Generation of Microservice Names from Functional Requirements: An Automated Approach

Sebastian Arias, Aracely Suquisupa, Maria Fernanda Granda, and Víctor Saquicela[https://orcid.org/0000−0002−2438−9220]

University of Cuenca, Department of Computer Science
{sebastian.arias,aracely.suquisupa, fernanda.granda, victor.saquicela}@ucuenca.edu.ec

Abstract. We study the problem of manual creation and definition of microservices in the context of modern software architectures, where high levels of knowledge and experience are often required. Our proposed solution aims to find the ideal names for microservices automatically, by means of the application of Natural Language Processing techniques, graph analysis and community detection, including the use of artificial intelligence language models such as ChatGPT, among others. The results showed that the proposed process is a robust and effective guideline for automatic microservice naming, ensuring consistency in the final outcome.

Keywords: Microservices · requirements · NLP · ChatGPT

1 Introduction

In recent years, service-oriented architectures have evolved towards microservices, providing new strategies for software development and enabling high-quality features delivery [1]. However, the process of defining microservices remains a complex activity mainly depending on human expertise. Requirements engineering plays a crucial role in this process, since it looks for ways, models or processes to facilitate the implementation of functional requirements into information systems [3].

In this context, Natural Language Processing (NLP) has been recognized to be useful on automatically analyzing functional requirements, which are usually described in natural language [2]. NLP techniques, including the use of artificial intelligence language models such as ChatGPT, enable the study of text to identify relevant patterns and connections.

Another area of interest that intersects with NLP is graph analysis, which allows representing textual data by means of constructing graphs and exploring semantic and syntactic relationships between elements. For example, community detection in graphs is a popular technique that identifies groups of connected nodes based on their similarity or relatedness.

In this paper, we propose an approach built on top of the process proposed by Tyszberowicz [4], which decomposes functional requirements using visualization and clustering tools, though still requiring human intervention for optimal partitioning of microservices. This limitation is addressed using an automatic approach which combines NLP techniques and community detection methods in graphs, a textual data preprocessing process to create an adjacency matrix and build graphs, using community detection algorithms to identify possible microservice names. This approach aims to simplify the work of developers in analyzing requirements and defining initial microservices.

2 Background and Related Works

Requirements engineering is a systematic and disciplined process that plays a critical role in the success of software projects [5], it focuses on understanding the needs, expectations and requirements of stakeholders, and ensuring the completeness, consistency, verifiability and feasibility of the identified requirements [6]. Functional requirements describe the behavior that a system function must fulfill, based on business needs and objectives. On the other hand, non-functional requirements focus on emergent properties of the software, constraints and external interfaces [7].

In recent years, an evolution in software architectures has emerged, moving from monolithic approaches to the adoption of microservices. This transformation represents an innovation by eliminating the monolithic nature of older systems and employing new software partitioning strategies to deliver high quality features [1].

On the other hand, Natural Language Processing (NLP) emerged at the intersection of artificial intelligence and linguistics [8] and has established itself as a branch of artificial intelligence that focuses on the analysis and generation of human language using computational methods and algorithms [9]. NLP has a wide range of applications, from virtual assistants such as Siri [1] and Alexa [2], to machine translation systems and sentiment analysis in social networks. It also plays a key role in the development of chatbots, recommender systems and data analytics.

In the field of computer science and mathematics, graphs are structures used to visualize data sets and their relationships. They are composed of vertices or nodes connected by edges, and may have associated weights or costs. A fundamental tool for representing graphs is the adjacency matrix, a two-dimensional data structure that indicates the connections between nodes through the elements of the matrix. Graphs are models of communication in interconnected networks and find applications in a variety of fields, such as social network analysis and the design of computer algorithms[12].

Clustering is a data mining and machine learning technique that consists of grouping similar data into clusters with the objective of detecting significant

[1] https://www.apple.com/es/siri/
[2] https://developer.amazon.com/es-ES/alexa.html

patterns and structures. In this paper, the density-based clustering technique [14] is employed, i.e., community detection in graphs seeks to identify groups of nodes with a community structure, which provides information about the organization of the graph and allows nodes to be classified according to their membership in a specific community. In this paper, Louvain's algorithm is employed to detect communities in graphs [15].

Software development using microservices architecture aims to overcome the limitations of monolithic architecture styles, where the user interface, logic, and databases are integrated into a single application, which can make it difficult to understand and maintain. However, one of the main challenges in microservices architecture design is to find a suitable partitioning of the system into microservices [20]. Therefore, microservices design is usually performed intuitively, based on the experience of the system designers [4].

