Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

RePizer: a framework for prioritization of software requirements*

Saif Ur Rehman KHAN†‡1, Sai Peck LEE1, Mohammad DABBAGH1,
Muhammad TAHIR2, Muzafar KHAN3, Muhammad ARIF4

(1Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia)
(2Faculty of Computing and Information Technology, University of Jeddah, Jeddah 21589, Saudi Arabia)
(3College of Computer and Information Sciences (Muzahmiyah Branch), King Saud University, Riyadh 11362, Saudi Arabia)
(4Department of Computer Science, University of Gujrat, Gujrat 50700, Pakistan)
†E-mail: saif_rehman@siswa.um.edu.my

Received May 18, 2015; Revision accepted Oct. 27, 2015; Crosschecked July 20, 2016

Abstract: The standard software development life cycle heavily depends on requirements elicited from stakeholders. Based on those requirements, software development is planned and managed from its inception phase to closure. Due to time and resource constraints, it is imperative to identify the high-priority requirements that need to be considered first during the software development process. Moreover, existing prioritization frameworks lack a store of historical data useful for selecting the most suitable prioritization technique of any similar project domain. In this paper, we propose a framework for prioritization of software requirements, called RePizer, to be used in conjunction with a selected prioritization technique to rank software requirements based on defined criteria such as implementation cost. RePizer assists requirements engineers in a decision-making process by retrieving historical data from a requirements repository. RePizer also provides a panoramic view of the entire project to ensure the judicious use of software development resources. We compared the performance of RePizer in terms of expected accuracy and ease of use while separately adopting two different prioritization techniques, planning game (PG) and analytical hierarchy process (AHP). The results showed that RePizer performed better when used in conjunction with the PG technique.

Key words: Software requirements, Requirements prioritization techniques, Prioritization framework, Planning game, Analytical hierarchy process
http://dx.doi.org/10.1631/FITEE.1500162 **CLC number:** TP311

1 Introduction

Requirements engineering plays a crucial role in the success of a software system by understanding and managing various stakeholders' needs and wishes (Achimugu *et al.*, 2014; Dabbagh and Lee, 2015). Based on the elicited requirements, software project development is planned from its inception to its closure phase. There are large complex software-intensive systems with thousands of individual requirements (Arias *et al.*, 2011). Software development organizations need to consider every single requirement to ensure the success of a software project. Incomplete and changing requirements can lead to project failure (Dominguez, 2009). Furthermore, the key factors, including limited project resources, a long project schedule, a low requirements engineering budget, and different levels of importance among the requirements, could affect the implementation of software requirements (Firesmith, 2004). Software

‡ Corresponding author

* Project supported by the Ministry of Education, Malaysia (No. UM.C/625/1/HIR/MOHE/FCSIT/13) and the Bright Sparks Program of University of Malaya, Malaysia (No. BSP-151(3)11)

ORCID: Saif Ur Rehman KHAN, http://orcid.org/0000-0002-9643-6858 development organizations aim to achieve higher customer satisfaction by addressing high-priority requirements first (Lehtola, 2006). However, it remains a challenging task for development organizations to meet all the requirements specified by stakeholders due to time and resource constraints (Otero *et al.*, 2010). Consequently, requirements prioritization is seen as the main remedy for these problems and has been recognized as one of the most important decision-making processes during the software development process (Achimugu *et al.*, 2014).

Requirements prioritization has been identified as the most frequently addressed topic in the requirements engineering domain (Daneva *et al.*, 2014). It plays a crucial role in software release planning by selecting a set of important requirements. Finally, the selected set of requirements is implemented in the subsequent planned release of a software system (Bourque and Fairley, 2014). The following scenarios are useful in understanding requirements proritization activities using incremental and sequential process models:

Scenario 1 In the case of an incremental process model, a subset of requirements is prioritized at a time to support multiple releases of a software system (Sommerville, 2010). Generally, software requirements continuously evolve due to technology advancement and changing business needs. Therefore, it is impossible to implement all requirements in a sequential manner. Suppose a requirements engineer has a set of 'n' requirements (Fig. 1) and wants to implement them for the planned three releases (R) of a software system. In this situation, the requirements engineer divides the 'n' requirements into three blocks (B) (i.e., 1 to k, k+1 to m, and m+1 to n, where k, m, $n\in\mathbb{N}$ and $k<m<n$). Subsequently, the requirements engineer needs to prioritize each block of requirements using a suitable prioritization technique for each planned release of a software system.

Scenario 2 In the case of a sequential process model, an entire block of requirements is prioritized (Sommerville, 2010). In this situation, the requirements engineer considers the set of requirements as one block and thus needs to prioritize the complete block of requirements for the planned single release of a software system (Fig. 2).

The above scenarios illustrate that requirements engineers divide the requirements into one or several blocks to determine high-value requirements, while considering resource constraints.

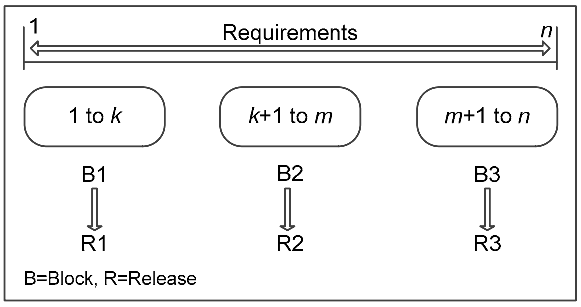

Fig. 1 Planning of software requirements for multiple releases

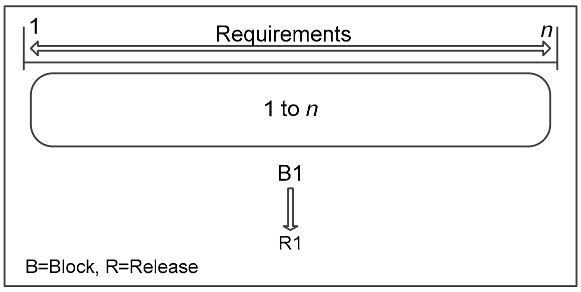

Fig. 2 Planning of software requirements for a single release

In this paper, we propose a framework called RePizer for prioritization of requirements. The framework accepts requirements and a prioritization technique as inputs to rank the requirements. Finally, RePizer produces a list of prioritized requirements for a given software project. To show the applicability of RePizer, we considered the requirements specification document of the Library of Congress (LCPAIG, 2003) as a case study. In addition, we evaluated the performance of RePizer in terms of expected accuracy and ease of use while adopting each of two different prioritization techniques, planning game (PG) and analytical hierarchy process (AHP).

The main contributions of this paper include:

1. providing a formal definition of the requirements prioritization problem,

2. comparing current state-of-the-art prioritization frameworks,

3. proposing a requirements prioritization framework by formally defining its components, and

4. evaluating the performance of the proposed framework while adopting two different prioritization techniques by conducting a real case study.

2 Background

According to Young (2004), a requirement is "a statement that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility to a customer or user". Stakeholders include customers, end-users, engineers, third parties, and all other people who have some influence on the system requirements (Gorschek, 2006). Requirements prioritization is defined as "an activity during which the most important requirements for the system (release) should be identified" (Sommerville, 2010).

