



Assisted requirements selection by clustering

José del Sagrado¹ · Isabel M. del Águila¹

Received: 4 November 2019 / Accepted: 29 October 2020 / Published online: 16 November 2020
© Springer-Verlag London Ltd., part of Springer Nature 2020

Abstract

Requirements selection is a decision-making process that enables project managers to focus on the deliverables that add most value to the project outcome. This task is performed to define which features or requirements will be developed in the next release. It is a complex multi-criteria decision process that has been focused by many research works, because a balance between business profits and investment is needed. The spectrum of prioritization techniques spans from simple and qualitative to elaborated analytic prioritization approaches that fall into the category of optimization algorithms. This work studies the combination of the qualitative MoSCoW method and cluster analysis for requirements selection. The feasibility of our methodology has been tested on three case studies (with 20, 50 and 100 requirements). In each of them, the requirements have been clustered, and then the clustering configurations found have been evaluated using internal validation measures for the compactness, connectivity and separability of the clusters. The experimental results show the validity of clustering strategies for the identification of the core set of requirements for the software product, being the number of categories proposed by MoSCoW a good starting point in requirements prioritization and negotiation.

Keywords Requirements selection · Next release planning · Requirements prioritization · MoSCoW · Cluster analysis

1 Introduction

Requirements engineering identifies, documents, negotiates and manages the desired features and constraints of software-intensive systems and the assumptions about the environment [1]. The most serious problems in complex systems development frequently arise from requirements engineering [2, 3]. The processes related to requirements are knowledge-intensive tasks that are often supported by human effort and expertise. Requirements are the glue that keeps together the stages in project development, because they collect the needs or conditions that the product under construction should meet to succeed.

A requirements engineering decision process, in software product development, is the definition of which features or requirements will be developed in the next release under

technical, resource, risk and budget constraints [4]. It is a complex multi-criteria decision process that, most times, entails achieving a balance between the value the requirements add to the project outcome and their cost. A clear instance is agile projects, because requirements and solutions quickly evolve through the collaborative teams and customers effort, making requirements selection a critical process.

Requirements selection has been the focus of attention in many research works, and the software industry [5–10]. The proposed techniques use different information associated with requirements. Some proposals use only one attribute, others a combination of them. Besides, both stakeholders' or developers' side could be considered inside the process. The range of suitable techniques to be applied is broad and diverse. However, both a deep knowledge of the domain and a skilful quantification capability have to be involved to succeed at using most of these techniques [1].

Although requirements engineering decision processes have been typically based both on the stakeholders' intuition and experience and on rational schemes (such as criteria, options or arguments), the recent soaring of data driven approaches in requirement engineering [11] puts pressure on practitioners to use and integrate quantitative data to decide

✉ Isabel M. del Águila
imaguila@ual.es

José del Sagrado
jsagrado@ual.es

¹ Department of Informatics, The Agrifood Campus of International Excellence (ceiA3), University of Almería, Almería, Spain

automatically on what requirements and features should be added to or removed from future releases. The use of experiments, case studies, surveys, and whatever available quantitative data is being extrapolated from software engineering empirical approaches, which emphasize the use of all kinds of empirical studies to accumulate knowledge in requirements related tasks [9].

We investigate if some kind of synergy can be found between two well-known methods, MoSCoW and clustering, as a model to automatically support the problem of requirements selection. MoSCoW is a qualitative technique for requirements prioritization which is based on the classification of requirements using plain English meaning of the prioritization categories. It helps stakeholders to make up their minds about the next release objectives, and it has the value of providing them a better understanding of the impact of their categorization decision. Clustering methods aim to partition n observations into k clusters in such a way that observations in one cluster are more similar to each other than those in other clusters. The notion of similarity used by clustering algorithms is related to that of distance and is based on quantitative data. There are several measures associated with requirements, such as development effort and stakeholders' satisfaction, which can be used for finding out requirements groups.

The research has been conducted stating a goal, which brings us into some research questions, and finally, carrying out some empirical experiments, we get the information needed to give the answers to the research questions. Our goal is to check if available quantitative records about requirements, in particular estimated development effort and clients' satisfaction, are reliable enough to define automatically the core set of requirements in a software development project (i.e. the mandatory requirements for a successful software product). Based on this goal, we derived three research questions:

- **RQ1** Can quantitative data about requirements (such as satisfaction and effort) be used to cluster and prioritize requirements automatically?
- **RQ2** Which clustering algorithm gets better results for requirements selection?
- **RQ3** Can the obtained requirements clusters be interpreted to assist in the selection of the requirements set to be developed?

The structure of the rest of the paper starts by describing the background of well-known techniques for requirements selection in Sect. 2. Then, Sect. 3 describes the quantitative formulation of the selection problem using development effort and customers' satisfaction and the two methods to be combined, MoSCoW and clustering. Section 4 is devoted to defining the process followed to check the suitability of

the combined approach and how to set the number of clusters and map MoSCoW categories to clusters. The results of the effectiveness study of the proposal for three different data sets are shown and discussed in Sect. 5. Section 6 addresses the limitations and threats to validity. Finally, Sect. 7 includes the conclusions.

2 Background

Software project planning strategies are becoming the cornerstones of software industries, because they not only define the features to be built but also exactly when they will be released and help to manage cost. The Standish Group annually reported that around 80% of surveyed software projects do not achieve their definition of success based on time, cost and scope criteria [3]. In fact, since 2015 the scope criterion, which represents how requirements have been fulfilled, has been relaxed. They also reported that one of the most relevant reasons for software project to fail, is shifting requirements, because requirements are often documented textually and the requirements specifications themselves are rarely changed [12]. This suggests that software projects fail is due to their inability to evolve efficiently to match the shifting requirements or to afford the new ones that appear in project evolution, reinforcing the importance of releases management. In consequence, making a proper decision about what functionality a release of an evolving software product should have, is critical for the success or failure of the whole project. The better the selection made is, the fewer problems with shifting requirements would be found in next releases.

To answer what to release, many approaches represent the problem as a constraint problem or as an analytic study of the features values, features dependencies, or even stakeholder' priorities or dislikes. Researchers in requirements triage, requirements prioritization, requirements selection, next release problem or release planning have been prolific from their origin [9, 13–16]. Each one, directly or indirectly, contributes to model different points of view and devises diversified solving methods.

When a software product is being developed, especially if agile methodologies are applied, requirements prioritization and selection are recurrent activities. Not only is it a process to identify and filter the important requirements, but also to solve conflicts and plan the different product deliveries. These complex decisions require a detailed knowledge of the domain and good quantification and estimation techniques of the requirements properties, also involving contradictory criteria [17]. However, the variety of prioritization methods makes it hard to select the most useful one.

Various factors and dimensions can be considered for requirements selection. Some of them are defined either by

customers or stakeholders (e.g. requirement perceived value, deadline), others by the development team (e.g. available effort, team size), or maybe by both (e.g. risk, volatility) or, going further, by external factors such as market value issues.

Many ranking techniques and prioritization techniques have been defined, each one using a subset of the information collected for requirements [8, 10, 18] or applying only people expertise. These methods may differ in the way priorities are computed, in the scale of values used to represent the resulting ordering and in the accuracy of the results. Classifying them is difficult, because several classification criteria could be used, either conjointly or independently.

Some methods are ad hoc, others use the attributes, mainly quantitative, that characterize requirements. There are techniques that manage only one attribute, but others use a combination of them, considering dependencies between requirements or not. Other techniques study only the client's position, only the team point of view, or both. Even their goals can be slightly different, since some methods select which are the requirements that should be included or rejected, or how to distribute them in releases. Prioritization approaches list is wide, even more, because some of them can be combined. Next, the most representative are briefly revised.

One well-known technique that does not use quantitative data about requirements attributes is the MoSCoW method. It proposes arranging requirements in four categories based on expert judgment and the elicited information. The term MoSCoW is derived from the first letter of each of four categories (Must have, Should have, Could have and Won't have), embedding a semantic meaning, as well [17]. Top-Ten technique can be used when a completely sorted or prioritized list of requirements is needed. The 10 most important requirements are selected by the clients.

The Kano model provides a different method to assist developers to understand customers' perspectives on product features by assessing two measures for each candidate requirement: the satisfaction and dissatisfaction. The responses to these two measures will arrange requirements into different scoring categories [19].

Another type of methods are those that use a numerical approach but without mapping numbers to specific quantitative attributes. Simple ranking is one of them. The requirements have to be sorted based on the prioritization criterion. For quantitative or ranked criteria, this approach becomes a sort problem, but the final ranking has to reach an agreement among how each client or expert ranks requirements. Cumulative voting method allows voters to distribute an explicit number of points amongst requirements. In the 100-point method, 100 units are given to stakeholders to cast their vote for their most valuable requirements [20]. Planning games approaches [21] actively involves stakeholders in ranking

processes. Time-boxing technique assesses the amount of work that the project team can deliver during the prescribed period. It is a way of focusing on achieving what needs to be done without delay or procrastination based on the effort bound, by selecting the requirements that fit in this bound.

