RePizer: a framework for prioritization of software requirements

Saif Ur Rehman KHAN1, Sai Peck LEE1, Mohammad DABBAGH1, Muhammad TAHIR2,
Muzafar KHAN3, Muhammad ARIF1,4

1 (Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia)
2 (Faculty of Computing and Information Technology, University of Jeddah, Jeddah 21589, Saudi Arabia)
3 (College of Computer and Information Sciences (Muzahimiyah branch), King Saud University, Riyadh, Saudi Arabia)
4 (Department of Computer Science, Gabriel College Mandi Bahauddin 50400, Pakistan)

E-mail: saif_rehman@siswa.um.edu.my

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

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 the ability to select 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 making 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.

doi:10.1631/FITTEE.1500162 Document code: A CLC number:

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 (Domínguez, 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 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 prioritization 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 requirement 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 ∈ N and k < m < n). Subsequently, the requirement 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 requirement 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 several blocks or into a single block to determine high-value requirements, while considering resource constraints.

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 “Library of Congress” (OpenURL, 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:
- Providing a formal definition of the requirements prioritization problem.
- Comparing current state-of-the-art prioritization frameworks.
- Proposing a requirements prioritization framework by formally defining its components.
- Evaluating the performance of the proposed framework while adopting two different prioritization techniques by employing a real case study.

**Fig. 1 Planning of software requirements for multiple releases**

**Fig. 2 Planning of software requirements for a single release**
The rest of the paper is structured as follows: Section 2 provides background of the requirements prioritization process. Section 3 gives an overview of current state-of-the-art requirements prioritization techniques and frameworks. Section 4 presents the proposed requirements prioritization framework, while Section 5 discusses a case study to demonstrate the applicability of the proposed framework. Section 6 presents a detailed evaluation of the proposed framework while adopting each of two different prioritization techniques. Section 7 concludes the paper and suggests possible future research directions.

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 (Ahl, 2005; Lauesen, 2002) 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. By 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 \( R \) is \( PR \)

A function \( f \) from \( PR \) to the real numbers

Problem: Find \( R' \in PR \) such that

\[
(\forall R'')(R'' \in PR)(R'' \neq R')
\[
[f(R') \geq f(R'')]
\]

where \( PR \) 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 Jónsson, 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 Current state-of-the-art requirements prioritization techniques and frameworks

3.1 Requirements prioritization techniques

Researchers have proposed a number of requirements prioritization techniques (Wiegers and Beatty, 2013; Leffingwell and Widrig, 2000; Mead, 2006; Berander, 2007; Saaty, 2008; Dabbagh and Lee, 2014; Ahl, 2005; Karlsson and Ryan, 1997; Berander and Andrews, 2005; Lauesen, 2002). 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 section presents the state-of-the-art quantitative techniques for requirements prioritization (Wiegers and Beatty, 2013; Leffingwell and Widrig, 2000; Mead, 2006; Berander, 2007). 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, 2000; 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).

- **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-ten requirements (Berander and Andrews, 2005). According to Achimugu et al., (2014), AHP has been recognized as the most cited prioritization technique among all the existing techniques.

- **Cost-Value approach** prioritizes the requirements using two factors, cost and value (Ahl, 2005; Karlsson and Ryan, 1997). 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.

- **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.

- **Top-Ten Requirements** technique selects the most important ten requirements from a pool of requirements (Lauesen, 2002). It helps stakeholders and requirements engineers to select the top-ten 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 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 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 PG technique, the customer divides the requirements into three
main categories similar to the NA technique. Similarly, the programmer also divides the requirements into those which 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; Ramzan et al., 2009; Danesh et al., 2009; Sadiq et al., 2010; Otero et al., 2010; Bebensee et al., 2010; Dabbagh and Lee, 2013; Perini et al., 2013; Dabbagh et al., 2014; Dabbagh and Lee, 2014). This section 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 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: (i) pair sampling, (ii) preference elicitation, and (iii) 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 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 fuzzy based project 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: (i) determining the core business values and (ii) 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 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 that considered only ten requirements 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 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, Lee, and Parizi (2014) proposed the HAM-based approach, which could be applied in the context of prioritizing functional and non-functional requirements. This approach was inspired by the Hybrid Assessment Method (HAM), first introduced by Ribeiro et al., (2011). HAM is a multiple criteria 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 both 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 (i) highlighting the similarities and differences which exist among those frameworks; (ii) 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, as highlighted in 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 prioritization 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 to 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 applied AHP technique to prioritize requirements based on a certain criterion such as implementation cost for a given project. The prioritization results obtained by applying 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,