Published textual descriptions of prioritization and related terms are not consistent. Various researchers have provided different definitions (Lauesen, 2002; Ahl, 2005) and concepts (Ramzan *et al.*, 2011; Creswell, 2013), which might be ambiguous for many researchers. Therefore, there is a need to formally define the requirements and associated prioritization problem. Borrowing the concepts from Khan *et al.* (2009), we formally represent the requirements prioritization problem as

Given: requirements set R,
a set of permutations of R (PR),
a function f from PR to the real numbers.
Problem: Find $R'\in\text{PR}$ such that
$$\forall R''\ (R''\in\text{PR}, R''\neq R')\ [f(R')\geq f(R'')].$$

Here PR represents the set of all possible prioritizations of original requirements R, and f is a function that, when applied to R' ($R'\subseteq R$), results in an award value for that prioritization (Khan *et al.*, 2009). The function f is an objective function because it is based on specific criteria, which is the set of existing requirements prioritization techniques, including cumulative voting (Berander and Jönsson, 2006), analytical hierarchy process (Saaty, 2008), numerical assignment (Brackett, 1990; Lehtola and Kauppinen, 2006), and planning game (Mead, 2006).

To prioritize the requirements, different dimensions should be considered, which may vary among stakeholders. Several dimensions for requirements prioritization have been published, including personal preference, business value, time-to-market, associated risk, implementation cost, dependencies, stability, and type of requirements (i.e., functional or non-functional).

During the requirements prioritization phase, a requirements engineer faces a number of challenges, such as difficulties in quantifying the associated benefits and risks, a trade-off between user/market-driven and technology-driven requirements in each product release, dependencies and relative priorities, prioritization of multi-versioned requirements, and limited knowledge of stakeholders (Lehtola, 2006).

3 State-of-the-art requirements prioritization techniques and frameworks

3.1 Requirements prioritization techniques

Researchers have proposed a number of requirements prioritization techniques (Karlsson and Ryan, 1997; Lauesen, 2002; Leffingwell and Widrig, 2003; Ahl, 2005; Berander and Andrews, 2005; Mead, 2006; Berander, 2007; Saaty, 2008; Wiegers and Beatty, 2013; Dabbagh and Lee, 2014). Prioritization techniques are based on different underlying concepts. As a result, they produce different prioritization results. Thus, careful selection of a prioritization technique is necessary to provide an effective solution for a particular project.

The requirements prioritization techniques can be grouped into two broad categories: quantitative and qualitative techniques (Creswell, 2013).

3.1.1 Quantitative techniques

This subsection presents the state-of-the-art quantitative techniques for requirements prioritization (Leffingwell and Widrig, 2003; Mead, 2006; Berander, 2007; Wiegers and Beatty, 2013). The priority is computed based on the defined nominal scale that represents the level of importance of the requirements (Berander, 2007).

Weiger's method involves three key terms: the value of a requirement, costs, and technical risks associated with its implementation (Wiegers and Beatty, 2013). The value is calculated by the customers on a scale from 1 to 9, while costs and risks are evaluated by the developers. Finally, the priority of requirements is calculated by dividing the value by the sum of the cost and risk associated with its implementation.

Cumulative voting (CV) is a method to select the requirements by casting votes equal to the total number of requirements (Leffingwell and Widrig, 2003; Mead, 2006). However, this method is unsuitable for a large number of requirements because of possible miscalculations when summing up the distributed units among the requirements (Berander and Andrews, 2005).

Hierarchical cumulative voting (HCV) helps in solving multi-aspect decision problems by arranging them into hierarchies. Instead of prioritizing all requirements at a same time, HCV takes a subset of requirements and prioritizes them one by one in a sequential manner (Berander, 2007).

3.1.2 Qualitative techniques

Qualitative prioritization techniques compute the requirements priority based on a defined ordinal scale (Likert, 1932).

The analytical hierarchy process (AHP) is useful in solving multi-aspect decision problems (Saaty, 2008; Tahriri *et al.*, 2014). To find the relative priorities of hierarchically classified requirements, AHP performs pairwise comparisons. In AHP, there is a direct relationship between the number of requirements and the number of comparisons. Therefore, AHP takes more time to prioritize the requirements than other prioritization techniques such as cumulative voting and top-10 requirements (Berander and Andrews, 2005). According to Achimugu *et al.* (2014), AHP has been recognized as the most cited prioritization technique among all existing techniques.

The cost-value approach prioritizes the requirements using two factors, cost and value (Karlsson and Ryan, 1997; Ahl, 2005). Customers determine the requirements value, while software engineers estimate the requirements cost using AHP pairwise comparison. Next, a cost-value diagram is plotted based on AHP-based comparison results. This diagram is further used to reach a consensus between stakeholders and requirements engineers for ranking the requirements.

Numerical assignment (NA) categorizes the requirements into three classes: mandatory, desirable, and inessential (Brackett, 1990; Lehtola and Kauppinen, 2006). The mandatory requirements must be fulfilled to satisfy the customers. Desirable requirements help in improving the customer's satisfaction level, while inessential requirements can be safely ignored.

The ranking technique assigns a number (using an ordinal scale) to each software requirement based on its importance as suggested by the stakeholders (Berander and Andrews, 2005). On this scale, the most important requirement is assigned a numeral value 1, and the least important a numeral value n, where n represents the last requirement. Finally, sorting techniques (e.g., bubble sort) are used to prioritize the requirements.

The top-10 requirements technique selects the most important 10 requirements from a pool of requirements (Lauesen, 2002). It helps stakeholders and requirements engineers select the top-10 requirements based on a consensus among stakeholders. Consequently, it helps avoid disagreements among stakeholders (Berander and Andrews, 2005).

Binary search tree (BST) handles a large number of requirements as it involves $n\log n$ comparisons for n requirements (Ahl, 2005). However, the major drawback associated with the BST technique is that it does not count the overall importance of all requirements; rather, it determines the importance of a single requirement compared with another.

Planning game (PG) (Mead, 2006) is a variation of the numerical assignment technique (Brackett, 1990), which uses one of the key practices of eXtreme Programming (XP), such as user stories, to prioritize the requirements (Beck, 2000; Mohammadi *et al.*, 2008). In the PG technique, the customer divides the requirements into three main categories similar to the NA technique. Similarly, the programmer divides the requirements into those that can be estimated precisely, reasonably well, or cannot be estimated at all.

3.2 Requirements prioritization frameworks

In the literature, a number of requirements prioritization frameworks have been proposed to rank software requirements (Moisiadis, 2002; Avesani *et al.*, 2004; Liu *et al.*, 2004; Danesh *et al.*, 2009; Ramzan *et al.*, 2009; Bebensee *et al.*, 2010; Otero *et al.*, 2010; Sadiq *et al.*, 2010; Dabbagh and Lee, 2013; 2014; Perini *et al.*, 2013; Dabbagh *et al.*, 2014). This subsection briefly discusses the current frameworks that support the requirements prioritization process.

Moisiadis (2002) integrated the multi-faceted aspects and used quality function deployment (QFD) and the AHP technique for requirements prioritization purposes. Basically, this framework prioritizes the requirements based on business goals and stakeholder viewpoints. A requirements prioritization tool based on the proposed framework is presented and evaluated using some case studies.

Avesani *et al.* (2004) introduced machine learning and a Boolean metrics based framework to estimate the requirements ranking. To rank the requirements, it performs three basic steps: (1) pair sampling, (2) preference elicitation, and (3) ranking learning. The authors conducted an experimental study and reported that a high level of requirements prioritization was achieved through a low level of requirements elicitation.