More formal and systematic approaches are the analytic hierarchy process (AHP) and prioritization matrix [22] that use more than two criteria to prioritize requirements. AHP evaluates requirements using only one attribute (e.g. effort, value, risk) by doing a pairwise comparison in a square matrix. It converts these values to a total order relationship between the requirements for this attribute [13]. This technique can be extended to combine more than one attribute or criterion. Another cost-value method combines AHP and a graphical approach for two attributes, value and effort. It has as the overall goal to select requirements that give maximum value for minimum cost within allowable cost limits of a software system, by applying a variant of 0-1 knapsack model [23].

Requirements selection problem has been also defined as an optimization problem, called next release problem [14], that lately has been reformulated as a multi-objective problem where contradictory objectives can be defined. Search-based software engineering (SBSE) combines computational methods that use the estimated attributes of a software artefact, with expert human expertise to achieve best ranking results [9, 24, 25], giving automatic support to requirements selection. Not only release related problems but also many processes in software engineering can be formulated as optimization problems throughout the software engineering life cycle, such as non-dominated sorting of the component to be reused, or the process of automatically generating test data according to a test adequacy criterion using search-based optimization algorithms. These approaches apply meta-heuristic search techniques from requirements and project planning to maintenance and re-engineering [26].

Most SBSE works that deal with requirements focus on the problem that involves the determination of what features or requirements should be covered by the software product under construction based on the cost-value criterion [23]. The starting point is the set of candidates requirements (or product backlog) with their associated development effort and a defined number of clients, each one with their particular demands about what to include in the next version of the product under development.

3 Quantitative MoSCoW requirements selection

The strong semantic approach of MoSCoW method can be supported by the data (value, effort) postulated for the cost-value criterion [23], which is described in this section by

combining the pioneering next release models [14, 27]. This quantitative formulation offers a suitable solution for the problem by applying clustering approaches; nonetheless, the robustness of clustering algorithms could allow extending this formulation by using additional scoring dimensions for the problem, such as risks or volatility. These methods have a solid base with many successfully developed algorithms that can manage and prove the suitability of the intertwining, and the research questions proposed.

Our purpose is to use clustering as an assistant in the critical decision process about which features or requirements will be developed in the next release. This task falls into the conflict resolution subtopic [1]. In most cases, it is unwise for the developers to make a unilateral decision, so, it becomes necessary to make an agreement. It is often important, for contractual reasons, that such decisions be traceable back to the stakeholders. Figure 1 shows the workflow we propose to use clustering in this decision-making process. In the usual workflow, once the requirements are elicited and scored by stakeholders and developers, specification and analysis tasks should be executed. Analysis is where requirements prioritization is usually performed.

Our proposal extracts an automated clustering activity being in charge of the definition of the core set of requirements, those that belong to the 'must have' category. According to the resources which are available, also called team capacity or effort bound, the next release objective could be easily negotiated starting from the obtained core set of requirements.

3.1 Quantitative problem formulation

Let $\mathbf{R} = \{r_1, r_2, \dots, r_n\}$ be the set of requirements to be considered. These requirements represent enhancements or new

features that are suggested by m customers and are also candidates to be solved in the next software release. Customers are not equally important. So, each customer i will have a weight w_i , which measures its importance for the software project. Let $\mathbf{W} = \{w_1, w_2, \dots, w_m\}$ be the set of customers' weights.

Each requirement r_j in \mathbf{R} has an associated development effort e_j , which represents estimated cost for its development. Let $\mathbf{E} = \{e_1, e_2, \dots, e_n\}$ be the set of requirements efforts. The same requirement can be suggested by several customers showing a different priority for each customer. Thus, the importance that a requirement r_j has for customer i is given by a value v_{ij} . The higher the v_{ij} value, the higher is the priority of the requirement r_j for customer i .

The added value given by the requirement inclusion r_j in the next software release, also called global satisfaction, s_j , is measured as a weighted sum of its importance values for all the customers ($s_j = \sum_{i=1}^m w_i \cdot v_{ij}$). The set of requirements satisfactions is denoted as $\mathbf{S} = \{s_1, s_2, \dots, s_n\}$.

Some extended models [28] also include dislikes about a requirement to take into account user opinion regardless of the effort that its development involve. We would rather maintain the widely agreed formulation of the next release problem [14, 27] putting in the same level the stakeholders' and developers' point of view.

The problem presents several constraints that should be fulfilled in the solution found. The one is due to the effort bound, also called cost limit or capacity B , which represents the amount of available development resources. Besides, requirements present different types of interactions or dependencies, which are also problem constraints. These interactions must be considered forcing us to check whether any conflict is present whenever we try to select a new requirement. This fact forces to implement the requirements

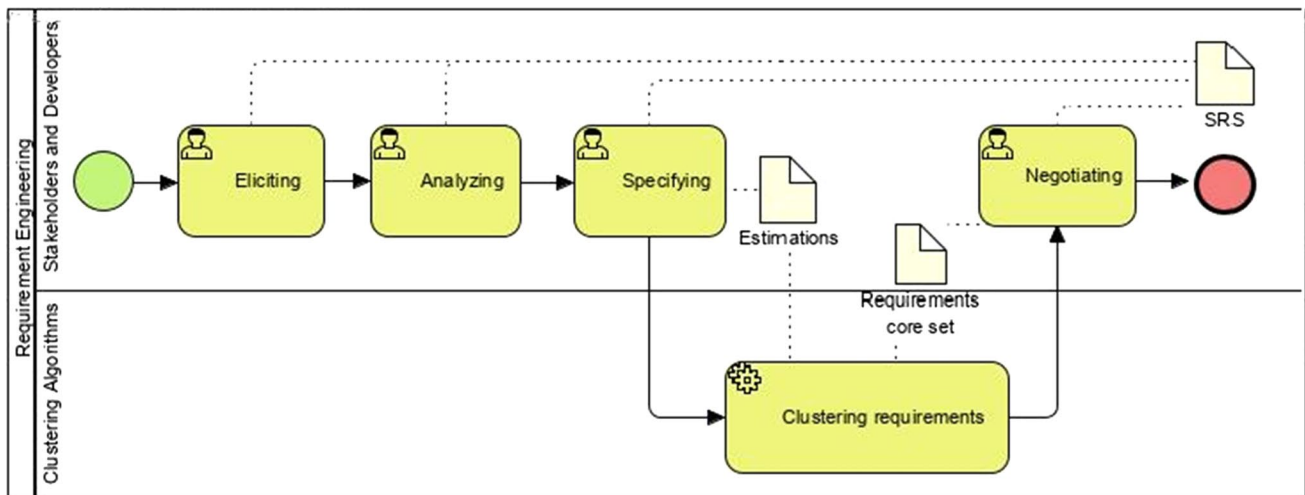


Fig. 1 Workflow for requirements selection

in a specific order [13, 29]. Dependencies can be broken down into two groups. The first can be explicitly represented as a graph G [30, 31] and comprises the functional dependencies: *Implication or precedence* ($r_i \Rightarrow r_j$) dependency stands for that the requirement r_j cannot be selected if a requirement r_i has not been implemented yet, *Combination or coupling* ($r_i \odot r_j$) represents that a requirement r_i cannot be included separately from a requirement r_j , *Exclusion* dependency ($r_i \oplus r_j$) means that the requirement r_i cannot be included together with the requirement r_j . The second group (denoted as ΔS and ΔE) includes those dependencies that imply changes in the amount of resources needed or the benefit related to each requirement: *Revenue-based* appears whether the development of a requirement r_i implies that some other requirements will increase or decrease their value; *Cost-based* relationship carries that the development of a requirement r_i implies some other requirements will increase or decrease their implementation cost. These dependencies can be modelled as a pair of $n \times n$ symmetric matrices: ΔS , where each element represents the increment or decrease of s_j and ΔE , that represents a change in the effort when implementation of two requirements take place in the same release [31]. So, we have a fivefold problem (R, S, E, L, \hat{R}) , being \hat{R} the set of requirements to be developed in the next release, that is the solution to the problem and $L = (B, G, \Delta S, \Delta E)$ the set of constraints to be considered.

3.2 MoSCoW method

Using a consistent language makes it easier to identify different classes of requirements. A simple example of this is the use of 'shall' as a keyword could indicate that a requirement is present in the written-down sentence. Some approaches go further and use 'shall,' 'should' and 'may' to indicate different priorities of a requirement. MoSCoW method reproduces this use of the language. Its basis is the fact that, even all requirements are important, the identification of vital requirements is a fundamental task. These requirements are those that contribute the most to the project success, and they are mandatory. If some of them are not

accomplished, the final product won't be a viable product. Besides, including the more important functionalities first facilitates the reception of an incremental feedback from users, schedule adjustments, and the removal of errors and misunderstandings between developers and clients in early stages which leads to customer satisfaction.

MoSCoW gets its name from an acronym formed by the following priority labels: Must have, Should have, Could have and Won't have. The letter *o* is added to make the acronym pronounceable. The classifications are shown in Table 1. MUST can also be considered an acronym for the minimum usable subset, because this category represents the compulsory requirements for the release, that is, those that have to be included in the software product in order to get clients' acceptance.

The main difference from other techniques, which classify using 'high', 'medium', or 'low', is that MoSCoW provides a semantic meaning. The stakeholders, in charge of the classification, know the real effect that their requirements assessment will produce. Human expertise and involvement is the basis of this qualitative prioritization method. MoSCoW provides a way to reach a common understanding on the relative importance of including a specific feature in the product. The categories define an agreed 'red line' that cannot be passed.

