information

MDPI

Article

Semantic Clustering of Functional Requirements Using Agglomerative Hierarchical Clustering

Hamzeh Eyal Salman [1,*], Mustafa Hammad [1], Abdelhak-Djamel Seriai [2] and Ahed Al-Sbou [3]

[1] Information Technology Department, Mutah University, Mutah 61710, Jordan; hammad@mutah.edu.jo
[2] LIRMM Lab, University of Montpellier, 34000 Montpellier, France; seriai@lirmm.fr
[3] Department of Computer Science, Al-Hussein Bin Talal University, Ma'an 20, Jordan; ahed_alsbou@ahu.edu.jo
* Correspondence: hamzehmu@mutah.edu.jo; Tel.: +962-79-140-7652

Received: 31 July 2018; Accepted: 29 August 2018; Published: 3 September 2018

Abstract: Software applications have become a fundamental part in the daily work of modern society as they meet different needs of users in different domains. Such needs are known as software requirements (SRs) which are separated into functional (software services) and non-functional (quality attributes). The first step of every software development project is SR elicitation. This step is a challenge task for developers as they need to understand and analyze SRs manually. For example, the collected functional SRs need to be categorized into different clusters to break-down the project into a set of sub-projects with related SRs and devote each sub-project to a separate development team. However, functional SRs clustering has never been considered in the literature. Therefore, in this paper, we propose an approach to automatically cluster functional requirements based on semantic measure. An empirical evaluation is conducted using four open-access software projects to evaluate our proposal. The experimental results demonstrate that the proposed approach identifies semantic clusters according to well-known used measures in the subject.

Keywords: requirements elicitation; functional requirements; semantic clustering; hierarchical clustering; software requirement specifications

1. Introduction

Software has become an essential and fundamental part of modern society. Every software system is built to meet a set of needs and conditions. This set is known as Software Requirements (SRs) [1]. Generally, there are two types of SRs [2,3]. The first one concerns with the functional behavior of the software (software services) and it is called functional requirements (FRs). An example of FR from MHC-PMS (Mental Health Care-Patient Management System) is [1]: "The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month". Non-functional requirements (NFRs), the second type, focus on system restrictions and everything that is not related to software functionality (quality attributes) such as reliability, performance, portability, etc.

The very first step in developing a software system is SR elicitation [4]. In this step, developers work with the customer to find out what services the software should provide [5]. It is very important to implement the requirements elicitation step carefully because the overall development process can be seen as a derivation of the collected SRs [6]. Therefore, SRs should be solid and valid such that further modifications are limited in the future to save time and cost in the development process. Any change in the SRs after starting of source code implementation is considered to be a very timely and costly operation [7,8].

SR elicitation is a challenging task for developers and engineers [9–12]. SRs are written in natural languages and describe different aspects of the target software [1,13,14]. Mainly, software developers need to understand, analyze, and validate SRs manually. For example, the collected SRs need to be categorized and grouped in different clusters in order to break down the target project into a set of sub-projects [9]. Eventually, each sub-project covers a set of related SRs and developed by a separate and specialized developer team. Moreover, these groups of SRs need to be analyzed to generate Software Requirement Specifications (SRSs), which can be used for validating and verifying the final software product. A software developer needs to extract all related SRs before preparing the final specifications (SRS) that describe them. As a result, grouping SRs into a set of clusters helps developers to better understand and realize the target software project. In addition, it helps to design initial architecture of the software product as these clusters represent components or sub-systems that should be implemented and reused [1,15].

There is a lack of tools or methodologies to automatically classify and cluster the collected SRs. On the one hand, some works are found in the literature with the goal of non-functional SR classification [16–18]. On the other hand, functional SR clustering has never been considered in the literature. In this paper, a generic framework to automatically cluster functional SRs based on their semantics is proposed. The proposed framework can be used with any clustering algorithms and distance measures.

This paper presents a hierarchical clustering approach to semantically cluster functional SRs. The proposed approach takes as an input, FR statements of a software product in a flat list, which is documented in an SRS document. This document is an official statement about what developers should implement [19]. The approach returns as output of a set of semantic clusters based on a semantic similarity measure. First, the input is preprocessed with a proposed preprocessing step to increase the accuracy of the clustering algorithm. Then, the Agglomerative Hierarchical Clustering (AHC) algorithm is applied to cluster the target functional SRs into a set of clusters. During the clustering process, a dendrogram report is generated to visualize the progressive clustering of the functional SRs. This can be useful for software engineers to have an idea of a suitable number of clusters into which the functional SRs categorized. In other words, it can be helpful for software engineers in deciding how to break down the target project into different sub-projects.

Our proposal makes the following contributions:

- A hierarchical clustering approach to clustering software FRs based on their semantics.
- A dynamic clustering framework, which can be extended to include different clustering algorithms and distance measures.
- An empirical study to evaluate the proposed automatic clustering using four open-access software projects.

The remainder of this paper is organized into main six sections. Section 2 introduces a motivational example of our proposal. Section 3 presents the most recent work on the subject. The proposed approach is detailed in Section 4. In Section 5, we describe case studies and our research questions. In Section 6, we analyze and discuss the experimental results. Finally, conclusions and future work are stated in Section 7.

2. Motivational Example

To better understand the benefits of clustering FRs into a set of semantic clusters, we introduce E-Store software as an example. This software product provides FRs and features that allow for online sales, distribution and marketing of electronics. In addition, it focuses on the company, stakeholders and applications. This software product provides 62 FRs.

Figure 1 shows the FRs and their corresponding groups as documented manually in the SRS document of E-Store software [20]. During the requirement engineering process phase, a collection of unstructured requirements of E-Store software (62 FRs) manually are grouped into coherent clusters

and then written down in an SRS document (as shown in Figure 1b). For example, the FR number 6 and 16 [*"The system shall display detailed information of the selected products"*, *"The system shall provide browsing options to see product details"*] are manually grouped together as a cluster with title "product details ". Performing such a task is time-consuming and error-prone especially in a large software project as the FRs are organized into different sub-systems that make up the software.