Liu *et al.* (2004) integrated the multi-perspective requirements elicited from all stakeholders into a single concise set of requirements. The proposed framework first accepts stakeholders' requirements and their initial probabilities, and then determines their impact relationship using a relationship matrix. Next, it calculates the final probability of requirements based on the normalized and adjusted probabilities. The procedure is continued until all stakeholders' perspectives have been considered. The authors conducted a case study and reported that the proposed framework could be applied to a large-scale system.

Ramzan *et al.* (2009) proposed an evaluation framework that uses a proposed value based fuzzy requirements prioritization technique. In this technique, requirements are first elicited from all involved stakeholders with their assigned values. Next, an expert group (further divided into two subgroups) examines the requirements (value-based requirements prioritization) and stakeholders (value-based stakeholder's prioritization) independently. Finally, after performing a value-based fuzzy logic requirements prioritization, it lists the stakeholders' prioritized requirements.

Danesh *et al.* (2009) focused on business value oriented requirements prioritization. The basic working of the proposed framework is: (1) determining the core business values and (2) finding their relative relationship by assigning weights using a simple ordinal scale ranging from 0 to 10, where 0 means 'not important' and 10 means 'critical'. The authors validated the framework using an online-banking system as a case study, and reported that it supports 'ease of use' during the requirements prioritization process.

Sadiq *et al.* (2010) proposed a prioritization framework that uses the AHP technique and also considers the risk factor of each requirement. Finally, requirements are prioritized by comparing the associated risks with their calculated weights.

Otero *et al.* (2010) proposed a framework based on quality attribute criteria measurement and their relative importance. The derived quality measurement is used as a main metric for requirements prioritization. The framework was evaluated using a case study where only 10 requirements are considered based on some quality attributes including type, scope, customer's satisfaction, perceived impact, application-specific attributes, and penalties. The authors concluded that the proposed technique is feasible for efficiently evaluating the quality and priority of software requirements.

Bebensee *et al.* (2010) introduced the use of the binary priority list (BPL) for requirements prioritization and compared its effectiveness with that of existing prioritization techniques. The authors validated the performance of the proposed framework by conducting case studies in two small Dutch product software companies. They reported that the technique could be effective for a medium amount of requirements and is useful for small-size software product companies.

Dabbagh and Lee (2013) proposed an AHP-based approach for prioritizing non-functional requirements. In this approach, the interrelationships which may exist among non-functional requirements are considered during the prioritization process, while non-functional requirements are prioritized based on their importance to the customers and users. In other words, the approach produces a consistent prioritized list of non-functional requirements, among which there are no conflicting relationships.

The case-based ranking approach (CBRank) is a prioritization approach, which combines the pairwise comparison and machine learning techniques to calculate the final ordering of requirements (Perini *et al.*, 2013). This approach aims to overcome the scalability issues associated with pairwise comparisons. In other words, using machine learning techniques makes the approach applicable for a large number of requirements. Perini *et al.* (2009) conducted a controlled experiment to compare two tool-supported requirements prioritization approaches, AHP and CBRank. The authors focused on measuring three properties: time consumption, ease of use, and accuracy of results. They concluded that CBRank shows a better performance than AHP in terms of time consumption and ease of use, but AHP outperforms CBRank in terms of accuracy.

Dabbagh *et al.* (2014) proposed the hybrid assessment method (HAM) based approach, which could be applied in the context of prioritizing functional and non-functional requirements. This approach was inspired by HAM, first introduced by Ribeiro *et al.* (2011). HAM is a multi-criterion decision-making method, in which a pairwise comparison decision matrix is integrated with a classical weighted average decision matrix to rank a collection of alternatives with respect to a set of criteria.

Dabbagh and Lee (2014) proposed an approach for integrating the process of prioritizing functional and non-functional requirements. The integrated prioritization approach (IPA) can be defined as an approach which prioritizes functional and non-functional requirements simultaneously, producing two separate prioritized lists of functional and non-functional requirements. One of the advantages of IPA is that it considers both functional and non-functional requirements during the prioritization stage using only one decision matrix. It also establishes the relationship between functional and non-functional requirements to perform the prioritization task.

Table 1 presents a comparison among existing requirements prioritization frameworks with the aim of (1) highlighting the similarities and differences which exist among those frameworks and (2) identifying the strengths and limitations of each framework.

By extensively reviewing the literature, we have observed that current frameworks either propose new prioritization techniques or suggest improvements in the existing techniques. However, to the best of our knowledge, there is no generalized framework that may be suitable for different types of applications and provides systematic end-to-end support during the requirements prioritization process. Moreover, there is no framework that keeps historical data useful for future releases of similar types of applications (Table 1). The framework presented in this study (RePizer) differs appreciably from all other frameworks presented in the literature, since RePizer does not always employ the same requirements prioritization technique, such as AHP (Sadiq *et al.*, 2010), but can employ any suitable prioritziation technique based on the stakeholders' viewpoints. As a result, RePizer supports multi-perspective requirements prioritization by considering stakeholders' viewpoints (i.e., perceived impacts and penalties) and business goals as analyzed by the requirements engineer. Furthermore, RePizer stores historical data, which is beneficial for selecting the most suitable prioritization technique effective for prioritizing a set of requirements of future releases of similar types of a large-scale system.

4 Proposed requirements prioritization framework

This section presents our proposed framework, called RePizer, which helps development organizations prioritize requirements by integrating the multi-perspective requirements elicited from all stakeholders. RePizer stores the prioritization results of an existing release, which could ultimately improve the decision-making process of requirements engineers for future releases of a particular project. For example, suppose that requirements engineers use the AHP technique to prioritize requirements based on a certain criterion such as implementation cost for a given project. The prioritization results obtained by applying the AHP technique may not be as accurate as expected in terms of a defined prioritization criterion (e.g., implementation cost). This is where RePizer could assist the development team by using historical data (i.e., version number, number of requirements, type of requirements, previously applied prioritization technique, and type of application) about prioritization results of similar types of projects. The historical data would guide requirements engineers to choose the most appropriate prioritization technique, such as PG, to achieve more accurate results for future releases of the same project or of a different project with similar characteristics. Fig. 3 depicts the overall view of the proposed framework.

The key components of RePizer are: (1) a requirements planner and (2) a requirements prioritizer, which are discussed in the following subsections. Before going through the detailed description of

Table 1 Comparison of state-of-the-art frameworks for requirements prioritization

Approach name (reference)	Employed prioritization technique(s)	Requirements type	Retaining historical data	Risk analysis	Multi-perspective support
Multi-faceted approach (Moisiadis, 2002)	Quality function deployment and AHP	Functional	No	No	Yes
Machine learning approach (Avesani *et al.*, 2004)	Pair sampling and rank learning	Functional	No	No	No
Requirements probability approach (Liu *et al.*, 2004)	Probability analysis and impact relationship	Functional	No	No	Yes
Value-based fuzzy logic approach (Ramzan *et al.*, 2009)	Value-based fuzzy	Functional	No	No	Yes
Business value oriented approach (Danesh *et al.*, 2009)	Relative relationship between business values	Functional	No	No	No
AHP-based approach (Sadiq *et al.*, 2010)	AHP	Functional	No	Yes	No
Quality criteria based approach (Otero *et al.*, 2010)	Quality attribute measurement	Functional	No	No	No
Binary priority based approach (Bebensee *et al.*, 2010)	Binary priority list	Functional	No	No	No
AHP-based approach (Dabbagh and Lee, 2013)	AHP	Non-functional	No	No	No
Case-based ranking approach (Perini *et al.*, 2013)	Pairwise comparison and machine learning	Functional	No	No	No
HAM-based approach (Dabbagh *et al.*, 2014)	Pairwise comparison	Functional and non-functional	No	No	Yes
IPA-based approach (Dabbagh and Lee, 2014)	Integrated prioritization	Functional and non-functional	No	No	No
RePizer (current study)	Applicable using different prioritization techniques	Functional and non-functional	Yes	No	Yes

AHP: analytical hierarchy process; HAM: hybrid assessment method; IPA: integrated prioritization approach

Repizer's components, note that two main assumptions need to be considered while using the proposed framework:

Assumption 1 There must be effective communication among the group of stakeholders. This framework follows the Onsite Customers practice of XP (Mohammadi *et al.*, 2008), which is helpful in improving the required communication.