The pursued goal is to identify the set of requirements that will make up the core of a software product. That is to say, the set of fundamental requirements that the software to be developed 'must have' using as little human effort as possible. Could we match quantitative requirement attributes, such as effort or satisfaction, with MoSCoW categories to arrange requirements automatically or semi-automatically becoming the starting point for requirements negotiation? What we need next is a protocol to make groups of requirements using their associated quantitative properties.

Some other well-known models also classify the properties or features of a product into categories, such as the Kano model. This method collects a set of ideas and techniques that help us determine customers' satisfaction with product features, by describing the connection between customer satisfaction and the realization of customer requirements.

Table 1 MoSCoW categories

Label	Meaning
M Must have	Contains the requirements that must be satisfied in the final solution for the solution to be considered a success. Must can be also thought of as a minimum usable subset
S Should have	Represents the high-priority items that should be included in the solution if it is possible. These are often critical requirements but which can be satisfied in other ways if strictly necessary. Should requirements are as important as Must but may not be time-critical or may have a work around
C Could have	Describes the requirements considered as desirable but unnecessary. These will be included if time and resources permit
W Won't have	Represents the requirements that stakeholders have agreed will not be implemented in a release, but may be considered for the future

While many works establish three types of attributes to products and services (must be, one-dimensional, attractive) [32] or (dissatisfiers, satisfiers, delighters) [33], other works also manage the customers' emotions and satisfaction/dissatisfaction defining five or even six categories [34]. In consequence, the starting point will be the number of categories indicated by MoSCoW, but without taking it for granted.

3.3 Clustering

Nowadays, large amounts of data are being collected continuously, and it is becoming increasingly important to discover knowledge in multidimensional data. Here is where clustering methods come into play, because their goal is to discover groups of similar objects within a data set. Thus, clustering approaches require some methods for measuring the distance or the (dis)similarity between objects, so that objects in the same group should be closer to each other and further than those in other groups of objects. Common clustering approaches can be classified into one of the next bigger groups:

- *Partitioning methods* that divide the data into a pre-specified number of groups. *K-means* and *K-medoids* (i.e. partitioning around medoids algorithm) belong to this group [35, 36].
- *Hierarchical methods* [37] that do not require pre-specifying the number of clusters to be generated. They return a dendrogram, a tree-based representation of objects and groups. Here, *agglomerative clustering* is the most representative approach.

K-means algorithm [35, 36] is perhaps the most popular strategy to finding clusters in data. Each cluster is represented by its *centroid*, which is the mean of the observations assigned to the cluster. Initially, once the number k of groups has been specified, k observations are chosen as centroids for the clusters. The clusters are defined so that the distance between each observation in a cluster and its centroid is a minimum. Even though *K-means* is simple, fast and can deal with large data sets, the number k of clusters has to be specified in advance and the results obtained depend on the initial centroids selection. Besides, it is sensitive to the data ordering (i.e. if you rearrange your data, possibly you'll get a different solution) and, finally, the *K-means* method is also sensitive to anomalous data points and outliers.

Partitioning around medoids (PAM) [38] rests on the concept of *medoid*, which is an observation within a cluster such as the sum of the distances between it and all the other observations in the cluster is a minimum. Clusters are constructed by assigning each observation to the nearest medoid. Then, *PAM* tries to improve the quality of the clustering by interchanging medoids with the other observations

and checking whether the distances respect to the medoid are reduced. Although being more reliable and less sensitive to outliers, *PAM* requires more computation effort than *K-means*.

In contrast to partitioning methods (i.e. *K-means* and PAM), *agglomerative hierarchical clustering* groups observations based on their similarity and does not require pre-specifying the number of clusters to be produced. Initially, each observation is considered as a separate cluster (i.e. a leaf in the dendrogram). Then, the less distant clusters are successively merged until there is just one single cluster (i.e. the root of the dendrogram). Several agglomeration methods have been proposed to determine the distance between clusters, such as the maximal distance between any two observations in the clusters, the average distance between the observations in the two clusters, the distance between centroids or Ward's minimum variance criterion which minimizes the total within-cluster variance.

The unsupervised nature of these methods makes them valuable for requirements selection, as they try to discover patterns into data by assigning each observation to a previously unknown group. Clustering methods can be applied based on the quantitative properties associated with requirements without human involvement. Once the requirements are clustered, that remains is to identify (or, perhaps, relate somehow) a MoSCoW category they fit in.

4 Experimental method design

The relationships between MoSCoW and clustering should be investigated, as a model to support the problem of requirements selection. The goal of the experimental process followed, as mentioned in the introduction, is to validate whether quantitative data about requirements (satisfaction and effort) are reliable enough to select requirements automatically by applying clustering methods (**RQ1**), including the assessment of the methods to find which gets a better solution to the problem of requirements selection (**RQ2**). Besides, a matching between groups and requirements importance categories has to be found (**RQ3**). The feasibility of tackling the requirements selection problem using clustering has been studied using both an experimental approach on three case studies and a practical one to compare the results with the manual prioritization made by some developers. The stages defined next will be also applied to empirically assess the validity of our proposal:

- (i) *Prepare the data*. Clustering algorithms are affected by disparity in units, since this influences distance computation. It can be avoided if all the variables are transformed to have a mean value of 0 and a standard deviation of 1, which makes the standard deviation

the unit of measurement. As data are provided by several sources, such as software engineers (effort) or customers (values/satisfaction), different scales are used. This first standardization step ensures that experiments are performed under comparable conditions.

- (ii) *Estimate the number k of clusters.* Since a relation between the groups of requirements discovered by clustering methods and MoSCoW categories is expected, we set $k = 4$ (MoSCoW categories number) as the *fons et origo* for finding out the true relationship, cross-checking is needed (**RQ1**). The number of categories should be pore over, because other prioritization models, such as the Kano model, define different numbers of categories (**RQ3**). There are various methods for determining the optimal number of clusters \hat{k} , such as:
- *Elbow method* [39]. The number of clusters, \hat{k} , has to be chosen in a way that adding a new cluster does not produce a significant improvement in the total intra-cluster variation (or total within-cluster sum of squares), which is given by

$$\text{WSS} = \sum_{k=1}^k \sum_{x_i \in C_k} (x_i - \mu_k)^2, \quad (1)$$

where x_i is an observation belonging to the cluster C_k and μ_k is the mean value of the observations assigned to the cluster C_k . That is, the number of clusters \hat{k} is chosen so that the total within-cluster sum of squares is minimized.

- *Silhouette method* [38, 40, 41]. This technique is based on the silhouette value which measures the cohesion and separation of clusters. Let x_i be an observation, its silhouette $s(x_i)$ is defined as:

$$s(x_i) = \frac{a(x_i) - b(x_i)}{\max\{a(x_i), b(x_i)\}}, \quad (2)$$

where $a(x_i)$ is the average distance between x_i and the rest of observations in the same cluster, and $b(x_i)$ is the lowest average distance between x_i and all observations in any other different cluster. Silhouette value, $s(x_i)$, ranges from -1 to $+1$, where a high value indicates that the observation is very close the ones in its cluster and very different from observations in neighbouring ones. The clustering configuration will be appropriate when most observations have a high silhouette value. However, if many observations have a low or negative value, it could be either too many or too few clusters in the current clustering

configuration. Thus, the average $s(x_i)$ on all observations can be considered as a measure of how appropriate the clusters are. That is, the optimal number of clusters, \hat{k} , should be the one that maximizes the average silhouette over a range of candidate values for k [38].

- *Gap Statistic method* [42]. This technique compares the total within cluster variation with that expected under a reference null distribution of the observations, i.e. a distribution with no obvious clustering. The gap statistic for a given k is defined as:

$$\text{Gap}_n(k) = E_n^* \cdot \log(W_k) - \log(W_k), \quad (3)$$

where W_k is the total within-cluster sum of squares (i.e. WSS considering k clusters) and E_n^* denotes the expectation under a sample of size n obtained from the reference distribution. The optimal number of clusters \hat{k} is chosen as the smallest value of k being the gap statistic within one standard deviation of the gap statistic at $k + 1$.

While elbow and silhouette methods only measure a global clustering characteristic, gap statistic formalizes the heuristics of the former methods. The optimal number of clusters \hat{k} is estimated by applying the *majority rule* to the answers returned by the three methods, i.e. it is selected the value in which at least two of the three methods coincide. Once set, it will provide some insights into the answers to **RQ1** and **RQ3**.

- (iii) *Choose the best clustering method based on cluster validation measures.* Once the optimal number of clusters is set, the next step is to apply different clustering methods (i.e. k-means, PAM and hierarchical) and evaluate the clustering configuration found that will give an answer to **RQ2**. Ideally, we want the distance within each group to be as small as possible, keeping the average distance between groups as big as possible. Therefore, to assess a clustering configuration, internal validation measures of the clusters could be used, because they reflect the compactness, connectivity and separability of the clusters. Among the most common measures or indexes are:
- *Connectivity index* measures to what extent items are placed in the same cluster as their nearest neighbours in the observations space. For a given clustering configuration, a lower *Connectivity* indicates better clustering.
 - *Dunn index* [43] aims to identify sets of clusters that are compact and well-separated. That is, when the diameter of the clusters is expected to be small and

the distance between cluster to be large. For a given clustering configuration, a higher *Dunn index* indicates better clustering.