In the paper "Identifying Microservices Using Functional Decomposition" [4], the authors propose an approach to identify microservices at the early design stage, which is to identify the relationships between required system operations and the state variables of those operations, and then visualize these relationships in a graph to identify dense relationship groups. This process is carried out using the CoCoME [21] requirements, which are decomposed and represented in a network, and then the nodes are manually grouped to obtain the desired microservices.

In his paper "Experiment on Automatically Extracting Functional Requirements" [2], Wang proposes an approach that combines machine learning, natural language processing, and parsing techniques to automatically extract functional requirements and facilitate requirements analysis. The approach consists of several stages, including document collection related to e-commerce systems and automobile manufacturers, extraction of sentences containing functional requirements, and natural language processing involving tokenization, lemmatization, part-of-speech tagging, and dependency parsing. The experiments conducted demonstrate good domain-independent performance and some robustness [2].

Author Vidya Sagar proposes in her article "Conceptual modeling of natural language functional requirements"[22] an alternative approach that uses lexical and syntactic analysis techniques to accurately and comprehensibly capture functional requirements expressed in natural language. This approach seeks to improve accuracy and comprehensibility in requirements gathering by preprocessing words, extracting syntactic features, and creating conceptual models.

The use of artificial intelligence techniques has seen a significant increase in recent months due to its widespread popularity, and has been used by both businesses and consumers for a wide variety of tasks, especially related to text processing [23]. Recently, the use of ChatGPT in software development has been explored as a promising tool. Previous research has highlighted its ability to improve code quality, facilitate refactoring, speed up requirements elicitation, and assist in software design [24].

White proposes in his paper "ChatGPT: Using Prompts for Effective Software Requirements Elicitation"[25] the use of prompts to improve software re-

quirements elicitation through interaction with large-scale language models, such as ChatGPT. These prompts are instructional patterns that guide the model during interactions to effectively elicit requirements. The study demonstrates how the application of these prompts helps elicit more accurate and complete requirements when interacting with ChatGPT, allowing users to ask specific questions and get relevant answers.

3 Process for generating microservice names from functional requirements

This section presents the proposed process for obtaining microservice names from functional requirements, including illustrative examples. Figure 1 shows the process and describes the phases in detail.

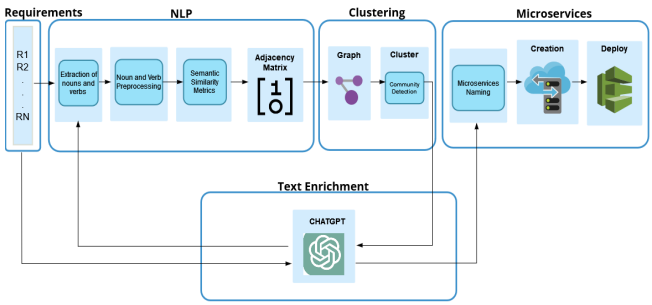


Fig. 1. Microservices identification process from functional requirements

3.1 Requirements collection

The process considers the requirements formalized in use cases defined in CoCoME, through a case study exposed by Rausch [21], in which a commercial system for a supermarket chain is shown and presents common processes such as selling, ordering and receiving products, inventory reports, price updates, exchange of products between stores (in low stock), etc. The descriptions and names of the CoCoME requirements are stored in a spreadsheet for later use, all requirements are available on the github site[3]. The requirements are formally defined in the equation 1

$$R = \{r_1, r_2, ..., r_n\} \tag{1}$$

where R is the set of requirements and where each element is represented from r_i to r_n where the interval of i is: $1 \leq i \leq n; \forall i \in N$ and represents a finite

[3] https://github.com/sebasari11/nlp-microservices/blob/main/requerimientos.xlsx

set of n requirements of R, and each r_i contains a set of words. To illustrate the process, we will exemplify with three of the nine proposed requirements, which are listed below:

1. Requirement r_1: At the Cash Desk the products a Customer wants to buy are detected and the payment, either by credit card or cash, is performed.
2. Requirement r_2: If some conditions are fulfilled a Cash Desk automatically switches into an express mode. The Cashier is able to switch back into normal mode by pressing a button at his Cash Desk. To indicate the mode the Light Display shows different colors.
3. Requirement r_3: The Trading System provide the opportunity to order product items.

3.2 Natural Language Processing (NLP)

Extraction of nouns and verbs The extraction of nouns and verbs from a requirement is crucial to understand the semantics and organizational structure. Nouns represent the entities of the system, while verbs describe the expected actions. The formalization is represented as follows:

$$NLP(R) = S, S_o \tag{2}$$

$$NLP(R) = V, V_O \tag{3}$$

Where $S = \{s_1, s_2, ...s_n\}$, each s_i represents an element of S and $1 \leq i \leq n; \forall n \in N$ represents the finite set of n nouns and $V = \{v_1, v_2, ...v_n\}$, each v_i represents an element of V and $1 \leq j \leq n; \forall n \in N\}$ represents the finite set of n verbs of V, obtained from the requirements of R.