Assumption 2 All elicited requirements must possess the mentioned quality attributes as described by Wiegers and Beatty (2013).

4.1 Requirements planner

The requirements planner contains essential information, including the requirements category, prioritization criteria, and project ID, provided by the requirements engineer after negotiating with different involved stakeholders (Fig. 3). This information is necessary for correct functioning of the framework and is used by the requirements prioritizer to ultimately generate the prioritized list of requirements. Therefore, the requirements planner includes:

Requirements category (RC): Requirements can be categorized as functional requirements (FR) and non-functional requirements (NFR). The set of requirements provided by the requirements engineer can be represented as follows: RS=$\{R_1, R_2, \ldots, R_n\}$. Note that an individual requirement can be represented as R_i $(1\le i\le n)$. A given set of requirements can be further categorized into $\mathrm{RS}_{\mathrm{FR}}$ and $\mathrm{RS}_{\mathrm{NFR}}$, where $\mathrm{RS}_{\mathrm{FR}}$ represents a set of functional requirements and $\mathrm{RS}_{\mathrm{NFR}}$ a set of non-functional requirements.

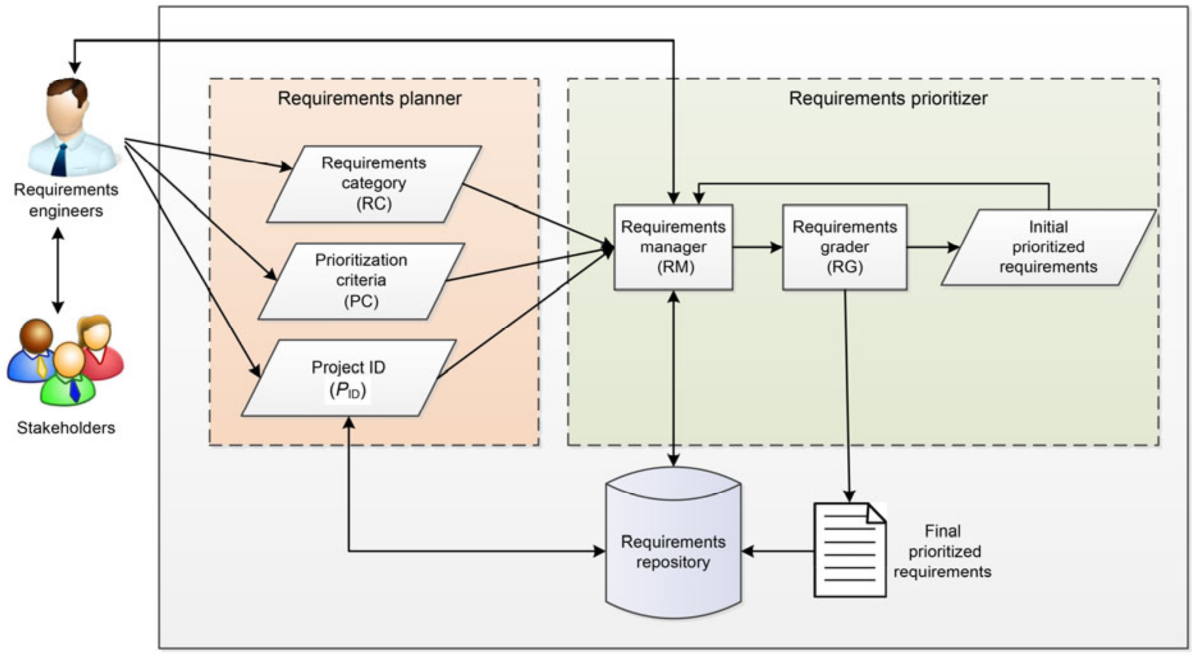

Fig. 3 High-level view of RePizer

Prioritization criteria (PC): One of the main elements of RePizer is requirements prioritization criteria (PC). Requirements can be prioritized based on different criteria such as implementation cost, value to users, risk, and time. Based on PC and the selected prioritization technique, RePizer can effectively prioritize the requirements depending on the requirements category (i.e., FR or NFR). A careful selection of the PC plays a significant role in meeting the defined project objectives, which can be easily obtained from all involved stakeholders.

Project ID (P_{ID}): The requirements planner contains a unique P_{ID} for each new project. Each P_{ID} is assigned by the requirements engineer and is compared with existing P_{ID}'s from the requirements repository (RR). P_{ID} can be formally represented as

$$\begin{aligned} P_{ID} = [\{P_x \mapsto (P_{new} \vee P_{old})\} \\ \wedge \{(C_1 : P_x \in \mathrm{RR} \oplus C_2 : P_x \notin \mathrm{RR})\}], \end{aligned} \tag{1}$$

where P_x represents the initial project ID, P_{new} the newly assigned project ID, and P_{old} the existing project ID. There are two conditions C_1 and C_2 that need to be met while allocating a project ID, as described in Eq. (1). Note that only one condition may hold true while assigning a project ID (exclusive OR: ⊕). If C_1 holds (i.e., P_x already exists in RR), then the requirements engineer has to assign P_{old} as a P_{ID}; otherwise, P_{new} will be assigned to a current project.

In a case where C_1 is true, the current project is a new release, whose historical data are stored in RR. Consequently, the requirements engineer can check historical data (e.g., previously applied prioritization techniques) to select the prioritization technique most suitable for the current release. However, if C_2 holds true, it means that no historical data of the current project are retained in RR. In this situation, the requirements engineer might search within RR for a different project with similar characteristics to select an appropriate prioritization technique for the current project.

4.2 Requirements prioritizer

The requirements prioritizer accepts inputs from the requirements planner (Fig. 3). Next, it applies existing prioritization techniques to generate a prioritized list of requirements. It consists of two main components: requirements manager and requirements grader.

Requirements manager (RM): RM helps regulate the activities of the requirements prioritizer component. First, it accepts inputs (i.e., P_{ID}, RC, and PC) from the requirements planner. Then, it checks whether it is a first or subsequent (new) release of a given project. In the case of a first release, the requirements engineer needs to apply a suitable prioritization technique for that particular project. However, in a case where historical data of a project are available for a new release of a given project, the requirements engineer extracts those data from the requirements repository (RR) through RM. Using the extracted data, the requirements engineer decides which prioritization technique is the most suitable (e.g., AHP) for the current release of a given project. After that, it transfers this information including the selected prioritization technique to the requirements grader (RG). RG produces an initial prioritized list of requirements based on the selected prioritization technique and defined criteria, and returns this information to RM. Next, RM forwards initial prioritized requirements to the requirements engineer for improving the decision-making process. The requirements engineer analyzes the prioritization results and may select another prioritization technique or confirm the produced results. Finally, the whole prioritization process is terminated once the requirements engineer is satisfied.