<table>
<thead>
<tr>
<th>Approach name (Ref.)</th>
<th>Employed prioritization technique(s)</th>
<th>Requirements type</th>
<th>Retaining historical data</th>
<th>Risk analysis</th>
<th>Multi-perspective support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-faceted approach (Moisidias, 2002)</td>
<td>Quality function deployment and analytical hierarchy process (AHP)</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Machine learning approach (Avesani et al., 2004)</td>
<td>Pair sampling and rank learning</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Requirements probability approach (Liu et al., 2004)</td>
<td>Probability analysis and impact relationship</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Fuzzy-based approach (Ramzan et al., 2009)</td>
<td>Value-based fuzzy</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Business value-oriented approach (Danesh et al., 2009)</td>
<td>Relative relationship between business values</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AHP-based approach (Sadiq et al., 2010)</td>
<td>AHP</td>
<td>Functional</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Quality criteria-based approach (Otero et al., 2010)</td>
<td>Quality attribute measurement</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Binary priority-based approach (Bebensee et al., 2010)</td>
<td>Binary priority list</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AHP-based approach (Dabbagh and Lee, 2013)</td>
<td>AHP</td>
<td>Non-functional</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Case-based ranking approach (Perini et al., 2013)</td>
<td>Pairwise comparison and machine learning</td>
<td>Functional</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HAM-based approach (Dabbagh et al., 2014)</td>
<td>Pairwise comparison</td>
<td>Functional and non-functional</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IPA-based approach (Dabbagh and Lee, 2014)</td>
<td>Integrated prioritization</td>
<td>Functional and non-functional</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RePizer (current study)</td>
<td>Applicable using different prioritization techniques</td>
<td>Functional and non-functional</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
which are discussed in following sub-sections. Before going through the detailed description of Repizer’s components, note that two main assumptions need to be considered while using the proposed framework:

- 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.
- All elicited requirements must possess the mentioned quality attributes as described by Wiegens and Beatty (2013).

4.1 Requirements planner

The Requirements Planner contains essential information, including 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 as provided by the requirements engineer can be represented as follows: \( RS = \{ R_1, R_2, R_3, \ldots, R_n \} \). Note that an individual requirement can be represented as \( R_i \), where \( 1 \leq i \leq n \). A given set of requirements can be further categorized into \( RS_{FR} \) and \( RS_{NFR} \), where \( RS_{FR} \) represents a set of functional requirements and \( RS_{NFR} \) represents a set of non-functional requirements.

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 the PC and selected prioritization technique, Repizer can effectively prioritize the requirements depending on the requirements category (i.e., FR or NFR). 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 (PID): 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 \( P_{ID} \) from the Requirements Repository (RR). The \( P_{ID} \) can be formally represented as:

\[
P_{ID} = [(P_x \mapsto (P_{new} \lor P_{old})) \land ((C_1: P_x \in RR \lor C_2: P_x \not\in RR))]
\]

where \( P_x \) represents the initial project ID, \( P_{new} \) represents the newly assigned project ID, and \( P_{old} \) represents the existing project ID. There are two conditions \( c_1 \) and \( c_2 \) that need to be met while

![Fig. 3 High-level view of Repizer](image-url)
allocating a project ID as described in Eq. (1). Note that only one condition may hold True while assigning a project ID (exclusive OR: $\oplus$). 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, it means that 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

Requirements Prioritizer accepts inputs from Requirements Planner (Fig. 3). Next, it applies existing prioritization techniques to generate a prioritized list of requirements. It consists of two main components: (i) Requirements Manager and (ii) Requirements Grader.