3. Literature Review

This section presents research work conducted thus far, which is very related to our research work presented in this paper. Before going further, it is important to distinguish between two main concepts: clustering and classification of software requirements. Firstly, the clustering is the process of grouping or organizing requirements into an undefined number of groups (clusters). To the best of our knowledge, no research work has studied the FR clustering. Secondly, the classification is the process of assigning requirements to a predefined number of classes. The requirements broadly are classified into two categories: functional and non-functional requirements [21,22]. In the following, we present research work interested in both categories.

3.1. Functional Requirements' Classification

The classification of FRs is seldom considered in the literature of requirement engineering. Sommerville, in [1], classifies FRs into two levels: user requirements and system requirements. In [23], Ghazarian et al classified manually FRs into only five classes. These classes include input data, output data, data persistence, application rules and actions. In [24], Ghazarian also used FRs classes in [23] as a starting point to assign manually a type for each system FR. Then, he expanded and modified the class as necessary. The resulting classification scheme consists of 12 classes. This classification was limited to web-based enterprise systems.

3.2. Non-Functional Requirements' Classification

In contrast to the FR classifications, most existing approaches focused on NFR classification. Moreover, NFR classifications are very detailed. For instance, NFRs in Sommerville's textbook are classified into three main types [1]: external, product and organizational requirements. Then, product requirements are further classified into four categories: efficiency, usability, dependability and security requirements. In addition, the efficiency requirements are detailed for space and performance requirements. Moreover, the rest of the main NFR classes are constantly detailed into lower levels. Consequently, the resulting classification scheme is a hierarchy consisting of three levels.

A large number of approaches are proposed to automate the process of identifying and classifying the NFRs. We present here the most recent work on the subject. In [25], Rashwan et al. proposed an ontology-based technique to detect and classify requirements sentences into four types of NFRs (maintainability, reliability, portability, security and usability/utility) using a Support Vector Machine Classifier (VSM). In [26], Singh et al. proposed a rule-based technique to identify and classify requirements sentences in the PROMISE corpus into eight types of NFRs. Their technique used linguistic relations among requirement statements to extract thematic roles. Recently, Kaur et al. surveyed a wide range of techniques interested in NFRs [16].

3.3. Other Related Work

Indeed, the distinction between functional and non-functional requirements is not a clear-cut separation where a non-functional requirement, such as security, may generate multi-functional requirement related to security services [1]. Therefore, a number of attempts are proposed to automate the identification of such distinction in [18,27–29].

1 The system shall display all the products that can be configured.
2 The system shall allow user to select the product to configure.
3 The system shall enable user to add one or more component to the configuration.
4 The system shall notify the user about any conflict in the current configuration.
5 The system shall allow user to update the configuration to resolve conflict in the current configuration.
6 The system shall provide browsing options to see product details.
7 The system shall allow user to confirm the completion of current configuration.
8 The system shall display detailed product categorization to the user.
9 The system shall enable user to enter the search text on the screen.
10 The system shall enable user to select multiple options on the screen to search.
11 The system shall display all the matching products based on the search.
12 The system shall display only 10 matching result on the current screen.
13 The system shall enable user to navigate between the search results.
14 The system shall notify the user when no matching product is found on the search.
15 The system shall allow user to create profile and set his credential.
16 The system shall display detailed information of the selected products.
17 The system shall authenticate user credentials to view the profile.
18 The system shall allow user to update the profile information.
19 The system shall display both the active and completed order history in the customer profile.
20 The system shall display the most frequently searched items by the user in the profile.
21 The system shall allow user to register for newsletters and surveys in the profile.
22 The system shall provide online help, FAQ's customer support, and site map options for customer support.
23 The system shall allow user to select the support type he wants.
24 The system shall allow user to enter the customer and product information for the support.
25 The system shall display the customer support contact numbers on the screen.
26 The system shall allow user to select the order from the order history.
27 The system shall display the detailed information about the selected order.
28 The system shall allow user to enter the contact number for support personnel to call.
29 The system shall display the online help upon request.
30 The system shall display detailed invoice for current order once it is confirmed.
……

(**a**) Functional requirements statements of E-Store system.

……..

3. Specific Requirements

The specific requirements are –

3.1 Functionality

This subsection contains the requirements for the e-store. These requirements are organized by the features discussed in the vision document. Features from vision documents are then refined into use case diagrams and to sequence diagram to best capture the functional requirements of the system. All these functional requirements can be traced using tractability matrix.

3.1.1 Sell Configured to Ordered Products.

3.1.1.1 The system shall display all the products that can be configured.
3.1.1.2 The system shall allow user to select the product to configure.
3.1.1.3 The system shall display all the available components of the product to configure
3.1.1.4 The system shall enable user to add one or more component to the configuration.
3.1.1.5 The system shall notify the user about any conflict in the current configuration.
3.1.1.6 The system shall allow user to update the configuration to resolve conflict in the current configuration.
3.1.1.7 The system shall allow user to confirm the completion of current configuration

3.1.2 Provide comprehensive product details.

3.1.2.1 The system shall display detailed information of the selected products.
3.1.2.2 The system shall provide browsing options to see product details.

3.1.3 Detailed product Categorizations

The system shall display detailed product categorization to the user.

3.1.4 Provide Search facility.

The system shall enable user to enter the search text on the screen.
The system shall enable user to select multiple options on the screen to search.
The system shall display all the matching products based on the search
The system shall display only 10 matching result on the current screen.
The system shall enable user to navigate between the search results.
The system shall notify the user when no matching product is found on the search.

3.1.5 Maintain customer profile.

……..

(**b**) Clusters of functional requirements in SRS document of E-Store system.

Figure 1. List of functional requirements and their corresponding clusters of an E-Store system.