Requirements grader (RG): RG receives collective information (i.e., information taken from the requirements engineer and requirements planner) from RM. This set of combined information acts as a function, which is applied to the set of requirements to prioritize them. RG then generates a prioritized list that it sends back to the RM, which afterwards sends this list to the requirements engineer for further analysis. Ultimately, RG generates the final prioritized list of requirements. All this information is also stored in RR for future reference.

5 Case study

The purpose of this section is to demonstrate how the proposed framework can be applied in real cases. To show the applicability of the proposed framework, we used the Library of Congress SRS document (LCPAIG, 2003) as a case study. This document describes an application, called the OpenURL resolver, which extracts metadata from users' requests. For example, a library researcher submits a request for a journal article that is viewed by the OpenURL resolver as metadata tags, such as author names, article title, volume number, and page numbers. Based on this metadata, the OpenURL resolver further fulfills the request by consulting the related application and verifying the user's access rights. This case study contains a set of requirements which are divided into six different categories (Table 2).

Table 2 Set of requirements (category)[*]

Category	Requirements
1	General requirements
2	Knowledge database requirements
3	Services menu requirements
4	Help facilities for end users
5	Documentation for administrators
6	Administration and vendor support

[*] From LCPAIG (2003)

In this particular case study, each category was divided into sub-categories tagged as mandatory (M) or desired (D) (Table 3). For instance, category 1 contained only one sub-category (1.1), which consisted of 9 mandatory and 1 desired requirements.

Generally, it is a straightforward task for requirements engineers to prioritize a small number of requirements. However, the Library of Congress study contains a large list of FRs (i.e., categories) at the first level. These FRs are further divided into sub-categories (i.e., mandatory or desired) at the second and third levels. This significantly increased the complexity of the selected study. In such cases, RePizer can be beneficial for prioritizing the requirements.

In this case study, we used planning game (PG) as a prioritization technique to rank the requirements. In practice, we used user stories and then performed two-level prioritization based on the variables (i.e., mandatory and desired). The first step was that the requirements engineer assigned a unique P_{ID} to the current project, for example, P_{01}. Suppose the condition C_2 (Eq. (1)) was true; i.e., the given project ID was not in RR. Then, a unique P_{ID} was assigned to this project and stored in RR. The static values extracted from the case study were: $\mathrm{RS}_{\mathrm{FR}}$=134 (total), 90 mandatory requirements, and 44 desired. To create user stories and levels of prioritization, the participants (totally 32; female 11, male 21) were asked to rank the set of requirements. The participants were Master's students who had studied introductory and advanced-level software engineering (SE) and had the experience of developing SE-related projects.

Table 3 Set of requirements (category and sub-category)*

Category	Sub-category	Mandatory	Desired
1	1.1	9	1
2	2.1	13	12
	2.2	4	1
	2.3	7	1
	2.4	3	–
	2.5	6	–
	2.6	2	1
3	3.1	3	2
	3.2	7	4
	3.3	2	–
4	4.1	1	1
5	5.1	8	–
	5.2	1	–
6	6.1	6	–
	6.2	6	12
	6.3	5	5
	6.4	3	1
	6.5	11	2
	6.6	3	1

* From LCPAIG (2003)

We have created a template in a tabular form to display the requirements of this case study. Participants were asked first to rank the categories and then the sub-categories. After that they were asked to rank the mandatory and desired requirements of a particular sub-category using three legends: *A* (very important), *B* (important), and *C* (least important). A lot of comparisons were needed to prioritize the requirements and thus a sufficient amount of time was essential to produce valid results. The total time given to each participant was 10 d. Table 4 depicts the final prioritized list for categories of the Library of Congress study.

The results indicate that 11 out of 32 participants preferred category 1 to be at position 1 (on the top of the prioritized list); 13 participants preferred category 2 to be at position 2; 13 participants preferred category 3 to be at position 5; 10 participants preferred category 4 to be at position 3; 11 participants preferred category 5 to be at position 6. Finally, 4 participants preferred category 6 to be at position 4. All the results were then submitted to RePizer, which took the maximum number of votes for each category and assigned a priority to each category (Table 4).

For example, for position number 6, 10 participants voted for category 4, and 11 voted for category 5. As more participants (11 participants) voted for the 6th position, category 5 was placed at position number 6. The same was applied for sub-categories, and also mandatory and desired requirements. Table 5 describes the compiled results for prioritizing the list of categories and sub-categories, respectively.

Finally, Table 6 presents the complete prioritized list of Library of Congress requirements at category and sub-category levels, as generated by RePizer.

Table 4 Prioritized list of categories

Category	Prioritized list of category
1	1
2	2
3	4
4	6
5	3
6	5

Table 5 Prioritized list of sub-categories

Prioritized list of category	Sub-category	Prioritized list of sub-category
1	1.1	1.1
2	2.1	2.1
	2.2	2.6
	2.3	2.3
	2.4	2.5
	2.5	2.4
	2.6	2.2
4	4.1	4.1
6	6.1	6.1
	6.2	6.2
	6.3	6.4
	6.4	6.5
	6.5	6.3
	6.6	6.6
3	3.1	3.1
	3.2	3.2
	3.3	3.3
5	5.1	5.2
	5.2	5.1

6 Evaluation

In this section, we illustrate in detail the experiment which we conducted to evaluate the performance of RePizer. Table 7 summarizes the key components of the experiment in terms of its main goal, independent variables, dependent variables, and context.

Table 6 Final prioritized list of requirements

Category	Sub-category	Mandatory	Desired
1	1.1	1.1.1, 1.1.4, 1.1.5, 1.1.8, 1.1.3, 1.1.6, 1.1.2, 1.1.7, 1.1.9	1.1.10
2	2.1	2.1.1, 2.1.2, 2.1.3, 2.1.4, 2.1.8, 2.1.9, 2.1.10, 2.1.13, 2.1.5, 2.1.6, 2.1.7, 2.1.11, 2.1.12	2.1.14, 2.1.18, 2.1.19, 2.1.25, 2.1.15, 2.1.16, 2.1.17, 2.1.22, 2.1.24, 2.1.20, 2.1.21, 2.1.23
	2.6	2.6.1, 2.6.2	2.6.3
	2.3	2.3.1, 2.3.2, 2.3.3, 2.3.4, 2.3.6, 2.3.5, 2.3.7	2.3.8
	2.5	2.5.1, 2.5.6, 2.5.2, 2.5.3, 2.5.4, 2.5.5	
	2.4	2.4.1, 2.4.2, 2.4.3	
	2.2	2.2.1, 2.2.2, 2.2.4, 2.2.3	2.2.5
4	4.1	4.1.1	4.1.2
6	6.1	6.1.1, 6.1.2, 6.1.5, 6.1.6, 6.1.3, 6.1.4	
	6.2	6.2.1, 6.2.2, 6.2.4, 6.2.3, 6.2.6, 6.2.5	6.2.7, 6.2.8, 6.2.9, 6.2.12, 6.2.13.4, 6.2.10, 6.2.11, 6.2.15, 6.2.17, 6.2.16, 6.2.18, 6.2.14
	6.4	6.4.1, 6.4.3, 6.4.2	6.4.4
	6.5	6.5.1, 6.5.3, 6.5.4, 6.5.6, 6.5.7, 6.5.9, 6.5.2, 6.5.8, 6.5.10, 6.5.5, 6.5.11	6.5.13 6.5.12
	6.3	6.3.1, 6.3.5, 6.3.2, 6.3.3, 6.3.4	6.3.6, 6.3.7, 6.3.8, 6.3.10, 6.3.9
	6.6	6.6.1, 6.6.2, 6.6.3	6.6.4
3	3.1	3.1.1, 3.1.2, 3.1.3	3.1.4, 3.1.5
	3.2	3.2.1, 3.2.3, 3.2.4, 3.2.5, 3.2.6, 3.2.2, 3.2.7	3.2.8, 3.2.9, 3.2.10, 3.2.11
	3.3	3.3.1, 3.3.2	
5	5.2	5.2.1	
	5.1	5.1.1, 5.1.2, 5.1.3, 5.1.5, 5.1.7, 5.1.8, 5.1.6, 5.1.4	