Requirements Manager (RM); RM helps to regulate the activities of the Requirements Prioritizer component. First, it accepts inputs (i.e., $P_{ID}$, RC, and PC) from 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); The Requirements Grader receives collective information from RM (i.e., information taken from the requirements engineer and Requirements Planner). 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 which 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 Library of Congress SRS document (OpenURL, 2003) as a case study. This document describes an application called OpenURL Resolver that extracts metadata from the 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, title of the article, volume number, and page numbers. Based on this metadata, 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>
<thead>
<tr>
<th>Category</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General requirements</td>
</tr>
<tr>
<td>2</td>
<td>Knowledge database requirements</td>
</tr>
<tr>
<td>3</td>
<td>Services menu requirements</td>
</tr>
<tr>
<td>4</td>
<td>Help facilities for end users</td>
</tr>
<tr>
<td>5</td>
<td>Documentation for administrators</td>
</tr>
<tr>
<td>6</td>
<td>Administration and vendor support</td>
</tr>
</tbody>
</table>
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.

Table 3 Set of requirements (category and sub-category) (OpenURL, 2003)

<table>
<thead>
<tr>
<th>Category</th>
<th>Sub-category</th>
<th>Mandatory (M)</th>
<th>Desired (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2.1</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2.3</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2.4</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2.6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3.1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4.1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5.1</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>5.2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>6.1</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>6.2</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>6.3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6.4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>6.5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>6.6</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

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 the two-level prioritization based on the variables (i.e., Mandatory and Desired). The first step was that the requirements engineer assigned a unique PID to the current project, for example, P01. 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 extracted static values from the case study were: \( RS_{FR} = 134 \) (total), Mandatory (M) = 90, and Desired (D) = 44. To create user stories and levels of prioritization, the participants (total 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.

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 ten days. 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).

Table 4 Prioritized list of categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Prioritized list of category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

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
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 more easy 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** \( H_0^{\text{accuracy}} \): The accuracy of results is equal while adopting PG or AHP.

- **Alternative hypothesis** \( H_1^{\text{accuracy}} \): The accuracy of results is not equal while adopting PG or AHP.

- **Null hypothesis** \( H_0^{\text{easeofuse}} \): There is no significant difference between PG and AHP in terms of ease of use when adopted by RePizer.

- **Alternative hypothesis** \( H_1^{\text{easeofuse}} \): There is a significant difference between PG and AHP in terms of ease of use when adopted by RePizer.

### Table 5 Prioritized list of sub-categories

<table>
<thead>
<tr>
<th>Prioritized list of category</th>
<th>Sub-category</th>
<th>Prioritized list of sub-category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>2</td>
<td>2.1</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>2.3</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>2.4</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>2.6</td>
<td>2.6</td>
</tr>
<tr>
<td>4</td>
<td>4.1</td>
<td>4.1</td>
</tr>
<tr>
<td></td>
<td>6.1</td>
<td>6.1</td>
</tr>
<tr>
<td></td>
<td>6.2</td>
<td>6.2</td>
</tr>
<tr>
<td>6</td>
<td>6.3</td>
<td>6.3</td>
</tr>
<tr>
<td></td>
<td>6.4</td>
<td>6.4</td>
</tr>
<tr>
<td></td>
<td>6.5</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td>6.6</td>
<td>6.6</td>
</tr>
<tr>
<td>3</td>
<td>3.1</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>5</td>
<td>5.1</td>
<td>5.1</td>
</tr>
<tr>
<td></td>
<td>5.2</td>
<td>5.2</td>
</tr>
</tbody>
</table>

6th position, therefore, category 5 is placed at position number 6. The same case 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.
6.2 Variables

Independent and dependent variables of the experiment were identified as follows:

---

6.2.1 Independent variables

The independent variables of the experiment were the PG and AHP techniques. 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 techniques, 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 value ranging from one (very low) to five (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 easy a decision maker is able to perform the prioritization process using a given prioritization technique. In the experiment, the second dependent variable (ease of use) was measured by the 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 value ranging from one to five according to the Likert scale (Likert, 1932) where one indicated very difficult and five represented very easy. Measuring this property could help us to 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 also participated in SCORE Contest 2011 (SCORE, 2011)).