4. The Proposed Approach

In this section, we detail our proposal to identify semantic clusters from a given set of FRs. We first give an overview of our proposal steps. Then, we explain each step in details.

4.1. Overview of the Approach

Figure 2 presents a dynamic framework for our identification process. Mainly, this framework consists of four steps. The first step takes as an input the SRS document, which is parsed to extract a list of FR statements. In the second step, each FR statement undergoes a set of preprocessing tasks. Then, semantic similarity among FRs is computed in the third step. Finally, these FRs are organized into semantic clusters by employment of a clustering algorithm. In this framework, steps three and four can be seen as switching points to bind different semantic measures and clustering algorithms, respectively.

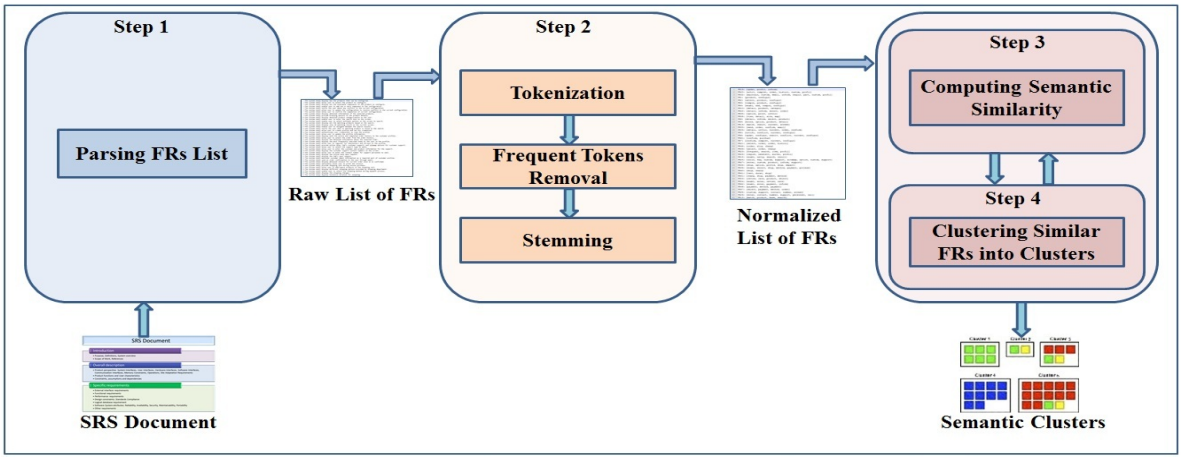


Figure 2. The proposed approach overview.

4.2. Parsing Functional Requirements

As FRs describe in detail what system should do and what developers should implement, the statements of these requirements are documented in a well-structured format in an SRS document. These statements are listed in a section under the title "Functional" in SRS document [19]. Therefore, this step parses the SRS document to locate the "Functional" section and then read each statement in this section as a functional requirement.

4.3. Pre-Processing Functional Requirements

In order to find best matching among FRs as an important step for identifying semantic clusters, the text of each statement is normalized. Such a normalization is achieved by performing three tasks: tokenization, stop word removal and stemming. These tasks are executed according to the following order.

4.3.1. Tokenization

In this task, each FR is divided into individual statements using comma and dot delimiters. Then, each statement is divided into tokens based on white space. In addition, each compound word is divided into simple tokens (e.g., *interDepartment* is divided into *inter* and *department*) using Camel case notation.

4.3.2. Frequent Tokens Removal

Rare tokens are more informative than frequent tokens in the free text because these frequent tokens have little meaning and no discriminative power [30,31]. In this task, we remove two types of

frequent tokens. The first type represents stop-words such as, *of*, *the*, *to*, etc. The second type represents a set of tokens with high frequency across all FR statements. This set depends on the case study of interest. For example, the tokens "system" and "user" exist across all FR statements of the E-Store system case study. Thus, these tokens have a high frequency and should be removed.

4.3.3. Stemming

Reducing tokens to their stems in information retrieval is known as stemming [30]. For example, *compressing* and *compresses* are reduced to *compress*. We rely on the most common English stemmer algorithm called Porter's algorithm [32] to perform this task.

4.4. Computing Semantic Similarity

As our proposal aims at constructing semantic clusters of FRs by grouping FRs that are semantically close, a semantic similarity measure is needed. Our proposal relies on the following heuristic to determine such a similarity measure:

- Heuristic [semantic similarity]: it indicates text matching between tokens derived from FR statements. Such tokens record important domain knowledge, which represents functionality(s). Therefore, when two or more FR statements share a lot of tokens, it is expected that those FRs are related to the same domain task, especially when the analysts use the same vocabulary across those FR statements.

The semantic similarity between two FRs is computed based on vector space model (VSM) [33]. It is a well-known technique in information retrieval (IR). In VSM's space, each FR is represented as a vector (call). The semantic similarity (semanticSim) between two FRs is defined in Equation (1) using cosine similarity as it is a well-known in our subject [34,35]. For two given FRs, this metric is used to determine how much relevant semantic information is shared among their corresponding token vectors:

$$semanticSim(FR_i, FR_j) = \frac{\overrightarrow{FR_i} \cdot \overrightarrow{FR_j}}{\| \overrightarrow{FR_i} \| \| \overrightarrow{FR_j} \|}. \quad (1)$$

4.5. Clustering Similar FRs into Clusters

In this final step, we employ an algorithm to group together similar FRs (of course semantic similarity). Among different possible algorithms, we pick out clustering algorithms as their functions serve the goal of this study.

Our clustering algorithm is based on Agglomerative Hierarchical clustering (AHC) [36]. However, this step is not limited to AHC but also any algorithm supporting clustering analysis can be used. Generally, AHC starts by singleton clusters such that each cluster is a single object. Then, the two most similar clusters are merged in each pass. When there are several FRs' clusters that have the same similarity to one cluster, the algorithm considers the first cluster that it encounters. Such a binary merging continues in each pass until a single large cluster is obtained. Such a cluster represents a dendrogram tree, which includes all candidate clusters of these objects. In our work, these singleton clusters initially consist of individual FRs and later clusters of FRs formed during each pass recursively. Below, we illustrate how a dendrogram tree of FRs is built and parsed to extract semantic clusters of a given set of FRs.