- *Silhouette index* [40] provides a validation of consistency within clusters. It is computed as the average of all observations silhouettes (see equation 2) and represents a measure of suitability with which they have been clustered. For a given clustering configuration, a higher *Silhouette index* indicates better clustering.
- *Caliński-Harabasz index* [44] is a variance ratio criterion to evaluate the cluster validity. It is defined as the ratio of the between-cluster variance (the variance of all cluster centroids from the centroid of the observations) to the total within-cluster variance (the average WSS of the clusters, see equation 1). Higher *Caliński-Harabasz index* values indicate better clustering, for a given clustering configuration.

Once the indicators indexes are obtained for all the clustering algorithms, the best is the one that gets the best measurement (minimizes connectivity and maximizes Dunn, Silhouette and Caliński-Harabasz indexes) (**RQ2**).

- (iv) *Analyse the clusters obtained and prioritize requirements*. At this point, we are ready to solve the problem of requirements selection using clustering and get an answer to the research questions **RQ1**, **RQ3**. We would redefine imprecisely MoSCoW categories in terms of satisfaction and effort of the requirements, as shown in Table 2. The objective is to validate if one of the identified cluster maximizes satisfaction and minimizes effort, becoming the core of the software product as a viable product. The algorithm selected (from step iii) is run, both to find the 4 groups (suggested by MoSCoW) and the optimal number of groups (determined in step ii). Besides, a statistical summary for each group, an element representative of the members of each cluster (i.e. its centroid) and a graphical representation of the clusters, should go together to analyse the clustering configuration found. These three elements will help to get an insight into the relationship between clusters and MoSCoW categories (see Table 2), to interpret

it as a mean to select requirements. Since clustering is an unsupervised technique, requirement selection will need less human effort.

5 Evaluation results

Three cases have been used for testing the effectiveness of our approach. The first one (20-Problem) is taken from [45]. It comprises 20 requirements and 5 customers weighed both in the 1 to 5 range. Each requirement has an associate effort scored between 1 and 10. Also, we consider implication and combination interactions between requirements. The second one (50-Problem) is taken from [46] that performs a case study of a project planned in 2008 that has been derived from the commonly known word processing tool MS Word. The raw data have been preprocessed to match problem formulation. In the case at hand, it comprises 50 requirements and 81 functional dependencies that have been elicited from 4 weighted customers. Each customer gives a value to the requirements and the effort that implies the development of each requirement has been estimated in person-hours by the development team. Finally, the third data set (100-Problem) is taken from [31]. It was generated randomly with 100 requirement, 5 customers and 44 requirements interactions, following the quantitative problem formulation given in Sect. 3.1. Prior to clustering, as our method indicates, the data has to be prepared. So, all these data sets were standardized to avoid the influence of the disparity in the units of satisfaction and effort associated with requirements. For the sake of clarity, the next stages will be discussed regarding each data set, giving special emphasis to the cluster analysis and requirements selection, to answer the research questions.

5.1 Empirical results for the 20 requirements problem

As the elbow and silhouette methods coincide (observe the vertical dashed line in Fig. 2 indicating the optimal number of clusters found by each method), the majority rule dictates that the optimal number of clusters can be set at $\hat{k} = 3$. In consequence, a cluster configuration for $k = 3$ and for $k = 4$ (suggested by MoSCoW) groups are going to be explored using k-means, PAM and hierarchical clustering algorithms. Table 3 collects the values achieved by the validation indexes used and best overall values are highlighted in boldface. PAM gets the best values on two indexes (i.e. Silhouette and Dunn), which suggest its choice as the best clustering algorithm for the problem at hand.

Table 2 Re-interpretation of MoSCoW categories in terms of the satisfaction and effort of the requirements

	Category	Meaning
M	Must have	High satisfaction–Low effort
S	Should have	High satisfaction–High effort
C	Could have	Low satisfaction–Low effort
W	Won't have	Low satisfaction–High effort

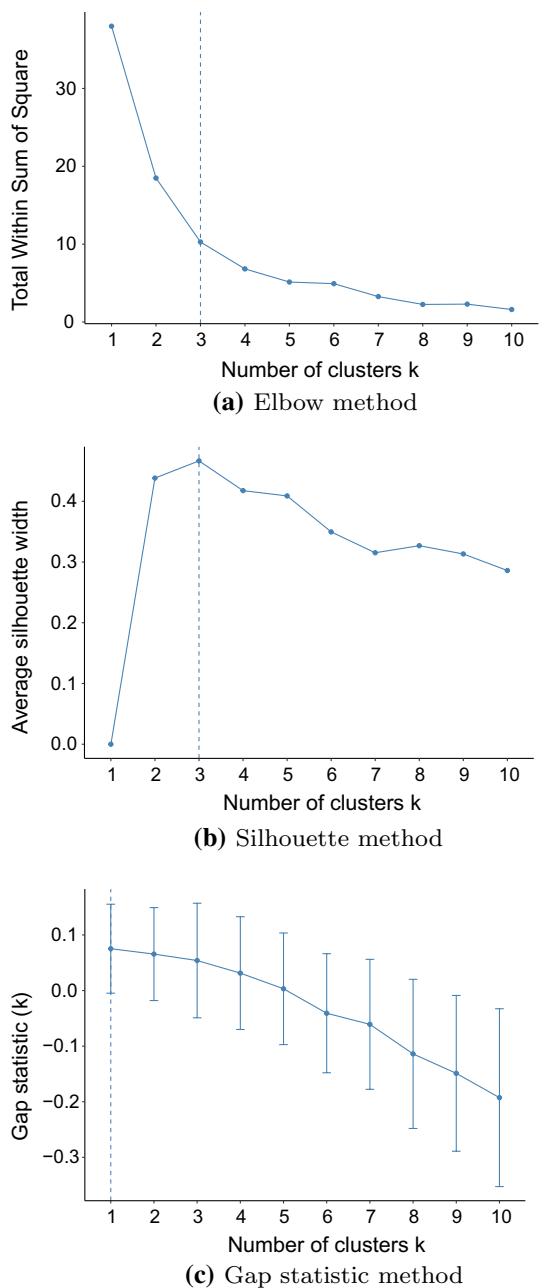


Fig. 2 Determining the optimal number of clusters (dashed line) for the 20-Problem

Table 3 Evaluation of clustering methods for the 20 requirements problem

Algorithm	K	Connectivity	Dunn	Silhouette	CH
K-means	3	12.8206	0.2090	0.4666	22.9273
PAM	3	11.5250	0.2607	0.4843	22.6144
Hierarchical	3	11.1429	0.2576	0.4549	18.6832
K-means	4	20.5603	0.2527	0.4176	24.3832
PAM	4	19.9687	0.3151	0.4116	24.0329
Hierarchical	4	19.9782	0.2482	0.3561	18.7909

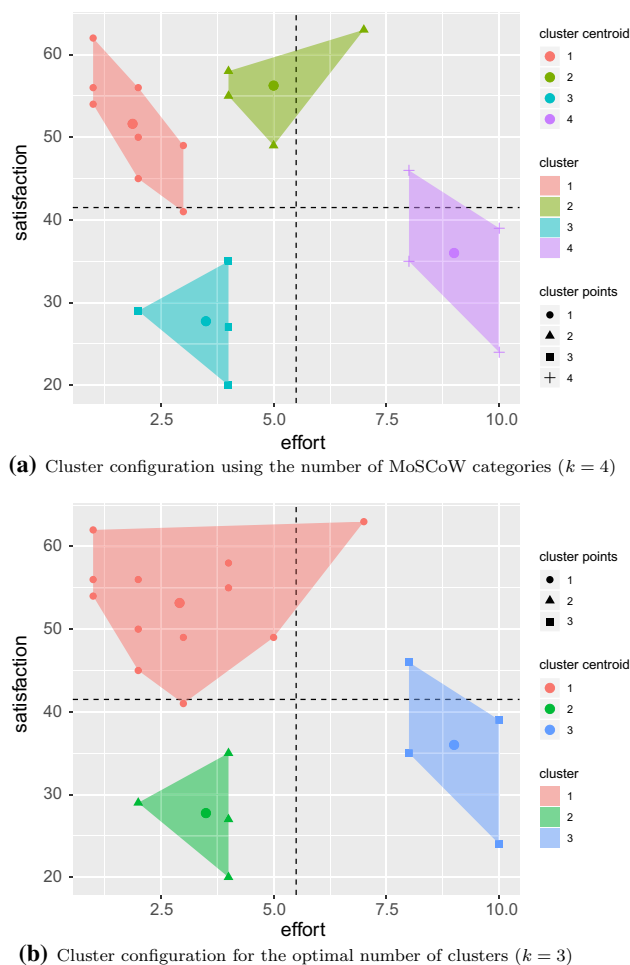


Fig. 3 Clustering for the 20-Problem

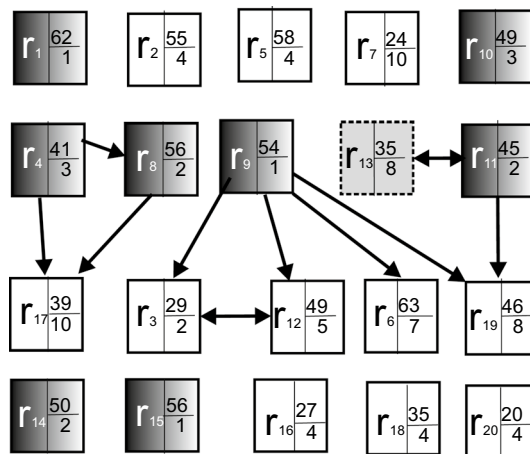


Fig. 4 The viable product based on MoSCoW categories ($k = 4$). Satisfaction, effort within each node