Table 7 Overview of the evaluation of RePizer

Criteria	Description
Goal	Evaluate the performance of RePizer while adopting PG and AHP techniques
Independent variable	Prioritization techniques: PG and AHP
Dependent variable	Accuracy of results; ease of use
Context	Experiment executed using 32 real subjects prioritizing 134 requirements of the OpernURL resolver project

The main goal of the experiment was to investigate the performance of the proposed framework (i.e., RePizer) while adopting each of two different prioritization techniques, PG and AHP. In practice, to evaluate the performance of RePizer, two properties were measured: (1) accuracy of the results produced by RePizer while using either the PG or AHP technique, and (2) ease of use of RePizer while exploiting each technique. These two properties are further discussed in Section 6.2.2.

The purpose of the experiment was to answer the following research questions:

RQ1: Which technique, PG or AHP, produces more accurate results when adopted by RePizer?

RQ2: Which technique, PG or AHP, makes RePizer easier to use?

The ultimate goal was to collect evidence to show that by adopting one or the other technique, the performance of RePizer could be improved.

6.1 Hypotheses

Based on the above research questions, we formulated the following null and alternative hypotheses:

Null hypothesis ($H_{0\,\mathrm{accuracy}}$): The accuracy of results is equal while adopting PG or AHP.

Alternative hypothesis ($H_{1\,\mathrm{accuracy}}$): The accuracy of results is not equal while adopting PG or AHP.

Null hypothesis ($H_{0\,\mathrm{ease\,of\,use}}$): There is no significant difference between PG and AHP in terms of ease of use when adopted by RePizer.

Alternative hypothesis ($H_{1\,\mathrm{ease\,of\,use}}$): There is a significant difference between PG and AHP in terms of ease of use when adopted by RePizer.

6.2 Variables

Independent and dependent variables of the experiment were identified as the following.

6.2.1 Independent variables

The independent variables of the experiment were the PG and AHP technique. These techniques were introduced in Section 3.

6.2.2 Dependent variables

To perform an effective evaluation of RePizer while adopting either the PG or AHP technique, two dependent variables were measured in the experiment: the accuracy of the results produced by RePizer and the ease of use.

We measured the accuracy of results as the first dependent variable in terms of expected accuracy. The expected accuracy was measured through a post-questionnaire (post-test *A*), in which each test subject was asked to answer the following question immediately after working with each prioritization technique, once he/she was provided with prioritized lists of requirements produced by the technique based on his/her judgment: How accurate did you find the results produced by the technique while using RePizer? In the experiment, the term ‘technique’ was replaced with PG or AHP. The test subject was asked to choose an integer ranging from 1 (very low) to 5 (very high) according to the Likert scale (Likert, 1932). Measuring the accuracy of results could be useful for answering the first research question (RQ1).

‘Ease of use’ represents how easily a decision maker performs the prioritization process using a given prioritization technique. In the experiment, the second dependent variable (ease of use) was measured by means of a post-questionnaire (post-test *B*). Immediately after working with each prioritization technique, the test subjects carried out the first post-test *B* by answering the following question: How easy was it to perform the actual prioritization using the technique while using RePizer? In the experiment, the term ‘technique’ was replaced with PG or AHP. The test subject was asked to choose an integer ranging from 1 to 5 according to the Likert scale (Likert, 1932), where 1 indicated very difficult and 5 represented very easy. Measuring this property could help us investigate the second research question (RQ2).

6.3 Subjects

The experiment was performed with 32 real participants (21 male and 11 female Master’s students) who had studied introductory and advanced-level software engineering (SE) and had experience in developing SE-related projects. They have also participated in SCORE Contest 2011 (http://score-contest.org/2011/).

6.4 Object

The object of the experiment was a collection of 134 requirements of the OpenURL resolver project (LCPAIG, 2003). The OpenURL resolver is a web-based application that extracts metadata from users’ requests. For example, a library researcher submits a request for a journal article that is viewed by the OpenURL resolver as metadata tags (such as author names, article title, volume number, and page numbers).

Based on the metadata, the OpenURL resolver fulfills the request by consulting the related application and verifying the user’s access rights. The requirements of the OpenURL resolver are given in Table 2.

6.5 Experiment design

We adopted a type of paired comparison design (Table 8), comprising one factor with two treatments (Wohlin *et al.*, 2012). In this design, each subject separately applied PG and AHP techniques to the same set of requirements (i.e., the object) while using RePizer. The order of executions was given at random to each subject to minimize the effect of execution order on the final results.

Table 8 The paired comparison design used for the experiment

Group	Prioritization task 1	Prioritization task 2
1	RePizer using PG	RePizer using AHP
2	RePizer using AHP	RePizer using PG

PG: planning game; AHP: analytical hierarchy process

6.6 Experiment results

This subsection presents the significant results achieved from the experiment. We initially performed descriptive analysis using Microsoft Excel. We then carried out statistical analysis using IBM SPSS Statistics Version 21 to reject or accept the null hypotheses formulated in Section 6.1. A 5% significance level was used for hypothesis testing.

6.6.1 RQ1: Which technique, PG or AHP, produces more accurate results when adopted by RePizer?

Table 9 summarizes the results collected from post-test *A*, which indicates the opinions of test subjects with respect to the expected accuracy of PG and AHP. The test subjects seemed to have different opinions of the two techniques.

Table 9 Results of measuring expected accuracy collected from post-test *A* at different Likert scales

Prioritization technique	Measured expected accuracy				
	1	2	3	4	5
PG	–	5	6	16	5
AHP	–	6	17	9	–

PG: planning game; AHP: analytical hierarchy process

Therefore, to understand which technique produces more accurate results, the third null hypothesis was tested statistically. Before starting to test the null hypothesis ($H_{0\,\text{accuracy}}$) (Section 6.1), we checked the distribution of data using the Shapiro-Wilk test to determine whether it was normal. According to the test (Table 10), we found that the distribution of data was not normal with respect to expected accuracy as the *P*-value was lower than 0.05 for both PG and AHP techniques. Due to the nature of the variables and the fact that the data were not normally distributed, we decided to investigate the first null hypothesis using a non-parametric test, the Mann-Whitney test (Siegel and Castellan, 1988). In this case, we observed that the difference between the two techniques with respect to the expected accuracy was statically significant since the *P*-value was 0.007 (<0.05). Therefore, the first null hypothesis was rejected and we concluded that PG produces more accurate results than AHP when adopted by RePizer.