The technology used for natural language processing (*equations 2 and 3*) is the pretrained model "en_core_web_sm"[4] from Spacy[5], one of the most robust libraries in the NLP field. This model is known for its ability to apply various NLP techniques, such as morphological analysis, lemmatization, parsing, named entity identification, and dependency classification.

The process detected 19 extracted nouns *(CashDesk, Product, Customer, Payment, CreditCard, Cash, Condition, CashDesk, ExpressMode, Cashier, NormalMode, Button, CashDesk, Mode, LightDisplay, DifferentColor, TradingSystem, Opportunity, ProductItem)* named as S and 15 verbs identified *(want, buy, detect, detect, perform, perform, fulfil, fulfil, switch, switch, press, indicate, show, provide, order)* in the requirements referred to as V.

Spacy also allows obtaining information such as the lemma, noun name and noun stem that are used to establish the relation with the verbs. In this way, the set represented in the equation 4 is obtained and displayed in the table 1.

$$S_O = \{(a_1, b_1, s_1), (a_2, b_2, s_2), ..., (a_n, b_n, s_n)\} \tag{4}$$

[4] https://spacy.io/models/en
[5] https://spacy.io/

Sustantivos S_o (name, lema y root)	**Verbs V_0 (name y lema)**
(nombre='the cash desk', lema='cashdesk', raiz='at') (nombre='the products', lema='product', raiz='detect') (nombre='a customer', lema='customer', raiz='want') (nombre='the payment', lema='payment', raiz='detect') (nombre='credit card', lema='creditcard', raiz='by') (nombre='cash', lema='cash', raiz='card') (nombre='some conditions', lema='condition', raiz='fulfil') (nombre='a cash desk', lema='cashdesk', raiz='switch') (nombre='an express mode', lema='expressmode', raiz='into') (nombre='the cashier', lema='cashier', raiz='be') (nombre='normal mode', lema='normalmode', raiz='into') (nombre='a button', lema='button', raiz='press') (nombre='his cash desk', lema='cashdesk', raiz='at') (nombre='the mode', lema='mode', raiz='show') (nombre='the light display', lema='lightdisplay', raiz='show') (nombre='different colors', lema='differentcolor', raiz='show') (nombre='the trading system', lema='tradingsystem', raiz='provide') (nombre='the opportunity', lema='opportunity', raiz='provide') (nombre='product items', lema='productitem', raiz='order') Total Sustantivos: 19	(nombre='wants', lema='want') (nombre='buy', lema='buy') (nombre='are detected', lema='detect') (nombre='detected', lema='detect') (nombre='is performed', lema='perform') (nombre='performed', lema='perform') (nombre='are fulfilled', lema='fulfil') (nombre='fulfilled', lema='fulfil') (nombre='switches', lema='switch') (nombre='switch', lema='switch') (nombre='pressing', lema='press') (nombre='indicate', lema='indicate') (nombre='shows', lema='show') (nombre='provide', lema='provide') (nombre='order', lema='order') Total Verbos: 15

Table 1. Nouns and Verbs

Where each element of S_O is represented in the equation 5.

$$s_o i = \{(a_i, b_i, s_i) | a_i \in Noun, b_i \in Root, s \in S\} \tag{5}$$

Where a_i represents the noun name, b_i represents the noun root and s_i represents the noun lemma, where s_i are elements that are part of the set S. Moreover, the sets $Noun$, $Root$ and S contain all possible nouns, lemmas and roots respectively, these sets are obtained from each requirement in R.

Information such as the lemma and the verb (verb or verb + auxiliary) is obtained from the verbs, which is obtained as a set represented in the equation 6 and displayed in the table 1.

$$V_O = \{(c_1, v_1,), (c_2, v_2), ..., (c_n, v_n)\} \tag{6}$$

Where each element of V_O is represented in the equation 7.

$$v_o i = \{(c_i, v_i) | c_i \in \text{Noun}, v \in V\} \tag{7}$$

Where c_i represents the verb name and v_i is the corresponding lemma, where v_i belongs to the set V. In turn, the sets $Name$ and V contain all possible names and lemmas obtained from each requirement in R.