Table 4 Clusters summary ($k = 4$) for the 20-Problem

Size	Cluster 1		Cluster 2		Cluster 3		Cluster 4	
	8		4		4		4	
	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.
Min.	1.00	41.00	4.0	49.00	2.0	20.00	8	24.00
1st. Qu.	1.00	48.00	4.0	53.50	3.5	25.25	8	32.25
Median	2.00	52.00	4.5	56.50	4.0	28.00	9	37.00
Mean	1.87	51.62	5.0	56.25	3.5	27.75	9	36.00
3rd. Qu.	2.25	56.00	5.5	59.25	4.0	30.50	10	40.75
Max.	3.00	62.00	7.0	63.00	4.0	35.00	10	46.00
Centroids	1.87	51.62	5.00	56.25	3.50	27.75	9.00	36.00

Let us consider the MoSCoW clustering configuration (i.e. $k = 4$) found by PAM (see Fig. 3a, Table 4 and Fig. 4 for a graph representation of the solution). The core of requirements identified as those that should be included in the software development project comprises $\{r_1, r_4, r_8, r_9, r_{10}, r_{11}, r_{14}, r_{15}\}$, which has an associated effort value of 15 (which covers 17.65% of the total effort) and a satisfaction value of 413 (that represents 46.25% of the total satisfaction). However, if we want to get a minimum usable subset of requirements for developing a viable software product, the dependencies that exists between requirements have to be taken into account. Because of the coupling dependence $r_{11} \odot r_{13}$, requirement r_{13} has to be included as part of the core of requirements, shown as dotted node in Fig. 4. Thus, the viable product obtained, shown in grey in Fig. 4, has an effort value of 23 (covering 27.06% of the total effort) and a satisfaction of 448 (a 50.18% of the total satisfaction). Comparing the viable product with the core of requirements, the percentage of relative increase in satisfaction is 8.47% at the expense of a relative increase in effort of 53.33%. Figure 4 shows requirements satisfaction and effort

within each node, single arrows represent implication, and double ones depict combination.

The study of the cluster configuration found by PAM using the optimal number of clusters $\hat{k} = 3$ suggests (see Fig. 3b, Table 5) that the requirements core set identified is $\{r_1, r_2, r_4, r_5, r_6, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{14}, r_{15}\}$ with an effort of 35 (41.18% of total effort) and satisfaction of 638 (71.45% of total satisfaction). The viable product comprises the core set enlarged with two requirements $\{r_3, r_{13}\}$, because of the coupling dependencies $r_3 \odot r_{12}$ and $r_{11} \odot r_{13}$. Thus, the viable product has an effort value of 45 (52.94% of total effort) and a satisfaction of 702 (78.61% of total satisfaction). Comparing it with the requirements core set, the relative percentage increments are 10.03% in satisfaction and 28.57% in effort. It is interesting to point out that the reduction in the number of classes (from the 4 suggested by MoSCoW to the optimal number of 3) has as the result the union of the two classes in the upper left quadrant of Fig. 3a) obtained by PAM for $k = 4$, which translates into a significant increase in the viable product in terms of the number of requirements, effort and satisfaction.

Table 5 Clusters summary ($k = 3$) for the 20-Problem

Size	Cluster 1		Cluster 2		Cluster 3	
	12		4		4	
	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.
Min.	1.000	41.00	2.0	20.00	8.0	24.00
1st. Qu.	1.750	49.00	3.5	25.25	8.0	32.25
Median	2.500	54.50	4.0	28.00	9.0	37.00
Mean	2.917	53.17	3.5	27.75	9.0	36.00
3rd. Qu.	4.000	56.50	4.0	30.50	10.0	40.75
Max.	7.000	63.00	4.0	35.00	10.0	46.00
Centroids	2.916	53.166	3.500	27.750	9.000	36.000

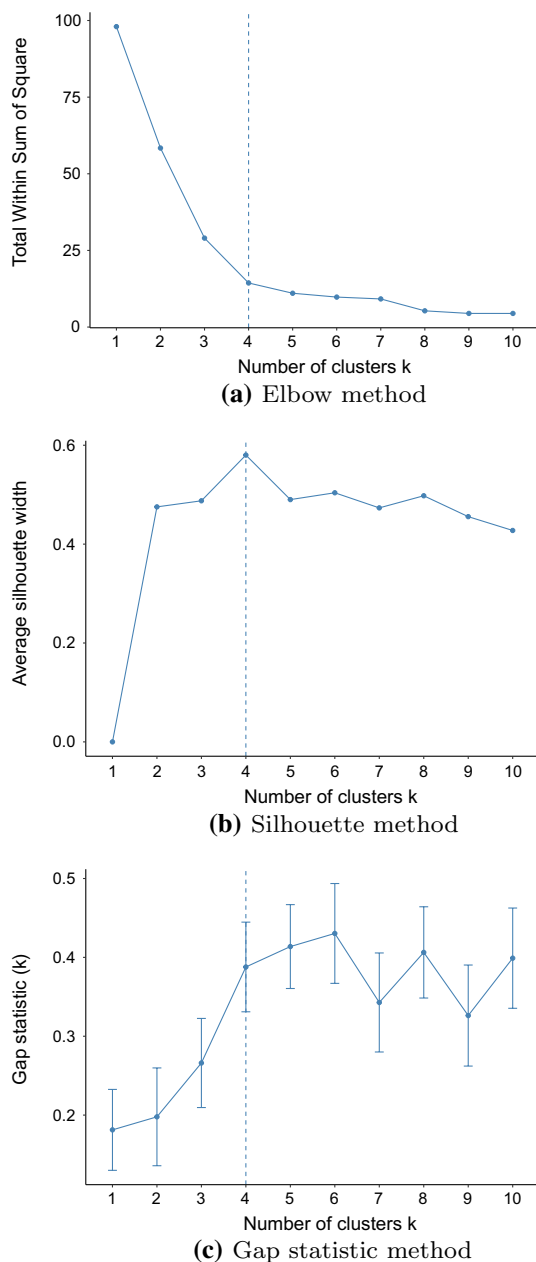


Fig. 5 Determining the optimal number of cluster for the 50-Problem

5.2 Empirical results for the 50 requirements problem

When determining the optimal number of clusters, all methods (elbow, silhouette and gap statistic) coincide (see Fig. 5) and indicate that $k = 4$ should be the number of clusters. Then, only the MoSCoW cluster configuration will be found and analysed. The validation indexes values achieved by the different clustering algorithms are shown in Table 6 (the best overall values are highlighted in boldface). *K-means* is

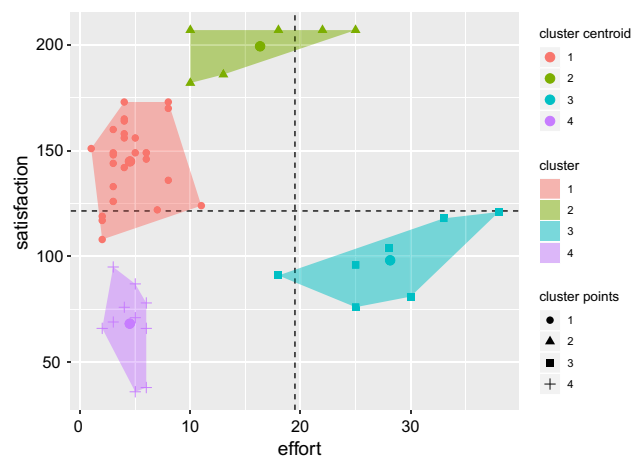


Fig. 6 Clustering for the 50-Problem

Table 6 Evaluation of clustering methods for the 50-Problem

Algorithm	K	Connectivity	Dunn	Silhouette	CH
K-means	4	14.3071	0.1317	0.5801	89.1628
PAM	4	15.7456	0.0909	0.5765	87.6092
Hierarchical	4	16.0647	0.1249	0.5655	83.4536

the best clustering algorithm, whose cluster configuration is shown in Fig. 6, see also Table 7 for detailed data.