6.6.2 RQ2: Which technique, PG or AHP, makes RePizer easier to use?

We measured the ease of use through post-test *B* (discussed in Section 6.2.2). The results of this post questionnaire are given in Table 11. Most subjects believed that it was easier to use PG for performing the prioritization.

Due to the non-normal distribution of data (Table 10), we applied the non-parametric Mann-Whitney test to investigate the second null hypothesis ($H_{0\,\text{ease of use}}$). The results showed that the difference between the ease of use of PG and AHP was significant ($P<0.05$). Thus, the second null hypothesis was rejected and we concluded that PG is easier to use than AHP when adopted by RePizer.

Table 11 Results of measuring ease of use collected from post-test *B* at different Likert scales

Prioritization technique	Measured ease of use				
	1	2	3	4	5
PG	–	7	6	15	4
AHP	–	13	13	6	–

PG: planning game; AHP: analytical hierarchy process

6.7 Threats to validity

This subsection discusses the potential threats which could bias the validity of the experiment results.

Since the experiment results (i.e., expected accuracy and ease of use) were obtained based on subjective opinions of the participants, it is possible that not all of the participants interpreted the questions asked in post-tests in the same way. To mitigate this threat, we designed easy-to-learn questionnaires using standard scales.

The experiment results are specific and dependent on the context in which the experiments were carried out. Therefore, it is unwise to generalize based on the results of this study. To minimize this threat, more experiments should be conducted in diverse contexts with different participants.

We used statistical analysis (i.e., non-parametric tests) to test hypotheses. Each statistical test might have some degree of tolerance, which could bias the results. To alleviate this threat, we used an automated

Table 10 Normality of experimental data tested using the Shapiro-Wilk test

Prioritization technique	Expected accuracy			Ease of use		
	Statics	*df*	Sig.	Statics	*df*	Sig.
PG	0.874	32	0.000359	0.846	32	0.000342
AHP	0.803	32	0.000045	0.793	32	0.000030

PG: planning game; AHP: analytical hierarchy process statistical tool, SPSS, which provides more reliable results than calculating statistical values manually. It is highly probable that human errors occur when performing complex statistical calculations manually.

In the experiment, the subjects were students, not professionals; hence, their reasoning and interpretation of requirements might not be representative of specialists in software product companies. However, the students had a thorough knowledge of the software product and its requirements, and therefore reasoning and interpretation threats were unlikely to have had a significant effect.

During the experiment, there was a risk that when switching techniques, the students may have been influenced by familiarity with the requirements or by learning from their experience with the first prioritization technique. We tried to minimize this threat by counterbalancing the groups, but further investigations with different systems are needed to confirm the results obtained in this study. Furthermore, the participants of the experiment could be influenced by fatigue. We limited this threat by limiting the number and complexity of the requirements. Moreover, we tried to arrange the time of each session to suit each participant so that he/she could be fresh at that time.

6.8 Discussion

The main results from this study are summarized in Table 12 in terms of hypothesis, dependent variable, statistical test, *P*-value, result, and direction. Based on the results, RePizer shows a better performance with respect to accuracy of results perceived by test subjects and ease of use when PG rather than AHP is used, since the first and second null hypotheses were rejected (Table 12).

Note that these results have been achieved in a situation where the PG and AHP techniques were applied separately by the same set of test subjects to the same set of requirements. Thus, when accuracy is an important issue in prioritization, we recommend that RePizer should be used in conjunction with PG. Moreover, a requirements engineer is likely to find it much easier to apply RePizer using PG. We believe that our approach of analyzing the particular outcomes in terms of accuracy of results and ease of use could be used in pilot studies for identifying trends before conducting a large-scale study in industry. The analysis provided valuable information to choose the most suitable prioritization technique when applying RePizer to a given software project in an organization.

7 Conclusions and future work

Requirements prioritization is an important activity performed in the early stages of a software development process. The involvement of multiple stakeholders often creates a set of conflicting requirements which cannot all be implemented. In this paper, we proposed a framework, called RePizer, which enables practitioners to prioritize requirements based on defined criteria using a prioritization technique. The main contribution of this work is that RePizer retains the prioritization results of an existing release, which could ultimately improve the decision-making process of requirements engineers in developing future releases of a given project. To show the applicability of the proposed framework, we applied it to a set of 134 requirements of the Library of Congress case study. We further conducted an empirical study to evaluate the performance of RePizer while adopting either the PG or AHP technique. The evaluation was based on measuring two properties: expected accuracy and ease of use. Statistical analysis of the results indicated better performance of RePizer when used in conjunction with the PG technique.

It would be of interest to conduct further experiments using different applications and other prioritization techniques to compare the results with the outcomes of this study. Also, it would be useful to develop a modified version of RePizer that includes a requirements knowledge base component to automate the selection process of the prioritization technique.

Table 12 Summary of hypothesis testing

Hypothesis	Dependent variable	Statistical test	*P*-value	Result	Direction
$H_{0\,\text{accuracy}}$	Expected accuracy	Mann-Whitney	0.007	Rejected	PG
$H_{0\,\text{ease of use}}$	Ease of use	Mann-Whitney	0.003	Rejected	PG

PG: planning game

References

Achimugu, P., Selamat, A., Ibrahim, R., *et al.*, 2014. A systematic literature review of software requirements prioritization research. *Inform. Softw. Technol.*, **56**(6):568-585. http://dx.doi.org/10.1016/j.infsof.2014.02.001

Ahl, V., 2005. An Experimental Comparison of Five Prioritization Methods—Investigating Ease of Use, Accuracy and Scalability. MS Thesis, Blekinge Institute of Technology, Ronneby, Sweden.

Arias, T.B.C., America, P., Avgeriou, P., 2011. Defining and documenting execution viewpoints for a large and complex software-intensive system. *J. Syst. Softw.*, **84**(9): 1447-1461. http://dx.doi.org/10.1016/j.jss.2010.11.908

Avesani, P., Bazzanella, C., Perini, A., *et al.*, 2004. Supporting the requirements prioritization process: a machine learning approach. Proc. 16th Int. Conf. on Software Engineering and Knowledge Engineering, p.306-311.

Bebensee, T., van de Weerd, I., Brinkkemper, S., 2010. Binary priority list for prioritizing software requirements. *LNCS*, **6182**:67-78. http://dx.doi.org/10.1007/978-3-642-14192-8_8

Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Berander, P., 2007. Evolving Prioritization for Software Product Management. PhD Thesis, Blekinge Institute of Technology, Sweden.

Berander, P., Andrews, A., 2005. Requirements prioritization. *In*: Aurum, A., Wohlin, C. (Eds.), Engineering and Managing Software Requirements. Springer, p.69-94. http://dx.doi.org/10.1007/3-540-28244-0_4

Berander, P., Jönsson, P., 2006. Hierarchical cumulative voting (HCV)—prioritization of requirements in hierarchies. *Int. J. Softw. Eng. Knowl. Eng.*, **16**(6):819-849. http://dx.doi.org/10.1142/S0218194006003026

Bourque, P., Fairley, R.E., 2014. Guide to the Software Engineering Body of Knowledge (SWEBOK$^{®}$), Version 3.0. IEEE Computer Society Press, Piscataway, New Jersey.