Noun and Verb Pre-processing Preprocessing of nouns and verbs, involves removing duplicate nouns and verbs, removing stop words, such as: articles, pronouns and prepositions [26] to improve the quality and precision of nouns and verbs. To perform the cleaning of verbs and nouns we used firstly the list of all punctuation marks [6] given by Python and secondly the NLTK library [7] which provides stop words [26] commonly known as stopwords. The elimination of stop

[6] https://docs.python.org/3/library/string.html
[7] https://www.nltk.org/

words reduces the complexity of the obtained nouns and verbs, facilitating their analysis. In addition, all nouns and verbs of both S and V are converted to singular.

$$(set \circ NLTK)(S) = set(NLTK(S)) = S_2 \tag{8}$$

The final result of nouns is a new set S_2 containing the unique and clean noun elements of the set S.

$$(set \circ NLTK)(V) = set(NLTK(V)) = V_2 \tag{9}$$

The final result of verbs is a new set V_2 containing the unique and clean verb elements of the set V.

By applying these two techniques, we detected the presence of nouns and their frequency of occurrence in the set S_O, in this case the noun *CashDesk* is repeated three times, so two are eliminated. In the case of the verbs in the set V_O , the number of times they are repeated is detected: *detect* two repetitions, *perform* two repetitions, *fulfil* two repetitions and *switch* with two repetitions. In both cases the number of repeated verbs and nouns are removed from their respective sets. The new sets after the reduction of two nouns from the set S, named now S_2 *(cash, customer, expressmode, condition, productitem, mode, payment, cashier, opportunity, lightdisplay, button, differentcolor, cashdesk, product, creditcard, tradingsystem, normalmode)* and the reduction of two verbs from the set V named the set V_2 *(buy, indicate, order, press, provide, fulfill, show, perform, switch, want, detect).*

Syntactic Similarity Metrics The algorithms of *Jaccard* [10] and *SequenceMatcher* [11] are applied for syntactic comparison of the sets of nouns (S_2) and verbs (V_2) obtained in the previous phase. Through these algorithms, each element of each set is compared with all other elements of the same set S_2 with S_2 and V_2 with V_2 respectively. Subsequently, the average of the results obtained by both algorithms is calculated, which generates a similarity metric that varies between 0 and 1. Based on this similarity metric, we proceed to replace the similar verbs and nouns in the corresponding sets.

$$Similar(S_2, S_2) = S_3 \tag{10}$$

$$Similar(V_2, V_2) = V_3 \tag{11}$$

The result of applying such similarity metrics are the sets S_3 and V_3 respectively, for example, the noun *productitem* was replaced by *product*, so they were eliminated from the set S_2 due to their similarity with other nouns present in the set. Similar verbs were not eliminated, since in the three requirements used there are no repetitions, however, when executing all the requirements, an increase in the substitution of nouns and verbs is observed due to the greater amount of information present.

The results of this phase generated a set of 16 nouns S_3 *(lightdisplay, customer, mode, button, opportunity, cashier, differentecolor, expressmode, tradingsystem, normalmodel, product, payment, creditcard, cash, cashdesk, condition)* and a set of 11 verbs V_3 *(perform, detect, fulfil, want, order, indicate, provide, buy, show, switch, press)* product of the application of the previously mentioned syntactic similarity metrics.

Adjacency Matrix The generation of the adjacency matrix represents the connection relations between the elements of the sets S_3 and V_3 in a word graph. For this purpose, a process is followed that is based on the information obtained from the sets S_3 and V_3. This process involves identifying the verbs and nouns relevant to the adjacency matrix, to achieve this, the sets V_3 and S_3 must be traversed. First, each element of V_3 is traversed so that each verb obtained acts as a row of the matrix. Then, the elements of S_3 are traversed, which will act as columns of the matrix. In other words, for each verb in V_3, each of the nouns in S_3 is traversed, this process is shown visually in the figure 2.

Fig. 2. Tour of V_3 and S_3

During the traversal of each verb in V_3 and each noun in S_3, a comparison is made with each element of the set S_O. Importantly, each element of S_O is compared individually with each noun in S_3 and each verb in V_3, respectively. For this comparison, the lemma of the noun and the root of the verb are considered, such comparison allows identifying possible connections, the formalization of this process is shown in the equation 12.

$$S_4 = s.lema \mid s_3i \cap s.lema \wedge v \cap s.root \wedge s \in S_O \wedge v \in V_3 \wedge s_3j \in S_3 \qquad (12)$$

The set S_4 is obtained by filtering S_O and retaining those nouns whose roots are verbs in V_3 and whose lemmas belong to S_3, as can be seen in figure 3.

It is important to emphasize that the nouns and verbs that are compared with each lemma and root must be exactly the same and not similar, because a previous substitution of all similar verbs and nouns was previously carried out.