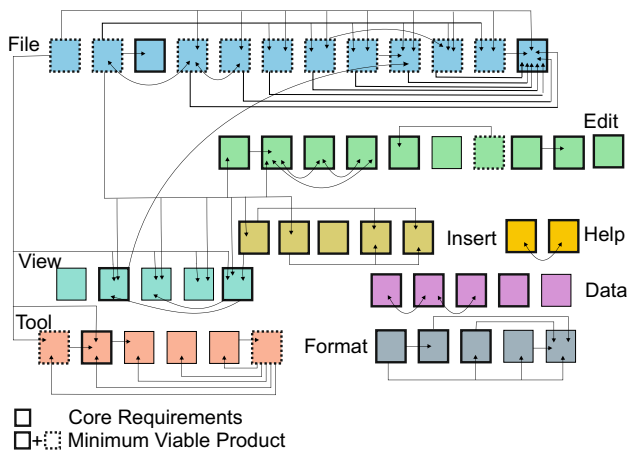
The identified requirements core set is $\{r_3, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, r_{20}, r_{21}, r_{22}, r_{24}, r_{27}, r_{28}, r_{29}, r_{30}, r_{31}, r_{32}, r_{33}, r_{34}, r_{35}, r_{39}, r_{44}, r_{45}, r_{46}, r_{47}, r_{49}, r_{50}\}$, which has to be enlarged with $\{r_1, r_2, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{19}, r_{38}, r_{43}\}$, because of implication dependencies to resemble a viable product that is shown in Fig. 7, where single arrows to represent implication, and double ones for combination. The requirements core set has an effort value of 122 (26.41% of total effort) and a satisfaction of 3913 (60.41% of total satisfaction). Once dependencies are taken into account, the viable product has an associated effort of 314 (67.97% of total effort) and satisfaction of 5700 (87.99% of total satisfaction), which, with respect to the core set, supposes an increase of 45.67% in satisfaction at the expense of increasing the effort by 157.38%. As it can be seen in Table 8, the set of functionalities that has not being included in the project comprises complementary views for the document, external links and specific location editing options, text formatting (change case and document background), additional tools and data import.

5.3 Empirical results for the 100 requirements problem

Figure 8 shows that the elbow and silhouette methods coincide in the optimal number of clusters \hat{k} and, it is set to

Table 7 Clusters summary ($k = 4$) for the 50-Problem

Size	Cluster 1		Cluster 2		Cluster 3		Cluster 4	
	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.
Min.	1.00	108.0	10.00	182.0	18.00	76.00	2.00	36.0
1st. Qu.	3.00	129.5	10.75	191.2	25.00	86.00	3.25	66.0
Median	4.00	149.0	15.50	207.0	28.00	96.00	5.00	70.0
Mean	4.52	144.9	16.33	199.3	28.14	98.14	4.50	68.2
3rd. Qu.	6.00	157.0	21.00	207.0	31.50	111.00	5.75	77.5
Max.	11.00	173.0	25.00	207.0	38.00	121.00	6.00	95.0
Centroids	4.52	144.92	16.33	199.33	28.14	98.14	4.50	68.20

**Fig. 7** The viable product for 50-Problem

3. As it differs from the number of MoSCoW categories ($k = 4$), the clusters configurations using both values of k will be explored. According to the validation indexes (see Table 9), *hierarchical* clustering gets two, out of four, best values (which are highlighted in boldface) for the indexes, becoming the selected algorithm.

The MoSCoW clustering configuration found by hierarchical clustering (see Fig. 9a, Table 10), identified cluster $\mathcal{C}_4\{r_8, r_9, r_{15}, r_{16}, r_{18}, r_{19}, r_{20}, r_{21}, r_{23}, r_{24}, r_{25}, r_{27}, r_{29}, r_{32}, r_{34}, r_{37}, r_{40}, r_{41}, r_{46}, r_{49}, r_{50}, r_{51}, r_{52}, r_{54}, r_{56}, r_{60}, r_{62}, r_{68}, r_{72}, r_{73}, r_{77}, r_{89}, r_{90}, r_{93}, r_{94}, r_{98}, r_{100}\}$, as the requirements core set. This set has an associated effort value of 218 (21.02% of total effort) and a satisfaction value of 1089 (40% of total satisfaction). From it the viable product is completed by adding the requirements set $\{r_2, r_3, r_{10}, r_{14}, r_{22}, r_{30}, r_{33}, r_{47}\}$, since the problem dependencies have to be fulfilled. Its effort and satisfaction values are 345 (33.27% of total effort) and 1282 (48.27% of total satisfaction), respectively. The relative percentage increments with respect to the requirements core set are 58.26% in effort and 17.72% in satisfaction.

However, if we use hierarchical clustering with the optimal number of clusters $\hat{k} = 3$ (see Fig. 9b, Table 11), we get

as requirements core set the union of clusters 4 and 3 found previously for $k = 4$ (Table 10 shows data for $\hat{k} = 4$). Once dependencies are taken into account, the viable product has an associated effort of 477 (50% of total effort) and satisfaction of 1733 (65.25% of total satisfaction). With respect to the core set, the relative percentage increment in effort is 44.99% and 14.01% in satisfaction. Therefore, the viable product will enlarge that found for MoSCoW ($k = 4$) and will be larger in terms of the number of requirements, effort and satisfaction.

5.4 Comparison with developers' classification

An additional validation has been made comparing our proposal with the subjective developers' prioritization. The 50-Problem is our testing scenario, because it is the only data set that includes requirements descriptions. This allows us to compare the automatic prioritization with a manual one made by developers. This comparison has certain risks such as the unknown skill level of researchers/participants or the different perceptions of the problem [47]. Nonetheless, this alternative validation approach could clarify if clustering is efficacious enough to classify requirement using MoSCoW model. Seven senior software engineering degree students, having no information on requirements dependencies, were requested to apply MoSCoW and get a prioritization of requirements. They relied only on requirements descriptions to come up with a prioritization without offering any value based rationale for making the decision about how to rate the priority [22].

Afterwards, the *majority rule* (i.e. the MoSCoW category selected was the one indicated more times by the students) was used to get the final students' classification shown in Fig. 10 and the tie-breaker is the order of the MoSCoW classes. The first finding worth highlighting is that there isn't a clear agreement between developers on the requirements that should be included in the 'must have' set. Requirements marked with '*' are those whose class is agreed by half or more of the developers.

Table 8 50-Problem—requirements description

	ID	Short name	Description
1	File-1 (*)	New File	Crete a New File
2	File-2 (*)	Open File	Open an Existing File
3	File-3	Close File	Close Current File
4	File-4 (*)	Save File	Save a File
5	File-5 (*)	Save as	Save a File as a Different File Type
6	File-6 (*)	Search File	Search for a File in The computer containing some Text
7	File-7 (*)	Protect File	Make a File Password Protected
8	File-8 (*)	Print Preview	Print Preview a File
9	File-9 (*)	Print File	Print Current File
10	File-10 (*)	Send To	Send File To Email/FAX
11	File-11 (*)	Set Properties	Set File Header Inform ton
12	File-12	Exit	Save and Exit from Word Processing Application
13	Edit-1	Undo a Task	Undo a Task and goes back to previous state
14	Edit-2	Redo a Task	Redo the most recent Change
15	Edit-3	Cut	Delete a Text and Copy to Clipboard
16	Edit-4	Copy	Copy a Text
17	Edit-5	Paste	Paste a Text from Clipboard
18	Edit-6	Paste Special	External Linking and Embedding
19	Edit-7 (*)	Go To	Go to a specific location in the current file
20	Edit-8	Find	Search for a Text in the current Document
21	Edit-9	Replace	Search and Replace a Text
22	Edit-10	Select All	Select All From Current File
23	View-1	Default	Switch to Default View
24	View-2	Print Layout	Print Layout View of Current File
25	View-3	Web layout	Web Page Layout View of Current File
26	View-4	Zoom	Zoom In/Out
27	View-5	Header/Footer	Show Healer/Footer of Current File
28	Insert-1	Page Numbers	Insert Page Numbers in Footer
29	Insert-2	Date/Time	Insert Date/Time in the File Footer
30	Insert-3	Symbol	Insert Symbol to the Cursor Location
31	Insert-4	Bookmark	Set/Update File Bookmark
32	Insert-5	Hyperlink	Insert a Hyperlinked Text
33	Format-1	Font	Change Font Setting of Selected Text
34	Format-2	Paragraph	Set the Paragraph Formatting
35	Format-3	Bullets/Numbers	Insert Bullets/Numbering to Selected Text
36	Format-4	Change Case	Change to Upper/Lower/Mixed Case selected Text
37	Format-5	Background	Change Background of the Document
38	Tools-1 (*)	Ch-Spell	Spelling Check Tool
39	Tools-2	Ch-Grammar	Grammar Check Tool
40	Tools-3	Speech	Read the Text of The Document
41	Tools-4	Mail Merge	Mail Merge with Existing Customer Database
42	Tools-5	Macro	Create/Use Automated Document Processing Function
43	Tools-6 (*)	Set Options	Configure Documents Options
44	Data-1	Insert Table	Insert a Table
45	Data-2	Delete Table	Delete an Existing Table
46	Data-3	Table Format	Format an Existing Table
47	Data-4	Sort	Sort Selected Data
48	Data-5	Import Data	Import Data from External Database
49	Help-1	Help	Loads Applicant Help File
50	Help-2	Search	Searches a Text in the Document Help File