Brackett, J.W., 1990. Software Requirements. Technical Report, No. SEI-CM-19-1.2. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

Creswell, J.W., 2013. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches (4th Ed.). Sage Publications.

Dabbagh, M., Lee, S.P., 2013. A consistent approach for prioritizing system quality attributes. Proc. 14th ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, p.317-322. http://dx.doi.org/10.1109/SNPD.2013.9

Dabbagh, M., Lee, S.P., 2014. An approach for integrating the prioritization of functional and nonfunctional requirements. *Sci. World J.*, Article ID 737626. http://dx.doi.org/10.1155/2014/737626

Dabbagh, M., Lee, S.P., 2015. An approach for prioritizing NFRs according to their relationship with FRs. *Lect. Notes Softw. Eng.*, **3**(1):1-5. http://dx.doi.org/10.7763/LNSE.2015.V3.154

Dabbagh, M., Lee, S.P., Parizi, R.M., 2014. Application of hybrid assessment method for priority assessment of functional and non-functional requirements. Proc. Int. Conf. on Information Science and Applications, p.1-4. http://dx.doi.org/10.1109/ICISA.2014.6847365

Danesh, A.S., Mortazavi, S.M., Danesh, S.Y.S., 2009. Requirements prioritization in on-line banking systems using value-oriented framework. Int. Conf. on Computer Technology and Development, p.158-161. http://dx.doi.org/10.1109/ICCTD.2009.41

Daneva, M., Damian, D., Marchetto, A., *et al.*, 2014. Empirical research methodologies and studies in requirements engineering: how far did we come? *J. Syst. Softw.*, **95**:1-9. http://dx.doi.org/10.1016/j.jss.2014.06.035

Dominguez, J., 2009. The Curious Case of the Chaos Report 2009. Available from http://www.projectsmart.co.uk/the-curious-case-of-the-chaos-report-2009.html

Firesmith, D., 2004. Prioritizing requirements. *J. Obj. Technol.*, **3**(8):35-47. http://dx.doi.org/10.5381/jot.2004.3.8.c4

Gorschek, T., 2006. Requirements Engineering Supporting Technical Product Management. PhD Thesis, Blekinge Institute of Technology, Sweden.

Karlsson, J., Ryan, K., 1997. A cost-value approach for prioritizing requirements. *IEEE Softw.*, **14**(5):67-74. http://dx.doi.org/10.1109/52.605933

Khan, S.U.R., Rehman, I.U., Malik, S.U.R., 2009. The impact of test case reduction and prioritization on software testing effectiveness. Int. Conf. on Emerging Technologies, p.416-421. http://dx.doi.org/10.1109/ICET.2009.5353136

Lauesen, S., 2002. Software Requirements: Styles and Techniques. Addison-Wesley Professional.

Leffingwell, D., Widrig, D., 2003. Managing Software Requirements: a Unified Approach. Addison-Wesley.

Lehtola, L., 2006. Providing Value by Prioritizing Requirements Throughout Product Development: State of Practice and Suitability of Prioritization Methods. PhD Thesis, Helsinki University of Technology, Finland.

Lehtola, L., Kauppinen, M., 2006. Suitability of requirements prioritization methods for market-driven software product development. *Softw. Process Improv. Pract.*, **11**(1):7-19. http://dx.doi.org/10.1002/spip.249

Library of Congress Portals Applications Interest Group (LCPAIG), 2003. Functional Requirements for an OpenURL Resolver for the Library of Congress. Available from http://www.loc.gov/catdir/lcpaig/openurl_requirements_20031104.pdf

Likert, R., 1932. A technique for the measurement of attitudes. *Arch. Psychol.*, **22**:1-55.

Liu, X.Q., Veera, C.S., Sun, Y., *et al.*, 2004. Priority assessment of software requirements from multiple perspectives. 28th Annual Int. Computer Software and Applications Conf., p.410-415. http://dx.doi.org/10.1109/CMPSAC.2004.1342872

Mead, N., 2006. Requirements Prioritization Introduction. Software Engineering Institute Web Publication, Carnegie

Mellon University, Pittsburgh, USA.

Mohammadi, S., Nikkhahan, B., Sohrabi, S., 2008. An analytical survey of "on-site customer" practice in extreme programming. Int. Symp. on Computer Science and Its Applications, p.1-6. http://dx.doi.org/10.1109/CSA.2008.72

Moisiadis, F., 2002. The fundamentals of prioritising requirements. Proc. Systems Engineering, Test and Evaluation Conf., p.109-119.

Otero, C.E., Dell, E., Qureshi, A., *et al.*, 2010. A quality-based requirement prioritization framework using binary inputs. 4th Asia Int. Conf. on Mathematical/Analytical Modelling and Computer Simulation, p.187-192. http://dx.doi.org/10.1109/AMS.2010.48

Perini, A., Ricca, F., Susi, A., 2009. Tool-supported requirements prioritization: comparing the AHP and CBRank methods. *Inform. Softw. Technol.*, **51**(6):1021-1032. http://dx.doi.org/10.1016/j.infsof.2008.12.001

Perini, A., Susi, A., Avesani, P., 2013. A machine learning approach to software requirements prioritization. *IEEE Trans. Softw. Eng.*, **39**(4):445-461. http://dx.doi.org/10.1109/TSE.2012.52

Ramzan, M., Jaffar, M.A., Iqbal, M.A., *et al.*, 2009. Value based fuzzy requirement prioritization and its evaluation framework. Proc. 4th IEEE Int. Conf. on Innovative Computing, Information and Control, p.1464-1468. http://dx.doi.org/10.1109/ICICIC.2009.375

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

Ribeiro, R.A., Moreira, A.M., van den Broek, P., *et al.*, 2011. Hybrid assessment method for software engineering decisions. *Dec. Supp. Syst.*, **51**(1):208-219. http://dx.doi.org/10.1016/j.dss.2010.12.009

Saaty, T.L., 2008. Decision making with the analytic hierarchy process. *Int. J. Serv. Sci.*, **1**(1):83-98. http://dx.doi.org/10.1504/IJSSCI.2008.017590

Sadiq, M., Shahid, M., Ahmad, S., 2010. Adding threat during software requirements elicitation and prioritization. *Int. J. Comput. Appl.*, **1**(9):50-54. http://dx.doi.org/10.5120/200-339

Siegel, S., Castellan, N.J., 1988. Nonparametric Statistics for the Behavioral Sciences (2nd Ed.). McGraw-Hill.

Sommerville, I., 2010. Software Engineering (9th Ed.). Pearson.

Tahriri, F., Dabbagh, M., Ale Ebrahim, N., 2014. Supplier assessment and selection using fuzzy analytic hierarchy process in a steel manufacturing company. *J. Sci. Res. Rep.*, **3**(10):1319-1338.

Wiegers, K., Beatty, J., 2013. Software Requirements (3rd Ed.). Microsoft Press.

Wohlin, C., Runeson, P., Höst, M., *et al.*, 2012. Experimentation in Software Engineering. Springer Science & Business Media. http://dx.doi.org/10.1007/978-3-642-29044-2

Young, R.R., 2004. The Requirements Engineering Handbook. Artech House, Norwood, MA, USA.