If the size of S_4 is nonzero, it indicates that there is a connection between an element s_{i3} of the set S_3 and a verb v in the set V_3. Therefore, the adjacency matrix is represented as:

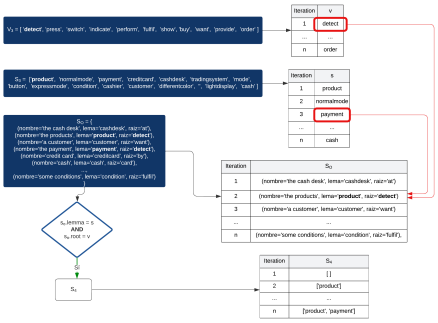


Fig. 3. Example of S_4

$$M[i,j] = \begin{cases} 1 & \text{si } len(S_4) > 0 \\ \\ 0 & \text{si } len(S_4) \leq 0 \end{cases} \tag{13}$$

Where if the length of S_4 is greater than zero, the value of 1 is assigned in the formation of the matrix, otherwise the value of 0 is assigned.

As a result of the described process, the adjacency matrix $M[i,j]$ is presented in figure 4.

	button	cash	cash desk	cashier	condition	credit card	customer	differen tcolor	express mode	light display	mode	normal mode	opportunity	payment	product	trading system
detect	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
provide	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
show	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0

Fig. 4. Adjacency matrix

3.3 Clustering

With the adjacency matrix M representing in a graph the values of i and j containing entities and relationships that have been processed in previous phases, it allows an analysis of the relationships between entities, as well as the identification of any patterns or groups that may be present in the data. Formally this is expressed as:

$$N = \{1, 2, \ldots, n\} \tag{14}$$

$$E = \{(i,j) \in N \times N \mid M[i,j] \neq 0\} \tag{15}$$

$$G = (N, E) \tag{16}$$

Where N is the set of nodes and n is the total number of nodes of the graph G. The equation 15 concerning edges, uses the condition that edge (i,j) exists if and only if the value of M[i,j] is nonzero. Equation 16 defines the graph G as a tuple of its set of nodes N and its set of edges E. The table 2 shows the result of this process from the adjacency matrix.

Once the graph G is obtained from the matrix $M[i, j]$, we proceed to verify the existence of isolated nodes. If nodes are detected that have no connection with any other node of the graph, they are eliminated to simplify the structure and avoid possible errors in the subsequent analysis. The elimination of isolated nodes can also help to simplify the graph and facilitate its interpretation. The result of this process is shown in the table 2.

$$Eliminar\_Aislados(G) = G' \tag{17}$$

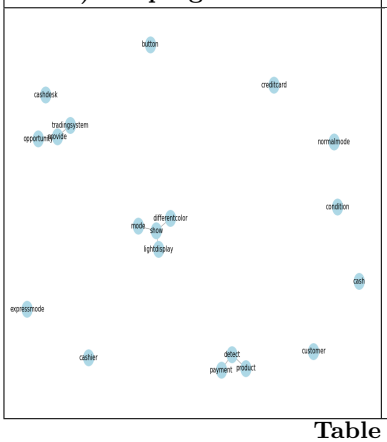
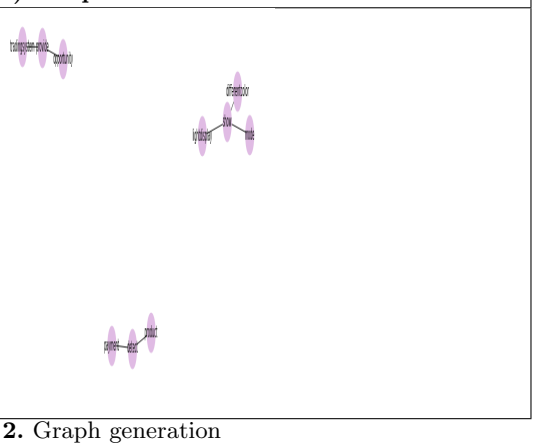
a) Graph generation	b) Graph without isolated nodes
	

Table 2. Graph generation

As can be seen in the table 2 there are only three subgraphs that are not connected to each other, so at first glance three communities can be visualized, it should be emphasized that this is an isolated case due to the number of requirements used for the exemplification.

The application of Louvain's algorithm allows grouping the data based on similar characteristics, identifying communities or groups; the algorithm detects the communities based on the density of connections present in the graph. The graph used is represented in the equation 17.

$$Louvain(G') = C \tag{18}$$

Where $C = \{C_1, C_2, \ldots C_n | 1 \leq i \leq n; \forall i \in \mathbb{N}\}$ represents the finite set of i Communities. Where each community C_n is in turn a set of nodes of the graph G', i.e:

$$Ci = \{n_{i,1}, n_{i,2}, \ldots, n_{i,j}\} \tag{19}$$

Where $n_{i,j}$ represents the j-th node of the i-th community.