Core requirements in bold

*Requirements included by effect of requirements implication and coupling

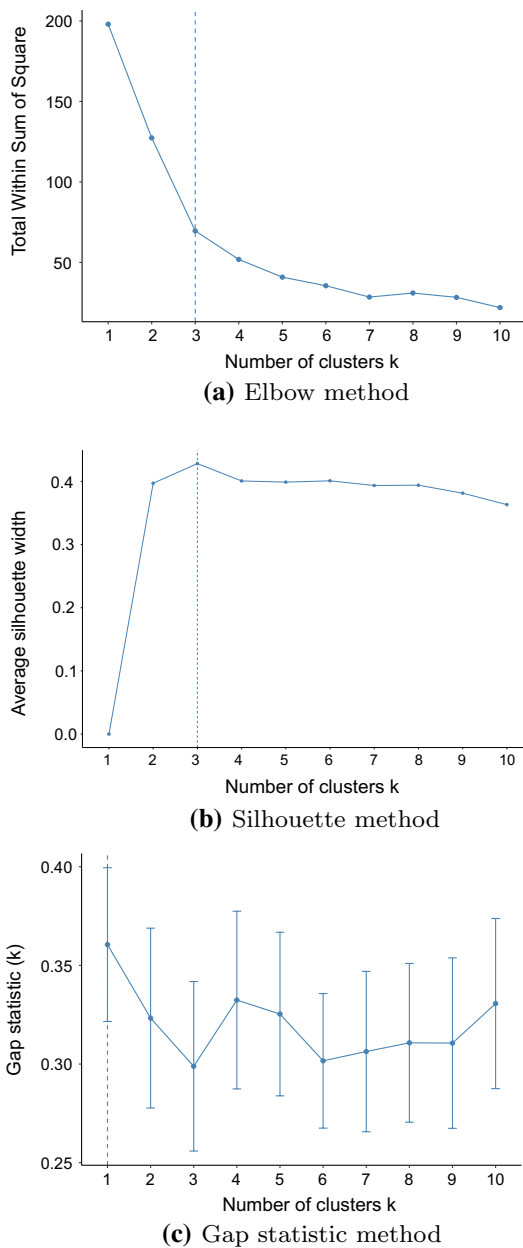


Fig. 8 Determining the optimal number of cluster for the 100-Problem

Table 9 Evaluation of clustering methods for the 100-Problem

Algorithm	K	Connectivity	Dunn	Silhouette	CH
K-means	3	17.7056	0.0548	0.4283	89.5132
PAM	3	13.4373	0.0831	0.4308	89.3966
Hierarchical	3	7.2357	0.1096	0.4278	88.0933
K-means	4	18.9746	0.0783	0.3993	90.9959
PAM	4	26.2714	0.0696	0.3993	88.7641
Hierarchical	4	14.8242	0.1096	0.3964	82.5902

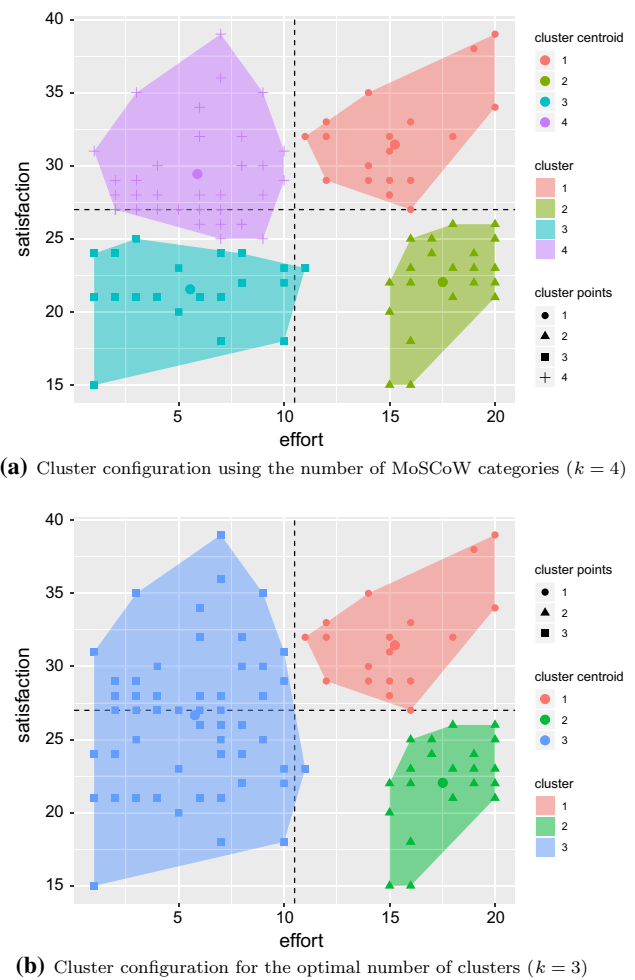


Fig. 9 Clustering for the 100-Problem

The requirements core set agreed by developers is $\{r_1, r_2, r_3, r_4, r_5, r_9, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, r_{20}, r_{26}, r_{28}, r_{31}, r_{33}, r_{38}, r_{39}, r_{44}, r_{45}, r_{46}, r_{49}, r_{50}\}$. From it we get the viable product by adding the requirements set $\{r_6, r_7, r_8, r_{10}, r_{11}, r_{19}, r_{24}, r_{29}, r_{43}\}$, since the dependencies of the problem have to be fulfilled. Requirements with a '1' in the bottom of Fig. 10 are those included in the core set, whereas those added due to dependencies are marked with a '2'. The core set has a value of 190 (41.13% of total effort) and a satisfaction of 3654 (56.41% of total satisfaction). After dependencies have been taken into account to obtain a viable product, the effort and satisfaction values change to 285 (61.69% of total effort) and 4626 (71.41% of total satisfaction), respectively. The relative percentage change with respect to the requirements core set is 50% for effort and 26.6% for satisfaction. It can be observed that developers include fewer requirements than clustering. An explanation for this could be that their prioritization is made based on natural language descriptions instead of effort and satisfaction values. Although in both approaches dependencies are

Table 10 Clusters summary ($k = 4$) for the 100-Problem

Size	Cluster 1		Cluster 2		Cluster 3		Cluster 4	
	20		23		20		37	
	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.
Min.	11.00	27.00	15.00	15.00	1.00	15.00	1.000	25.00
1st. Qu.	14.00	29.00	16.00	21.00	2.75	21.00	4.000	27.00
Median	15.00	31.50	17.00	23.00	5.50	21.50	6.000	28.00
Mean	15.25	31.45	17.52	22.04	5.55	21.55	5.892	29.43
3rd. Qu.	16.00	33.00	19.00	24.00	8.00	23.25	8.000	31.00
Max.	20.00	39.00	20.00	26.00	11.00	25.00	10.000	39.00
Centroids	15.25	31.45	17.52	22.04	5.55	21.55	5.89	29.43

Table 11 Clusters summary ($k = 3$) for the 100-Problem

Size	Cluster 1		Cluster 2		Cluster 3	
	20		23		57	
	Eff.	Sat.	Eff.	Sat.	Eff.	Sat.
Min.	11.00	27.00	15.00	15.00	1.000	15.00
1st. Qu.	14.00	29.00	16.00	21.00	3.000	23.00
Median	15.00	31.50	17.00	23.00	6.000	27.00
Mean	15.25	31.45	17.52	22.04	5.772	26.67
3rd. Qu.	16.00	33.00	19.00	24.00	8.000	30.00
Max.	20.00	39.00	20.00	26.00	11.000	39.00
Centroids	15.250	31.450	17.521	22.043	5.771	26.666

needed to get the viable product from the core set of requirements, that got by developers has lower effort and satisfaction values than in the case of clustering. This fact suggests that clustering (Must have cluster, see Table 2) could be a good starting point for requirements negotiation (see Fig. 1).

Experimental results obtained on the three data sets reinforce the idea that requirements effort and satisfaction can be used as measures for clustering requirements and to define the group of requirements that must be included into a successful next software release. That is to say, automatic selection of requirements can be approached applying clustering techniques (RQ1). The decision making processes related to software negotiation and conflict resolution can be successfully assisted by clustering the scored requirements attributes, see Fig. 1. However, there is no single applicable clustering algorithm, which is the one to be applied depends on the problem (RQ2). Although the optimal number of clusters \hat{k} differs between the case studies, this discrepancy can be interpreted, from the requirements' selection point of view, as the union of the two higher priority requirements groups (Table 2 (RQ3)), which is also advocated by other models as Kano. What is worth to highlight here is that, independently of the case at hand, clustering methods can identify a core set of requirements. However, requirements dependencies have to be managed at the time of getting a

viable product, when additional requirements have to be incorporated to the core to fulfil dependencies. The comparison with manual prioritization suggests that the subjective nature of the problem makes difficult to reach an agreement, making interesting the use or any automatic approach, such as clustering.

6 Limitations and threats to validity

The validity of any research work indicates how trustworthy and generalizable their results are [48]. The potential threats are discussed next, describing also what has been done to mitigate them.

In this paper, since the objects studied are requirements and their associated quantitative data (satisfaction and effort) are estimated and provided by software engineers and customers, a construct validity threat could appear because of the subjectivity and accuracy of the estimations. Although development teams use estimations in everyday work, to assure that experiments are performed under a common unified framework, we prepare data as the first step before clustering, as it has been described in Sect. 4, so as to unify value ranges.

Acknowledgements This research has been financed by the Spanish Ministry of Science, Innovation and Universities under the Search-based software engineering research network (RED2018-102472-T), by the Spanish Ministry of Economy and Competitiveness under project TIN2016-77902-C3-3-P (PGM-SDA II project), and it is partially supported by Data, Knowledge and Software Engineering (DKSE) research group (TIC-181) of the University of Almería.