4.5.1. Building a Dendrogram Tree of FRs

A dendrogram tree is a representation to illustrate the arrangement of clusters generated by AHC [36]. In this work, we adapt AHC to build a dendrogram tree for a given set of FRs based on the Algorithm 1. This algorithm starts with creating a cluster for each individual FR. Then, the two most similar clusters are merged in each iteration. In our work, this similarity refers to semantic

similarity, which is computed between term vectors of the two compared clusters using semanticSim equation. Then, the two most similar clusters are replaced with a new cluster, which is the result of merging of those clusters. This process continues until a single cluster obtained. This cluster represents a dendrogram that represents a set of nested clusters. Figure 3 displays an example of a dendrogram tree. At the bottom level, a cluster is created for each FR. At the top level, all FRs belong to the single large cluster. The internal nodes (internal clusters) refer to the new clusters which are formed by merging the clusters that appear as children of those new clusters.

Algorithm 1: BuildingDendrogramTree

Input: *FRs* // Functional Requirements
Output: *dendgr*// Dendrogram Tree
1 *stack clusters* $\leftarrow$ *FRs*
2 **while** $(|clusters| > 1)$ **do**
3 $(Cl1, Cl2) \leftarrow mostSimilarClusters(clusters)$
4 $Pop(Cl1, clusters)$ //Cl1: Cluster 1
5 $Pop(Cl2, clusters)$ //Cl2: Cluster 2
6 $Cl3 \leftarrow Merge(Cl1, Cl2)$
7 $Push(Cl3, clusters)$
8 **end**
9 *dendgr* $\leftarrow$ *clusters*
10 **return** *dendgr*

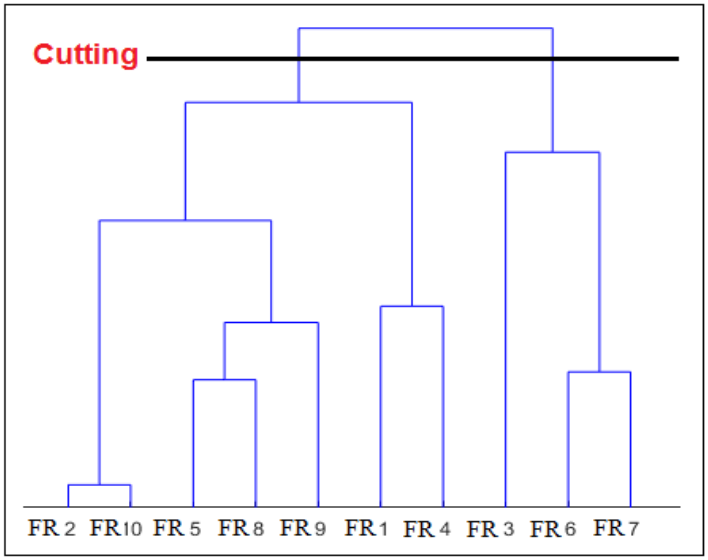


Figure 3. A graphical representation of a dendrogram tree.

4.5.2. Identifying Semantic Clusters

As a dendrogram tree is a hierarchy of nested clusters, when this hierarchy is cut off at a specific point based on predefined criteria, a set of clusters is obtained (see Figure 3). Each one is a candidate semantic cluster. Therefore, in this step, we propose Algorithm 2 based on a depth-first search to determine the appropriate cutting point. The input of this algorithm is a dendrogram tree and the output is a set of semantic clusters. The algorithm compares the semantic similarity value of each node (parent node) starting from the root with its children's nodes (its left and right nodes). If the similarity value of the parent node is less than the average similarity value of its sons, the algorithm goes to the next immediate sons. Otherwise, the parent node is identified as a semantic cluster, added to the

accumulator (semCluster) and the algorithm goes to the next node in the stack (pile). Along these lines, the most relevant semantic clusters are identified as the traversal proceeds.

Figure 3 is a simple example to show an imaginary picture of the execution of Algorithm 2. In this figure, the horizontal line crosses the hierarchy in two points (called cutting points). Therefore, two semantic clusters are identified. The first cluster has seven FRs {FR2, FR10, FR5, FR8, FR9, FR1, FR4} while the second cluster consists of three FRs {FR3, FR6, FR7}.

Algorithm 2: Identifying Semantic Clusters

```
Input: dendgr // a dendrogram tree of FRs
Output: semClusters // array of semantic clusters
stack pile
pile.push(root(dendgr))
while (|pile| > 0) do
    parent ← pile.pop()
    Cl1 ← getLeftCluster(parent, dendgr)
    Cl2 ← getRightCluster(parent, dendgr)
    avg ← (Similarity(Cl1), Similarity(Cl2))/2
    if (Similarity(parent) > avg) then
        add(parent, semClusters)
    else
        pile.push(Cl1)
        pile.push(Cl2)
    end
end
return semClusters
```

5. Experimental Evaluation Settings

The goal of this section is to describe case studies and the research questions being investigated in this work.

5.1. Case Studies

The effectiveness of our proposed approach is evaluated using SRS documents of four open-access software products from different domains with different sizes. These products are: E-Store system [20], WASP system [37], UUIS system [20] and MHC-PM system [38].

The E-Store allows for online sales, distribution and marketing of electronics. The WASP system is web architectures for services platforms. The UUIS system is a unified university inventory system to access and manage the integrated inventory. The MHC-PM system is a mental health care patient management system.

Table 1 shows statistical information of each software product of interest in terms of number of clusters, size of these clusters and number of FRs.

Table 1. Statistics on software products of interest.