The dictionary obtained is shown in the table 3 a) with the node name as "key" and the community as "value". The graphical representation of the dictionary of communities is shown in the table 3 b).

a) Dictionary of Communities

Palabra	# Comunidad
show	0
lightdisplay	0
mode	0
differentcolor	0
product	1
detect	1
payment	1
opportunity	2
tradingsystem	2
provide	2

b) Community Detection

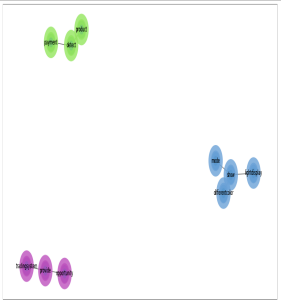


Table 3. Communities

3.4 Microservices

With the communities detected in the network it is possible to assign a microservice name that fits the requirements, i.e., once each value of $C_1, ..., C_n$ is obtained, each element represents a possible microservice name, therefore, the names of these nodes are entered in ChatGPT to obtain a generic name for each microservice.

$$ChatGpt(C) = Micro \tag{20}$$

Where $Micro = \{Micro_1, Micro_2, Micro_n | 1 \leq i \leq n; \forall i \in N\}$ represents the finite set of n Microservices. For this, we use the **openia** library provided by ChatGPT for open source, in which we generated a prompt that specifies the requirements as a context and the dictionary with the communities, thus, the suggested names are: **Payment Microservice, Ligth Display Microservice and Trading System Microservice**.

It should be noted that in order to obtain the most accurate results at the time of assembling the prompt, not only the list of initial requirements but also the list of communities obtained in the process is passed, therefore, ChatGPT has the necessary context to create an accurate answer according to each community found.

3.5 Text Enrichment

A major challenge with the requirements is the fact that the text always omits relevant information. For humans, inferring this information and retrieving the meaning is effortless, but for machines, natural language processing is not an easy task [27]. It is for this reason, that the initial requirements were processed to enrich with more information, for which ChatGPT is used. In addition, ChatGPT allows text enrichment, correction of grammatical errors, suggestion of synonyms or lexical alternatives, generation of summaries and expansion of concepts and texts, among others. Formally, this is described as follows:

$$ChatGPT(R) = R' \tag{21}$$

With the requirements R' enriched, the process described for comparison is run again.

Algorithm 1 provides a logical representation of the process followed to arrive at the solution to the problem posed.

4 Results and Evaluation

Table 4 shows the comparison between the application of the process for two different sets of requirements, the first the initial CoCoMe requirements and the second set enriched with ChatGPT, in terms of initial noun and verb counts, processed nouns and verbs, and the number of communities.

The number of communities depends on the total number of verbs and nouns obtained from each set of requirements. These communities represent groups of words related to the analyzed requirements. Table 5 shows the microservice names obtained from table 4.

The evaluation was carried out through a survey to obtain the opinion and comments of experts and professionals with experience in the development of microservices. The table 6(a) shows the answers obtained from the respondents. It can be seen that the evaluation of the proposed process involved a total of 18 professionals from different areas and with different levels of experience in the development of microservices.

The table 6 shows the percentage of microservices groups chosen by the respondents. It shows that Group A accounts for 45% of the responses, followed by Group B with 40%, and Group C with 15%. It is important to note that Groups A and B are those obtained through the proposed process thus adding up to 85% of preference by the professionals, while Group C is the approach by Tyszberowicz [4].

It is relevant to mention that the surveyed professionals were presented with three groups of microservices to make their choice, without being disclosed which group was generated through the proposed process and which was obtained through the Tyszberowicz [4] approach. In addition, they were provided with the

Algorithm 1: General process for obtaining microservice names through functional requirements

```
Input: R ;                    /* R represents the list of requirements */
Result: Micros ;              /* M represents the list of recommended
        microservices */
S, S_O ← nlp_nouns(R);
V, V_O ← nlp_verbs(R);
S_2 ← set(nlkt(S);
V_2 ← set(nlkt(V);
S_3 ← similar(S_2);
V_3 ← similar(V_2);
M[i, j] ← [];
for lema_v en V_3 do
    list_v ← [];
    for verb en V_O do
        if verb.lema == lema_v then
            list_v.add(verb.lema)
        end
    end
    fila ← [];
    for lema_s en S_3 do
        counter ← 0;
        list_s ← [];
        for noun en S_O do
            if (noun.lema == lema_s and noun.root en list_v) then
                list_s.add(noun.lema);
            end
        end
        counter ← length(list_s);
        fila.add(counter);
    end
    total_conections ← sum(fila);
    if total_conections > 1 then
        M[i, j].add(fila);
    end
end
G = (M[i, j]) ;    /* G represents the graph generated from the Matrix
 M[i, j] */
G' = Delete_Isolated(G);
C = Louvain(G') ;  /* C represents the list of communities obtained
 from G' */
Micros = ChatGPT(C);
```

list of requirements, which is available at github[8], in order to ensure objectivity in their choice of the preferred microservices group.

