. IEEE (2010)

Racheva, Z., Daneva, M., Herrmann, A., Wieringa, R.: A conceptual model and process for client-driven agile requirements prioritization. In: 2010 Fourth International Conference on Research Challenges in Information Science (RCIS), pp. 287–298. IEEE (2010)

Otero, C., Dell, E., Qureshi, A., Otero, L.: A quality-based requirement prioritization framework using binary inputs. In: 2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation (AMS), pp. 187–192. IEEE (2010)

Carod, N., Cechich, A.: Cognitive-driven requirements prioritization: a case study. In: 2010 9th IEEE International Conference on Cognitive Informatics (ICCI), pp. 75–82. IEEE (2010)

Gaur, V., Soni, A., Bedi, P.: An agent-oriented approach to requirements engineering. In: 2010 IEEE 2nd International Advance Computing Conference (IACC), pp. 449–454 (2010)

Beg, M., Verma, R., Joshi, A.: Reduction in number of comparisons for requirement prioritization using B-Tree. In: IEEE International Advance Computing Conference, 2009, IACC 2009, pp. 340–344. IEEE (2009)

Hatton, S.: Choosing the right prioritisation method. In: 19th Australian Conference on Software Engineering, 2008, ASWEC 2008, pp. 517–526. IEEE (2008)

Daneva, M., Herrmann, A.: Requirements prioritization based on benefit and cost prediction: a method classification framework. In: EUROMICRO-SEAA, pp. 240–247. IEEE (2008a)

Beg, R., Abbas, Q., Verma, R.P.: An approach for requirement prioritization using b-tree. In: First International Conference on Emerging Trends in Engineering and Technology, 2008, ICETET 2008, pp. 1216–1221. IEEE (2008)

Laurent, P., Cleland-Huang, J., Duan, C.: Towards automated requirements triage. In: 15th IEEE International Requirements Engineering Conference, 2007, RE 2007, pp. 131–140 (2007)

Avesani, P., Bazzanella, C., Perini, A., Susi, A.: Facing scalability issues in requirements prioritization with machine learning techniques. In: RE 2005, pp. 297–306 (2005)

Avesani, P., Bazzanella, C., Perini, A., Susi, A.: Supporting the requirements prioritization process: a machine learning approach. In: Proceedings of 16th International Conference on Software Engineering and Knowledge Engineering, SEKE 2004, pp. 306–311. KSI Press, Banff (2004)

Moisiadis, F.: The Fundamentals of prioritizing requirements. In: Proceedings of Systems Engineering Test and Evaluation Conference, SETE 2002 (2002)

Aaron, K.M., Paul, N., Anton, A.I.: Prioritizing legal requirements. In: Second International Workshop on Requirements Engineering and Law, 2009, RELAW 2009, pp. 27–32. IEEE (2009)

Svahnberg, M., Karasira, A.: A study on the importance of order in requirements prioritisation. In: 2009 Third International Workshop on Software Product Management (IWSPM), pp. 35–41. IEEE (2009)

Tonella, P., Susi, A., Palma, F.: Using interactive GA for requirements prioritization. In: 2010 Second International Symposium on Search Based Software Engineering (SSBSE), pp. 57–66. IEEE (2010)

Bebensee, T., van de Weerd, I., Brinkkemper, S.: Binary priority list for prioritizing software requirements. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 67–78. Springer, Heidelberg (2010)

Duan, C., Laurent, P., Cleland-Huang, J., Kwiatkowski, C.: Towards automated requirements prioritization and triage. Requir. Eng. **14**(2), 73–89 (2009)

Carod, N., Cechich, A.: Requirements Prioritization Techniques (2001)

Karlsson, L., Thelin, T., Regnell, B., Berander, P., Wohlin, C.: Pair-wise comparisons versus planning game partitioning-experiments on requirements prioritisation techniques. Empir. Softw. Eng. **12**(1), 3–33 (2007)

Lehtola, L., Kauppinen, M.: Suitability of requirements prioritization methods for market-driven software product development. Softw. Process Improv. Pract. **11**(1), 7–19 (2006)

Berander, P., Andrews, A.: Requirements prioritization. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 69–94. Springer, Heidelberg (2005)

Lehtola, L., Kauppinen, M., Kujala, S.: Requirements prioritization challenges in practice. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 497–508. Springer, Heidelberg (2004)

Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE Softw. **14**, 67–74 (1997)

Saaty, T.L.: The Analytic Hierarchy Process. McGraw-Hill, New York (1980)

Herrmann, A., Daneva, M.: Requirements prioritization based on benefit and cost prediction: an agenda for future research. In: RE 2008, pp. 125–134. IEEE Computer Society (2008b)

Wiegers, K.E.: First things first: prioritizing requirements. Softw. Dev. **7**(9) (1999). www.processimpact.com/pubs.shtml#requirements

Regnell, B., Host, M., Dag, J.: An industrial case study on distributed prioritization in market-driven requirements engineering for packaged software. Requir. Eng. **6**, 51–62 (2001)

Edwin, D.: Quality function deployment for large systems. In: International Engineering Management Conference 1992, Eatontown, NJ, USA, 25–28 October 1992

Olson, H., Rodgers, T.: Multi-criteria preference analysis for systematic requirements negotiation. In: COMPSAC 2002, pp. 887–892 (2002)

Berander, P.: Prioritization of Stakeholder Needs in Software Engineering. Understanding and Evaluation. Licenciate Thesis, Blekinge Institute of Technology, Sweden, Licentiate Series, 12 (2004)

Karlsson, J., Olsson, S., Ryan, K.: Improved practical support for large scale requirements prioritizing. J. Requir. Eng. **2**, 51–67 (1997)

Peng, S.: Sample selection: an algorithm for requirements prioritization. ACM (2008)

Racheva, Z., Daneva, M., Buglione, L.: Supporting the dynamic reprioritization of requirements in agile development of software products. In: Second International Workshop on Software Product Management, 2008, IWSPM 2008, pp. 49–58. IEEE (2008)

Lim, S.L., Finkelstein, A.: StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation. IEEE Trans. Softw. Eng. **38**(3), 707–735 (2012)

Kyosev, T.H.: Comparing Requirements Prioritization Methods in Industry: A study of the Effectiveness of the Ranking Method, the Binary Search Tree Method and the Wiegers Matrix. MSc Thesis, Negometrix BV, Germany (2014)

Babar, M., Ramzan, M., Ghayyur, S.: Challenges and future trends in software requirements prioritization. In: 2011 International Conference on Computer Networks and Information Technology (ICCNIT), pp. 319–324. IEEE (2011)

Gruenbacher, P.: Collaborative requirements negotiation with easy winwin. In: Proceedings of 2nd International Workshop on the Requirements Engineering Process, Greenwich London, September 2000

Lima, D.C., Freitas, F., Campos, G., Souza, J.: A fuzzy approach to requirements prioritization. In: Cohen, M.B., Ó Cinnéide, M. (eds.) SSBSE 2011. LNCS, vol. 6956, pp. 64–69. Springer, Heidelberg (2011)

Barney, S., Aurum, A., Wohlin, C.: Quest for a silver bullet: creating software product value through requirements selection. In: 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2006. pp. 274–281. IEEE (2006)

Karlsson, L., Berander, P., Regnell, B., Wohlin,C.: Requirements prioritization: an experiment on exhaustive pair wise comparisons versus planning game partitioning. In: Proceedings of Empirical Assessment in Software Engineering (EASE 2004), Edinburgh, Scotland (2004)

Grunbacher, P., Halling, M., Biffl, S., Kitapci, H., Boehm, B.: Repeatable quality assurance techniques for requirements negotiations. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 9 p. IEEE (2003)

Ramzan, M., Arfan, J., AlIliad, I., Anwar, S., Shahid, A.: Value based fuzzy requirement prioritization and its evaluation framework. In: Fourth International Conference on Innovative Computing, Information and Control. pp. 1464–1468 (2009)

Achimugu, P., Selamat, A., Ibrahim, R., Mahrin, M.N.R.: A systematic literature review of software requirements prioritization research. Inf. Softw. Technol. **56**(6), 568–585 (2014)

Lim, S.L., Damian, D., Finkelstein, A.: StakeSource 2.0: using social networks of stakeholders to identify and prioritise requirements. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 1022–1024. ACM (2011)

Soni, A.: An evaluation of requirements prioritisation methods. Int. J. Innovative Res. Adv. Eng. **1**(10), 402–411 (2014)