Software Product	#Clusters	Cluster's Size			#FRs
		Min	Avg	Max	
E-Store System	20	1	3	7	62
WASP System	14	1	4.7	8	66
UUIS System	11	1	2.3	4	25
MHC-PM System	6	1	3.2	6	19

5.2. Investigation Research Questions and Evaluation Procedure

As this work aims at organizing a set of given FRs into a set of clusters which are semantically correct, we investigate the following main research questions which collectively meet the objective of this study:

- RQ1: to what extent are the identified clusters semantically correct? This is recognized as semantic clustering.
- RQ2: to what extent is the number of identified clusters close to the actual number of clusters? This is recognized as clustering gap.

To address the first research question (RQ1), we need a measurement to assess the semantic clustering of identified clusters. For this, we rely on two well known measures in the Information Retrieval (IR) field. These metrics are Precision and Recall [30]. Both Precision and Recall take values in a range between 0 and 1. The ideal value for Precision and Recall is 1. For each identified cluster, we evaluate the cluster correctness by executing the following steps:

- We match the identified cluster (i.e., their FRs) with all actual clusters of a software product of interest. The actual clusters are manually clustered and documented in an SRS document of each case study by analysts. Let X be the identified cluster. The cluster that has a large number of matches in terms of FRs with X is called a reference cluster of X.
- We compute precision and recall values based on the reference cluster according to the following equations:

$$Precision = \frac{|\{REF\_CLUSTER\} \cap \{IDE\_CLUSTER\}|}{|\{IDE\_CLUSTER\}|}, \quad (2)$$

$$Recall = \frac{|\{REF\_CLUSTER\} \cap \{IDE\_CLUSTER\}|}{|\{REF\_CLUSTER\}|}, \quad (3)$$

where *REF_CLUSTER* and *IDE_CLUSTER* refers to FRs of reference and identified clusters, respectively. If the Precision is equal to 1, this indicates that all FRs of identified cluster is a subset of the reference cluster's FRs. If the Recall value is equal to 1, this refers to all FRs of the reference cluster being a subset of the identified cluster's FRs. Therefore, both Precision and Recall should be used together as complementary parts to evaluate the correctness of the identified cluster.

To answer the second research question (RQ2), we need to compare the number of identified clusters with the actual number of clusters of case study of interest. Such a comparison represents clustering gap metric. For simplicity, we call this metric as *C_Gap*. The values of this metric is computed by applying the following equation:

$$C\_Gap = |\#I\_CLUSTER - \#A\_CLUSTER|, \quad (4)$$

where *#I_CLUSTER* and *#A_CLUSTER* represent the number of identified and actual clusters, respectively. The ideal value of the *C_Gap* is 0, which refers to that number of identified clusters is equal to the number of the actual cluster of the case study of interest.

As a summary, the results mainly evaluated by *Precision*, *Recall* and *C_Gap*. These metrics work together as an integrated group. Precision and Recall are used to evaluate each identified cluster individually while *C_Gap* is used to evaluate the number of identified clusters.

6. Results Analysis and Discussion

In this section, our research questions are answered and the experimental results of our proposed approach are analyzed.

6.1. Semantic Clustering (RQ1)

In order to answer the first research question (RQ1), we analyze and evaluate the semantic clustering of the identified clusters. This is performed using Precision and Recall metrics presented in Section 5.2. Table 2 shows average Precision and Recall values of identified clusters from each software product. These values indicate that the FRs of these identified clusters are semantically grouped together to form semantic clusters. Indeed, Precision values take a high-range (0.72–0.83) and Recall values take a reasonable range (54–61) across different software products.

Table 2. Average Precision and Recall of the identified clusters for each software product.

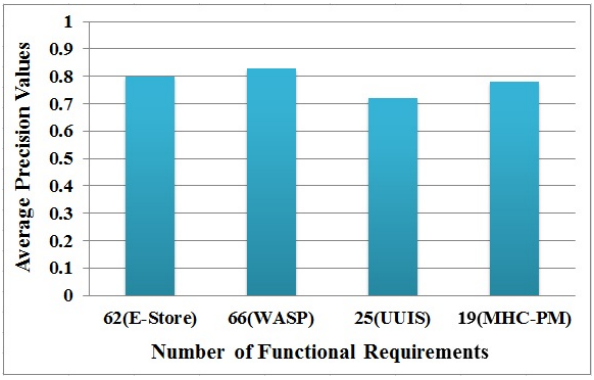
Software Product	Average Precision	Average Recall
E-Store System	0.80	0.61
WASP System	0.83	0.54
UUIS system	0.72	0.60
MHC-PM System	0.78	0.57

In Figure 4, we graphically display the average Precision and Recall values of identified clusters from each software product against the number of FRs. As shown in this figure, the average values for Precision are relatively close to each other. In addition, the average values for Recall are relatively close to each other. This represents an indicator that our proposed approach works effectively regardless of the number of FRs.

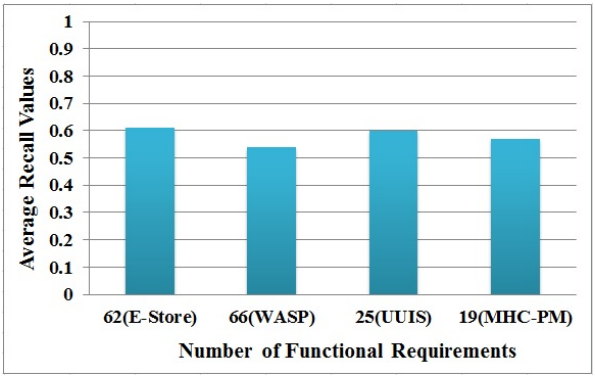
Table 3 shows statistics for Precision and Recall values of identified clusters from each software product. It is noticeable that standard deviation (StdDev) of these values is low. In addition, the maximum values for Precision and Recall reach the ideal value (1.0). As a result, this is evidence that our approach identifies semantic clusters that always have high Precision and Recall values across different sizes of software products.