References

- Bourque P, Fairley RE (eds) (2014) SWEBOK: guide to the software engineering body of knowledge, version 3.0 edn. IEEE Computer Society, Los Alamitos, CA
- Group S (1995) The chaos report. Technical report, Standish Group
- El Emam K, Koru AG (2008) A replicated survey of it software project failures. *IEEE Softw* 25(5):84–90
- Franch X, Ruhe G (2016) Software release planning. In: Proceedings of the 38th international conference on software engineering companion, pp 894–895. <https://doi.org/10.1145/2889160.2891051>
- Berander P, Andrews A (2005) Requirements prioritization. Springer, Berlin, pp 69–94. https://doi.org/10.1007/3-540-28244-0_4
- Svahnberg M, Gorschek T, Feldt R, Torkar R, Saleem SB, Shafique MU (2010) A systematic review on strategic release planning models. *Inf Softw Technol* 52(3):237–248. <https://doi.org/10.1016/j.infsof.2009.11.006>
- Achimugu P, Selamat A, Ibrahim R, Mahrin MN (2014) A systematic literature review of software requirements prioritization research. *Inf Softw Technol* 56(6):568–585. <https://doi.org/10.1016/j.infsof.2014.02.001>
- Thakurta R (2017) Understanding requirement prioritization artifacts: a systematic mapping study. *Requir Eng* 22(4):491–526. <https://doi.org/10.1007/s00766-016-0253-7>
- Zhang Y, Harman M, Ochoa G, Ruhe G, Brinkkemper S (2018) An empirical study of meta-and hyper-heuristic search for multi-objective release planning. *ACM Trans Softw Eng Methodol (TOSEM)* 27(1):3
- Hujainah F, Bakar RBA, Abdulgaber MA, Zamli KZ (2018) Software requirements prioritisation: A systematic literature review on significance, stakeholders, techniques and challenges. *IEEE Access*
- Maalej W, Nayebe M, Johann T, Ruhe G (2016) Toward data-driven requirements engineering. *IEEE Softw* 33(1):48–54. <https://doi.org/10.1109/MS.2015.153>
- Wagner S, Fernández DM, Felderer M, Vetrò A, Kalinowski M, Wieringa R, Pfahl D, Conte T, Christiansson MT, Greer D et al (2019) Status quo in requirements engineering: a theory and a global family of surveys. *ACM Trans Softw Eng Methodol (TOSEM)* 28(2):9
- Karlsson J, Olsson S, Ryan K (1997) Improved practical support for large-scale requirements prioritising. *Requir Eng* 2(1):51–60. <https://doi.org/10.1007/BF02802897>
- Bagnall AJ, Rayward-Smith VJ, Whittle I (2001) The next release problem. *Inf Softw Technol* 43(14):883–890. [https://doi.org/10.1016/S0950-5849\(01\)00194-X](https://doi.org/10.1016/S0950-5849(01)00194-X)
- Davis AM (2003) The art of requirements triage. *Computer* 36(3):42–49. <https://doi.org/10.1109/MC.2003.1185216>
- Domínguez-Ríos MA, Chicano F, Alba E, del Águila I, del Sagrado J (2019) Efficient anytime algorithms to solve the bi-objective next release problem. *J Syst Softw* 156:217–231. <https://doi.org/10.1016/j.jss.2019.06.097>
- Brennan K (ed) (2009) A Guide to the Business Analysis Body of Knowledge (BABOK Guide), Version 2.0. International Institute of Business Analysis
- Bühne S, Herrmann A (2019) Handbook Requirements Management according to the IREB Standard. International Requirements Engineering Board, IREB, 1 edn. Education and Training for the IREB Certified Professional for Requirements Engineering Qualification Advanced Level Requirements Management
- Kano N (1984) Attractive quality and must-be quality. *Hinshitsu (Qual J Jpn Soc Qual Control)* 14:39–48
- Leffingwell D (2003) Managing software requirements: a use case approach. Pearson Education India, Bangalore
- Beck K (2000) Extreme programming explained: embrace change. Addison-Wesley Professional, Boston
- Wieggers K, Beatty J (2013) Software Requirements, 3e. Microsoft Press, Redmond
- Jung HW (1998) Optimizing value and cost in requirements analysis. *IEEE Softw* 15(4):74–78. <https://doi.org/10.1109/52.687950>
- del Águila IM, del Sagrado J (2016) Three steps multiobjective decision process for software release planning. *Complexity* 21(S1):250–262. <https://doi.org/10.1002/cplx.21739>
- Harman M, Krinke J, Medina-Bulo I, Palomo-Lozano F, Ren J, Yoo S (2014) Exact scalable sensitivity analysis for the next release problem. *ACM Trans Softw Eng Methodol (TOSEM)* 23(2):19. <https://doi.org/10.1145/2537853>
- Harman M, Mansouri SA, Zhang Y (2012) Search-based software engineering: trends, techniques and applications. *ACM Comput Surv* 45(1):11:1–11:61. <https://doi.org/10.1145/2379776.2379787>
- Van den Akker JM, Brinkkemper S, Diepen G, Versendaal J (2005) Determination of the next release of a software product: an approach using integer linear programming. In: Proceeding of the 11th international workshop on requirements engineering: foundation for software quality REFSQ'05, pp 247–262
- Nayebe M, Ruhe G (2018) Asymmetric release planning—compromising satisfaction against dissatisfaction. *IEEE Trans Softw Eng*. <https://doi.org/10.1109/TSE.2018.2810895>
- Carlshamre P, Sandahl K, Lindvall M, Regnell B, et al (2001) An industrial survey of requirements interdependencies in software product release planning. In: Fifth IEEE international symposium on requirements engineering, 2001. Proceedings. IEEE, pp 84–91. <https://doi.org/10.1109/ISRE.2001.948547>
- Ngo-The A, Ruhe G, Shen W (2004) Release planning under fuzzy effort constraints. In: Proceedings of the third IEEE international conference on cognitive informatics, 2004, pp 168–175. IEEE. <https://doi.org/10.1109/COGINF.2004.1327472>
- del Sagrado J, del Águila IM, Orellana FJ (2015) Multi-objective ant colony optimization for requirements selection. *Empir Softw Eng* 20(3):577–610. <https://doi.org/10.1007/s10664-013-9287-3>
- Sauerwein E, Bailom F, Matzler K, Hinterhuber HH (1996) The kano model: how to delight your customers. *Int Work Seminar Prod Econ* 1:313–327
- Pohl K, Rupp C (2016) Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant. Rocky Nook Inc, San Rafael
- Matzler K, Hinterhuber HH (1998) How to make product development projects more successful by integrating kano's model of customer satisfaction into quality function deployment. *Technovation* 18(1):25–38. [https://doi.org/10.1016/S0166-4972\(97\)00072-2](https://doi.org/10.1016/S0166-4972(97)00072-2)
- MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Cam LML, Neyman J (eds) Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. University of California Press, Berkeley, pp 281–297

36. Hartigan J, Wong M (1979) Algorithm as 136: a k-means clustering algorithm. *J R Stat Soc Ser C (Appl Stat)* 28(1):100–108. <https://doi.org/10.2307/2346830>
37. Murtagh F, Contreras P (2012) Algorithms for hierarchical clustering: an overview. *WIREs Data Min Knowl Discov* 2(1):86–97. <https://doi.org/10.1002/widm.53>
38. Kaufman L, Rousseeuw PJ (1990) Finding groups in data: an introduction to cluster analysis. Wiley, New York
39. Thorndike R (1953) Who belongs in the family? *Psychometrika* 18:267–276. <https://doi.org/10.1007/BF02289263>
40. Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J Comput Appl Math* 20:53–65. [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7)
41. Amorim RCd, Hennig C (2015) Recovering the number of clusters in data sets with noise features using feature rescaling factors. *Inf Sci* 324:126–145. <https://doi.org/10.1016/j.ins.2015.06.039>
42. Tibshirani R, Walther G, Hastie T (2001) Estimating the number of clusters in a data set via the gap statistic. *J R Stat Soc B* 63(2):411–423. <https://doi.org/10.1111/1467-9868.00293>
43. Dunn J (1973) A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *J Cybernet* 3(3):32–57. <https://doi.org/10.1080/01969727308546046>
44. Caliński T, Harabasz J (1974) A dendrite method for cluster analysis. *Commun Stat* 3(1):1–27. <https://doi.org/10.1080/03610927408827101>
45. Greer D, Ruhe G (2004) Software release planning: an evolutionary and iterative approach. *Inf Softw Technol* 46(4):243–253. <https://doi.org/10.1016/j.infsof.2003.07.002>
46. Agarwal N, KariFmpour R, Ruhe G (2014) Theme-based product release planning: an analytical approach. In: 2014 47th Hawaii international conference on system sciences. IEEE, pp 4739–4748. <https://doi.org/10.1109/HICSS.2014.582>
47. Lethbridge TC, Lyon S, Perry P (2008) The management of university–industry collaborations involving empirical studies of software engineering. In: Guide to advanced empirical software engineering. Springer, Berlin, pp 257–281
48. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer, Berlin

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.