6.4 Object

The object of the experiment was a collection of 134 requirements of the OpenURL Resolver project (OpenURL, 2003). 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, title of the article, volume number, and page numbers.

Based on this metadata, OpenURL Resolver fulfills the request by consulting the related application and verifying the user’s access rights. The requirements of 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.

6.6 Experiment results

This section presents the significant results achieved from the experiment. We initially performed descriptive analysis using Microsoft Excel. We then

<table>
<thead>
<tr>
<th>Group</th>
<th>Prioritization task 1</th>
<th>Prioritization task 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RePizer using PG</td>
<td>RePizer using AHP</td>
</tr>
<tr>
<td>2</td>
<td>RePizer using AHP</td>
<td>RePizer using PG</td>
</tr>
</tbody>
</table>
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 the 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>
<thead>
<tr>
<th>Prioritization technique</th>
<th>Likert scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>AHP</td>
<td>1 2 3 4</td>
</tr>
</tbody>
</table>

Table 9 Results of measuring expected accuracy collected from post-test A

Therefore, to understand which technique produces more accurate results, the third null hypothesis was tested statistically. Before starting to test the null hypothesis (H_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 the 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 more easy 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 (see the right hand columns of Table 10), we applied the non-parametric Mann-Whitney test to investigate the second null hypothesis (H_easeofuse). The result 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.

6.7 Threats to validity

This section discusses the potential threats which could bias the validity of the experiment’s results.

- Since the results of the experiment (i.e. expected accuracy and ease of use) were obtained based on subjective opinions of the experiment’s 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’s 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

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

<table>
<thead>
<tr>
<th>Prioritization technique</th>
<th>Expected accuracy</th>
<th>Ease of use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>statics</td>
<td>df</td>
</tr>
<tr>
<td>PG</td>
<td>0.874</td>
<td>32</td>
</tr>
<tr>
<td>AHP</td>
<td>0.803</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 11 Results of measuring ease of use collected from post-test B

<table>
<thead>
<tr>
<th>Prioritization technique</th>
<th>Likert scale</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>PG</td>
<td>7 6 15 4</td>
</tr>
<tr>
<td>AHP</td>
<td>13 13 6 -</td>
</tr>
</tbody>
</table>
bias the results. To alleviate this threat, we used an automated statistical tool (SPSS) which provides more reliable results than calculating statistical values manually. It is highly probable that human error occurs 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 that is 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. Furthermore, we conducted an empirical study to evaluate the performance of RePizer while adopting either PG or AHP techniques. The evaluation was based on measuring two properties: expected accuracy and ease of use. The 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 (RKB) component to

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Dependent variable</th>
<th>Statistical test</th>
<th>$P$-value</th>
<th>Result</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{\text{accuracy}}^\text{expected}$</td>
<td>Expected accuracy</td>
<td>Mann-Whitney</td>
<td>0.007</td>
<td>Rejected</td>
<td>PG</td>
</tr>
<tr>
<td>$H_{\text{ease of use}}^\text{ease of use}$</td>
<td>Ease of use</td>
<td>Mann-Whitney</td>
<td>0.003</td>
<td>Rejected</td>
<td>PG</td>
</tr>
</tbody>
</table>
automate the selection process of the prioritization technique.

Acknowledgements

The corresponding author would like to acknowledge the financial support of the Bright Sparks Program (BSP) at the University of Malaya.

References


Likert, R., 1932. A technique for the measurement of attitudes, Archives de Psychology. 22(140):1-55.