Table 3. Statistics on precision and recall of the identified clusters for each software product.

Software	Precision			Recall		
	Avg	StdDev	Max	Avg	StdDev	Max
E-Store System	0.80	0.28	1.0	0.61	0.29	1.0
WASP System	0.83	0.23	1.0	0.54	0.31	1.0
UUIS system	0.72	0.29	1.0	0.60	0.29	1.0
MHC-PM System	0.78	0.25	1.0	0.57	0.35	1.0



(**a**) Average Precision values of identified clusters against the number of FRs.



(**b**) Average Recall values of identified clusters against the number of FRs.

Figure 4. Average Precision and Recall of identified clusters against the number of FRs.

In summary, the answer of the first research question (RQ1) is that our approach identifies clusters with confidence that these clusters are semantic clusters regardless of the number of FRs provided by a software product. This answer is based on the average precision and recall values and their statistics shown in Table 3.

6.2. Clustering Gap (RQ2)

It is important, before going further, to provide statistics about the identified clusters from software products of interest. Table 4 shows, for each software product, statistical information in terms of number of identified clusters and sizes of these clusters (minimum, average and maximum sizes). By comparing it with Table 1 (statistics about actual clusters), we note that the number of identified clusters is close to the actual number of clusters (shown in Table 1) of each software product. In addition, the size of identified clusters is similar to their corresponding actual clusters. Consequently, these statistics represent an indicator about the efficiency of our proposed approach.

Table 4. Statistical information of the identified clusters for each software product.

Software Product	#Clusters	Cluster's Size		
		Min	Avg	Max
E-Store System	19	2	3.3	11
WASP System	18	1	3	13
UUIS system	9	1	2.8	8
MHC-PM System	6	1	3.2	6

Figure 5 visualizes the relationship between the number of the identified clusters and number of FRs across all software products of interest. In addition, this figure visualizes the relationship between the number of identified clusters and the number of actual clusters. Based on this figure, it is clear that the number of identified clusters is very close to the number of actual clusters across different software products regardless the number of FRs provided by that product. This fact is made evident by Table 5 that describes the minor gaps between the number of actual and identified clusters for each software product.

In summary, the answer of the second research question (RQ2) is that the number of identified clusters for a given software product are the same or very similar to their corresponding actual clusters. The answer of this question is based on Tables 4 and 5.

We present different reference and identified clusters for all studied case studies (WASP System, UUIS System, E-Store System and MHC-PM System) in Tables 6–9, respectively. The shaded FRs in each identified cluster are irrelevant FRs in that cluster.

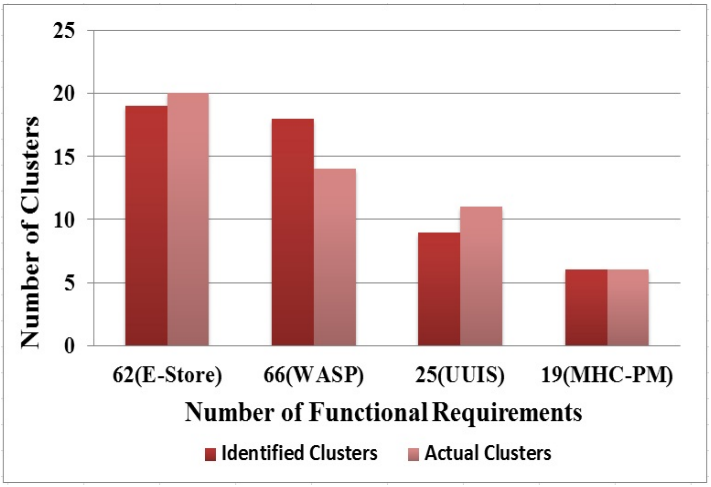


Figure 5. Number of identified clusters against number of FRs for each software product.

Table 5. C_Gap Values of the identified clusters for each software product.

Software Product	C_Gap
E-Store System	1
WASP System	4
UUIS system	2
MHC-PM System	0

Table 6. An example of a semantic cluster identified from the WASP system.

Semantic Cluster Members	Reference Cluster Members
The WASP platform MUST allow end-users to set an alert on an event	The WASP platform MUST allow end-users to set an alert on an event
The WASP platform SHOULD allow the end-user to specify the notification type when setting an alert	The WASP platform SHOULD allow the end-user to specify the notification type when setting an alert
The WASP platform MUST maintain a list of events the end-user can be notified about	The WASP platform MUST maintain a list of events the end-user can be notified about
The WASP platform SHOULD be able to decide how to notify the user of an alert for which an event was set	The WASP platform SHOULD be able to decide how to notify the user of an alert for which an event was set
The WASP platform MUST actively monitor all events	The WASP platform MUST actively monitor all events
The WASP platform MUST allow the end-user to remove previously set alerts on events	The WASP platform MUST allow the end-user to remove previously set alerts on events
If the user cannot be notified of the event the first time, the WASP platform SHOULD retry to notify the user of the occurrence of the event, until the user has been notified or a specified time-out elapses	If the user cannot be notified of the event the first time, the WASP platform SHOULD retry to notify the user of the occurrence of the event, until the user has been notified or a specified time-out elapses
The WASP platform MUST notify the end-user about the occurrence of an event for which an alert was set, as soon as the event occurs	The WASP platform MUST notify the end-user about the occurrence of an event for which an alert was set, as soon as the event occurs
The WASP platform SHALL allow end-users to maintain a buddy list	

Table 7. An example of a semantic cluster identified from an E-Store System.

Semantic Cluster Members	Reference Cluster Members
The system shall allow user to create profile and set his credential	The system shall allow user to create profile and set his credential
The system shall authenticate user credentials to view the profile	The system shall authenticate user credentials to view the profile
	The system shall allow user to update the profile information

Table 8. An example of a semantic cluster identified from a UUIS system.