[8] https://github.com/sebasari11/nlp-microservices

	CoCoMe	ChatGPT
Nouns	94	215
Verbs	57	143
Processed Nouns	53	99
Processed Verbs	38	64
Nro. Communities	7	9

Table 4. Comparison of CoCoMe and ChatGPT communities.

CoCoMe	Enriched with ChatGPT	Tyszberowicz [4]
1. CashDeskService	1. CashDeskService	1. ReportService
2. LightDisplayService	2. TradingSystemService	2. StockOrderService
3. TradingSystemService	3. InventoryService	3. SaleService
4. InventoryService	4. DeliveryTimeService	4. ProductListService
5. DeliveryService	5. PaymentService	
6. PriceChangeService	6. StoreStaffService	
7. QueryService	7. LightDisplayService	
	8. QueryService	
	9. UserInterfaceService	

Table 5. Communities identified

5 Conclusions and Future Work

We have developed a novel and practical approach to identify microservices names out of functional requirements, offering clear and structured phases that guide developers in their identification. The process has been evaluated by professionals and experts in the field, exhibiting a wide acceptance demonstrating its effectiveness and efficiency, with 85% of votes in favor of the results of the process execution. This meaningful finding confirms that the proposed process is a solid and effective guide for the microservices deployment, moreover ensuring consistency in the final result.

For future work, we seek to explore the use of natural language generation techniques using ChatGPT on creating specific prompts to guide the development of methods or functions associated with each identified microservice. In addition, the use of ChatGPT can be further explored to discover new methods, entities, and potential nomenclature in the paths of the detected microservices. These explorations would take advantage of natural language processing to improve the quality and understanding of the generated microservices. Finally, it is suggested to evaluate the effectiveness and adaptability of the proposed process in larger and more complex projects, covering different business domains.

References

1. Chavez, K., Cedillo, P., Espinoza, M. & Saquicela, V. A Systematic Literature Review on Composition of Microservices through the Use of Semantic Annotations: Solutions and Techniques. *2019 International Conference On Information Systems And Computer Science (INCISCOS)*. pp. 311-318 (2019,11), https://ieeexplore.ieee.org/document/9052275/
2. Wang, Y. & Zhang, J. Experiment on automatic functional requirements analysis with the EFRF's semantic cases. 2016 International Conference

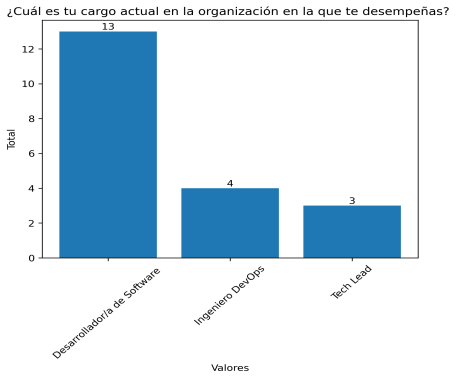
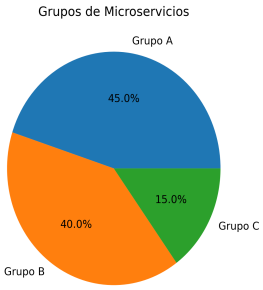
a) Positions/positions occupied	b) Microservices group
	

Table 6. Evaluation

On Progress In Informatics And Computing (PIC). pp. 636-642 (2016,12), http://ieeexplore.ieee.org/document/7949577/