Semantic Cluster Members	Reference Cluster Members
Any DA group member or authorized inventory group member asset is owned by the department	Any DA group member or authorized inventory group member asset is owned by the department
Any faculty member can add all related departments inventory	Any faculty member can add all related departments inventory
Any university group member can add all assets in the inventory	Any university group member can add all assets in the inventory
Inter departments: request must be approved by a DA group member and faculty group member unless it came from a higher level group	A bulk entry can be used to add many assets

Table 9. An example of a semantic cluster identified from MHC-PM system.

Semantic Cluster Members	Reference Cluster Members
System will generate a daily list of patients who missed their appointments and email/SMS to the clinician responsible for the patient's care	System will generate a daily list of patients who missed their appointments and email/SMS to the clinician responsible for the patient's care
System should generate weekly reports for each clinic the number of patients attending each day, the number of patients attended for a mental health treatment, the number of patients suffering from each condition and the total amounts of drugs provided as medication	System should generate weekly reports for each clinic the number of patients attending each day, the number of patients attended for a mental health treatment, the number of patients suffering from each condition and the total amounts of drugs provided as medication
System will generate a daily list of patients who attended an appointment but don't have an updated record and email to the clinic where the patient attended the appointment	System will generate a daily list of patients who attended an appointment but don't have an updated record and email to the clinic where the patient attended the appointment
System should generate regular reports on number of patients attended each clinic each month summary of prescriptions issued and the average wait time by the patients but omit the patient details	System should generate regular reports on number of patients attended each clinic each month summary of prescriptions issued and the average wait time by the patients but omit the patient details
Clinical Staff should be able to look up patient information including appointment history, diagnosis history, prescriptions history	
System should highlight the patients with a high or increased risk classification level	

6.3. Threats to Validity

In this section, we discuss the limitations of our proposal in terms of the internal and external threats. These threats as follows:

- As the identification process mainly relies on textual matching among FRs to identify semantic clusters, our approach is sensitive to the vocabulary used in FR statements. Thus, it does not come as a surprise that our proposal successes and failures depend on the vocabulary used. The obtained results can not be generalized to all software products where the use of the same vocabulary yields best results while the different vocabularies represent an internal threat. However, this threat is shared by all research work that depends on textual matching between SR artifacts.
- An important external threat to validity is the use of a limited number of case studies to evaluate the effectiveness of our proposal. Despite the face that the used case studies are fair enough to validate our proposal, a larger number of case studies is needed for a better test.

7. Conclusions and Future Work

7.1. Conclusions

In this paper, we present an approach to group software functional requirements of a given product into semantic clusters. The identification process employs textual similarity between functional requirement statements. Such similarity reflects the domain's functions and tasks embedded in functional requirement vocabulary. The proposed approach relies on an agglomerative hierarchical clustering algorithm. However, our proposal is a dynamic clustering framework, which can be extended to include different clustering algorithms and distance measures. The empirical study that is conducted using four open-access software products shows that our proposal achieves high performance according to the well-known measures in the subject. Moreover, the experimental results

show that the proposed approach identifies semantic clusters that reflect the domain functionalities embedded in given functional requirements.

7.2. Future Work

We plan to do a comparative study in order to investigate the results of different clustering algorithms for clustering functional requirements. In addition, we plan to document each cluster by extracting key words from FR statements of that clusters. Such words describe the domain knowledge embedded in that cluster.

Author Contributions: Conceptualization and Research Idea by H.E.-S. and M.H.; Methodology, Software, Result Analysis and Validation by H.E.-S.; Writing—Original Draft Preparation by H.E.-S. and M.H.; Reviewing and Editing by A.-D.S and A.A.-S.

Funding: This research received no external funding.

Acknowledgments: We thank the anonymous reviewers for their comments. In addition, We thank LIRMM laboratory (https://www.lirmm.fr/), which will pay the publication fee.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SRs	Software Requirements
FRs	Functional Requirements
NFRs	Non-Functional Requirements
SRSs	Software Requirement Specifications
AHC	Agglomerative Hierarchal Clustering
SVM	Support Vector Machine
StdDev	Standard Deviation
MHC-PMS	Mental Health Care-Patient Management System

References

1. Sommerville, I. *Software Engineering*, 9th ed.; Addison-Wesley Publishing Company: Boston, MA, USA, 2010.
2. Davis, A.M. *Software Requirements: Objects, Functions, and States*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1993.
3. Pressman, R. *Software Engineering: A Practitioner's Approach*, 7th ed.; McGraw-Hill, Inc.: New York, NY, USA, 2010.
4. Kotonya, G.; Sommerville, I. *Requirements Engineering: Processes and Techniques*, 1st ed.; Wiley: Hoboken, NJ, USA, 1998.
5. Pitangueira, A.; Tonella, P.; Susi, A.; Maciel, R.; Barros, M. Minimizing the stakeholder dissatisfaction risk in requirement selection for next release planning. *Inf. Softw. Technol.* **2017**, *87*, 104–118. [CrossRef]
6. Pitangueira, A.M.; Maciel, R.S.P.; Barros, M. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. *J. Syst. Softw.* **2015**, *103*, 267–280. [CrossRef]
7. McGee, S.; Greer, D. Towards an understanding of the causes and effects of software requirements change: Two case studies. *Requir. Eng.* **2012**, *17*, 133–155. [CrossRef]
8. Kuhn, A.; Ducasse, S.; Gírba, T. Semantic Clustering: Identifying Topics in Source Code. *Inf. Softw. Technol.* **2007**, *49*, 230–243. [CrossRef]
9. Zowghi, D.; Coulin, C. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and Managing Software Requirements*; Aurum, A., Wohlin, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 19–46.
10. Sommerville, I.; Sawyer, P.; Viller, S. Viewpoints for requirements elicitation: A practical approach. In Proceedings of the 1998 Third International Conference on Requirements Engineering, Colorado Springs, CO, USA, 10 April 1998; pp. 74–81.