3. Saquicela, V., Campoverde, G., Avila, J. & Fajardo, M. Building microservices for scalability and availability: Step by step, from beginning to end. *New Perspectives In Software Engineering: Proceedings Of The 9th International Conference On Software Process Improvement (CIMPS 2020)*. pp. 169-184 (2021)
4. Tyszberowicz, S., Heinrich, R., Liu, B. & Liu, Z. Identifying Microservices Using Functional Decomposition. *Dependable Software Engineering. Theories, Tools, And Applications*. **10998** pp. 50-65 (2018), http:link.springer.com10.1007978-3-319-99933-3
_4, Series Title: Lecture Notes in Computer Science
5. Laplante, P. & Kassab, M. Requirements Engineering for Software and Systems. (CRC Press,2022,6)
6. Sommerville, I. Software engineering. (Pearson,2011), OCLC: ocn462909026
7. Wiegers, K. & Beatty, J. Software requirements. (Microsoft Press, s division of Microsoft Corporation,2013), OCLC: ocn850176256
8. Nadkarni, P., Ohno-Machado, L. & Chapman, W. Natural language processing: an introduction. *Journal Of The American Medical Informatics Association*. **18**, 544-551 (2011,9), https://academic.oup.com/jamia/article-lookup/doi/10.1136/amiajnl-2011-000464
9. Khurana, D., Koli, A., Khatter, K. & Singh, S. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools And Applications*. **82**, 3713-3744 (2023,1), https://link.springer.com/10.1007/s11042-022-13428-4
10. Bag, S., Kumar, S. & Tiwari, M. An efficient recommendation generation using relevant Jaccard similarity. *Information Sciences*. **483** pp. 53-64 (2019,5), https://linkinghub.elsevier.com/retrieve/pii/S0020025519300325
11. Jaiswal, N. SequenceMatcher in Python. *Medium*. (2021,7), https://towardsdatascience.com/sequencematcher-in-python-6b1e6f3915fc
12. Riaz, F. & Ali, K. Applications of Graph Theory in Computer Science. *2011 Third International Conference On Computational Intelligence, Communication Systems And Networks*. pp. 142-145 (2011,7), http://ieeexplore.ieee.org/document/6005872/

13. Ahmed, M., Seraj, R. & Islam, S. The k-means Algorithm: A Comprehensive Survey and Performance Evaluation. *Electronics*. **9**, 1295 (2020,8), https://www.mdpi.com/2079-9292/9/8/1295
14. Jiawei Han, Micheline Kamber & Jian Pei Data Mining: Concepts and Techniques. *Data Mining: Concepts And Techniques*.
15. Zhang, J., Fei, J., Song, X. & Feng, J. An Improved Louvain Algorithm for Community Detection. *Mathematical Problems In Engineering*. **2021** pp. 1-14 (2021,11,23), https://www.hindawi.com/journals/mpe/2021/1485592/
16. Fortunato, S. Community detection in graphs. *Physics Reports*. **486**, 75-174 (2010), https://www.sciencedirect.com/science/article/pii/S0370157309002841
17. Fortunato, S. & Hric, D. Community detection in networks: A user guide. *Physics Reports*. **659** pp. 1-44 (2016,11), http://arxiv.org/abs/1608.00163
18. Ghosh, S., Halappanavar, M., Tumeo, A., Kalyanaraman, A., Lu, H., Chavarria-Miranda, D., Khan, A. & Gebremedhin, A. Distributed Louvain Algorithm for Graph Community Detection. *2018 IEEE International Parallel And Distributed Processing Symposium (IPDPS)*. pp. 885-895 (2018,5), https://ieeexplore.ieee.org/document/8425242/
19. Blondel, V., Guillaume, J., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *Journal Of Statistical Mechanics: Theory And Experiment*. **2008**, P10008 (2008,10,9), http://arxiv.org/abs/0803.0476
20. Namiot, D. & Sneps-Sneppe, M. On Micro-services Architecture. (2014)
21. Reussner, R., Krogmann, K., Koziolek, H., Rausch, A., Herold, S., Klus, H., Welsch, Y., Hummel, B., Meisinger, M., Pfaller, C. & Others CoCoME-the common component modelling example. *Lecture Notes In Computer Science*. (2007)
22. Vidya Sagar, V. & Abirami, S. Conceptual modeling of natural language functional requirements. *Journal Of Systems And Software*. **88** pp. 25-41 (2014,2), https://linkinghub.elsevier.com/retrieve/pii/S0164121213002379
23. Fraiwan, M. & Khasawneh, N. A Review of ChatGPT Applications in Education, Marketing, Software Engineering, and Healthcare: Benefits, Drawbacks, and Research Directions.
24. Bang, Y., Cahyawijaya, S., Lee, N., Dai, W., Su, D., Wilie, B., Lovenia, H., Ji, Z., Yu, T., Chung, W., Do, Q., Xu, Y. & Fung, P. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. (arXiv,2023,2), http://arxiv.org/abs/2302.04023, arXiv:2302.04023 [cs]
25. White, J., Hays, S., Fu, Q., Spencer-Smith, J. & Schmidt, D. ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design. (arXiv,2023,3), http://arxiv.org/abs/2303.07839, arXiv:2303.07839 [cs]
26. Hardeniya, N., Perkins, J., Chopra, D., Joshi, N. & Mathur, I. Natural language processing: python and NLTK. (Packt Publishing Ltd,2016)
27. Penas, A. & Hovy, E. Semantic Enrichment of Text with Background Knowledge.