11. Bitencourt, A.S.; Paiva, D.M.; Cagnin, M.I. Requirements elicitation from business process model in bpmn: A systematic review. In Proceedings of the XII Brazilian Symposium on Information Systems on Brazilian Symposium on Information Systems: Information Systems in the Cloud Computing Era, Florianopolis, Brazil, 17–20 May 2016; pp. 200–207.
12. Azmeh, Z.; Mirbel, I.; Crescenzo, P. Highlighting stakeholder communities to support requirements decision-making. In *Requirements Engineering: Foundation for Software Quality*; Doerr, J., Opdahl, A.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 190–205.
13. Kamalrudin, M.; Grundy, J.; Hosking, J. Tool support for essential use cases to better capture software requirements. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'10), Antwerp, Belgium, 20–24 September 2010; pp. 255–264.
14. Sagar, V.B.R.V.; Abirami, S. Conceptual modeling of natural language functional requirements. *J. Syst. Softw.* **2014**, *88*, 25–41. [CrossRef]
15. Alebrahim, A. Framework for identifying meta-requirements. In *Bridging the Gap between Requirements Engineering and Software Architecture: A Problem-Oriented and Quality-Driven Method*; Springer: Wiesbaden, Germany, 2017; pp. 51–109.
16. Harsimran, K.; Ashish, S. Non-functional requirements research: Survey. *Int. J. Sci. Eng. Appl.* **2015**, *3*, 172–182. [CrossRef]
17. Deocadez, R.; Harrison, R.; Rodriguez, D. Automatically classifying requirements from app stores: A preliminary study. In Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal, 4–8 September 2017; pp. 367–371.
18. Ishrar, H.; Leila, K.; Olga, O. Using linguistic knowledge to classify non-functional requirements in srs documents. In *Natural Language and Information Systems*; Kapetanios, E., Sugumaran, V., Spiliopoulou, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 287–298.
19. IEEE-SA Standards Board. *IEEE Recommended Practice for Software Requirements Specifications*; IEEE Std 830-1998; IEEE: Piscataway, NJ, USA, 1998; pp. 1–40.
20. National Research Council of Italy. Natural Language Requirements Dataset. Available online: http://fmt.isti.cnr.it/nlreqdataset/ (accessed on 3 June 2018).
21. Eckhardt, J.; Vogelsang, A.; Fernández, D.M. Are "Non-functional" Requirements Really Non-functional? An Investigation of Non-functional Requirements in Practice. In Proceedings of the 38th International Conference on Software Engineering (ICSE'16), Austin, TX, USA, 14–22 May 2016; pp. 832–842.
22. Sommerville, I.; Sawyer, P. *Requirements Engineering: A Good Practice Guide*, 1st ed.; Wiley: New York, NY, USA, 1997.
23. Ghazarian, A.; Tehrani, M.S.; Ghazarian, A. A software requirements specification framework for objective pattern recognition: A set-theoretic classification approach. In Proceedings of the 2011 16th IEEE International Conference on Engineering of Complex Computer Systems, Las Vegas, NV, USA, 27–29 April 2011; pp. 211–220.
24. Ghazarian, A. Characterization of functional software requirements space: The law of requirements taxonomic growth. In Proceedings of the 2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, 24–28 September 2012; pp. 241–250.
25. Rashwan, A.; Ormandjieva, O.; Witte, R. Ontology-based classification of non-functional requirements in software specifications: A new corpus and svm-based classifier. In Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference, Kyoto, Japan, 22–26 July 2013; pp. 381–386.
26. Singh, P.; Singh, D.; Sharma, A. Classification of non-functional requirements from SRS documents using thematic roles. In Proceedings of the 2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS), Gwalior, India, 19–21 December 2016; pp. 206–207.
27. Kurtanović, Z.; Maalej, W. Automatically classifying functional and non-functional requirements using supervised machine learning. In Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, 4–8 September 2017; pp. 490–495.
28. Rashwan, A. Semantic analysis of functional and non-functional requirements in software requirements specifications. In Proceedings of the 25th Canadian Conference on Advances in Artificial Intelligence (Canadian AI'12), Toronto, ON, Canada, 28–30 May 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 388–391.

29. Shah, T.; Patel, S. A novel approach for specifying functional and non-functional requirements using rds (requirement description schema). *Procedia Comput. Sci.* **2016**, *79*, 852–860. [CrossRef]
30. Manning, C.D.; Raghavan, P.; Schütze, H. *Introduction to Information Retrieval*; Cambridge University Press: New York, NY, USA, 2008.
31. Term Freuency Weighting. Available online: http://opencourseonline.com/ (accessed on 3 November 2017).
32. Porter, M.F. An algorithm for suffix stripping. *Program* **1980**, *14*, 130–137. [CrossRef]
33. Salton, G.; Wong, A.; Yang, C.S. A vector space model for automatic indexing. *Commun. ACM* **1975**, *18*, 613–620. [CrossRef]
34. Marcus, A.; Maletic, J.I. Recovering documentation-to-source-code traceability links using latent semantic indexing. In Proceedings of the 25th International Conference on Software Engineering (ICSE'03), Portland, OR, USA, 3–10 May 2003; pp. 125–135.
35. Saied, M.A.; Abdeen, H.; Benomar, O.; Sahraoui, H. Could we infer unordered api usage patterns only using the library source code? In Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (ICPC'15), Florence, Italy, 18–19 May 2015; pp. 71–81.
36. Zhao, H.; Qi, Z. Hierarchical agglomerative clustering with ordering constraints. In Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining, Phuket, Thailand, 9–10 January 2010; pp. 195–199.
37. Menzies, T., Krishna, R., Pryor, D. The Promise Repository of Empirical Software Engineering Data. Available online: http://openscience.us/repo/requirements/requirements-other/wasp.html (accessed on 3 June 2018).
38. Mental Health Care Patient Management System. Available online: https://bscs143.files.wordpress.com/2015/11/requirement-mhc-pms.docx (accessed on 3 June 2018).