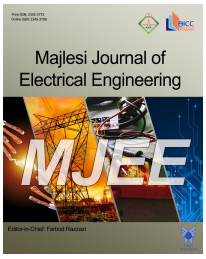

Volume 18, Issue 4, 182450 (1-17)

Majlesi Journal of Electrical Engineering (MJEE)

https://dx.doi.org/10.57647/j.mjee.2024.1804.50

Capability-driven framework to automate discovery of bounded contexts in large-scale requirements engineering

Shohreh Ajoudanian[1,2,*], Maryam Nooraei Abadeh [3]

[1]*Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Iran.*
[2]*Big Data Research Center, Najafabad Branch, Islamic Azad University, Najafabad, Iran.*
[3]*Department of Computer Engineering, Abadan Branch, Islamic Azad University, Abadan, Iran.*

[*]**Corresponding author:** shajoudanian@pco.iaun.ac.ir

Original Research

Received:
15 June 2024
Revised:
20 July 2024
Accepted:
1 August 2024
Published online:
15 December 2024

Abstract:

Large-scale requirement engineering needs automated precise and efficient capability modeling and analyzing methods formally to interoperate with the evolving and goal-driven requirements. The proposed capability-driven requirement engineering framework presents a two-layer framework for the automation of requirements engineering. In the first layer, a meta-model is proposed to define a fault-free model instantiating the requirements model, and thereby ensuring consistency in the process of requirement execution and in the second layer the analysis algorithms are provided for discovering and querying the boundaries and capabilities of the system at the abstract level. The proposed capability-driven requirement framework offers the ability to specify, decompose, and identify the requirement traces to execute the activities regarding available capacities and resources. We also provide the applicability of the approach from various points of view including quality and stability of bounded contexts, average precision, and query assessment. As a running example, we highlight the essential role of electrical features in achieving seamless integration and operation, encompassing power distribution, automation systems, energy efficiency, and safety measures. The proposed capability-driven requirement framework is crucial for effective smart home engineering in this context. The proposed structured, formal description of software requirement capabilities may increase the precision and recall of module discovery mechanisms for large-scale software engineering. An average precision of more than 93% is a significant achievement in the context of information retrieval and evaluation.

Keywords: Requirement engineering; Smart Home; Software capability; Bounded context; Capability querying.

1. Introduction

Requirements Engineering (RE) is divided into five different subprocesses, i. e. , requirements gathering / elicitation, modeling, analysis, verification, and validation, and software requirements management [1], as the first step of any software development process. It is a systematic approach to develop requirements through an iterative process of analyzing a problem, documenting the resulting observations, and verifying the preciseness of the comprehending obtained requirements, and recently evolving according to changes [2–4].

For large-scale systems, early capability extraction, modeling, linking, and discovery are fundamental to separating the concerns while considering functional and non-functional aspects. We define the capability of a system as the functionality or behavior of a program, components of a program, or system using its features. Usually used in a comparative manner, as ”the things a system can do according to its features”.

Various approaches have been introduced to enrich RE by considering non-functional requirements, e.g., goal-oriented RE, capability-driven RE, and feature - oriented RE. Conventional RE methods focus only on “what a system will support” [5–7], but contemporary methods identify goals and transform them through a syntactical structure. RE needs new RE methods that aim to automatically extract and analyze stakeholders’ needs using advanced tools e. g. , natural language processing and information retrieval

processing to create the software according to the system's features.

The capability-driven requirements engineering [4, 8–10] as presented in this work, is a part of RE since an activity happens on the boundary between the developer's technical view and the business view of the stakeholder to find and document requirements of the system in a way to precise analysis, bound and discover its capabilities. The capability is a fundamental concept in domains such as Enterprise Information Systems and Service-Oriented Architecture. The capability concept defines what an action (i. e. , a program, a service, a business process, etc.) may do from a functional perspective, from an abstract class of actions to a very concrete and corresponds to a specific stakeholder request [11]. IEEE has provided a compatible yet intricate definition [12]; a software capability that the user needs to attain an objective or to solve a problem. This term is linked to the following software requirements: quality attributes, functional requirements, design constraints, external interface requirements, and performance requirements.

Analyzing the capabilities of a software and extracting bounded contexts at the requirement level can define the boundaries of the biggest possible functionalities that would not have any conflicting models inside. This separation of concerns at the requirement level provides enough modularity to divide the project between different teams and on parallel teams working especially in agile and fast approaches. If you cross the boundary, those conflicting models will eventually lead to a crosscutting management approach at the abstract level. In addition, in this paper, a running example for representing delivery capabilities is discussed.

In the context of the IEEE Standard Glossary of Software Engineering Terminology, a requirement refers to a condition or capability necessary for a user to solve a problem or accomplish an objective. The proposed approach addresses the challenges related to mining the capability (or capacity) within a large volume of stakeholder requirements focusing on a specific scope or context [13].

Recent research reflects the ongoing evolution and innovation in requirement engineering and mining techniques, driven by advancements in AI, data analytics, and a user-centric approach. Keeping abreast of these trends is crucial for organizations aiming to stay at the forefront of effective software development [14–18].

This paper aims to explore techniques or approaches for extracting and understanding the capabilities or capacities expressed within stakeholder requirements, possibly in a large dataset or extensive set of requirements. By addressing these challenges, developers may provide insights or solutions to better understand and manage stakeholder requirements.

Therefore, the signification of capabilities can be done at numerous abstraction levels from the most abstract (with only one action verb) to the most concrete one, which is the precise need of an end-user, which current capability modeling approaches cannot do. Moreover, we can interlink capabilities to develop a hierarchical structure that enables the refining of the detection process.

The proposed early contexts bounding clarifies, encapsulates, and defines the specific capabilities of the requirement model while ensuring that the various sub-domains will not affect each other. We propose to detect context implicitly defined within stakeholder's requirement, and every context defines boundaries.

In this paper, we use the smart home domain to show numerous parts of our theoretical model. From the service consumer viewpoint, it is much more important to function on consumable, concrete boundaries offers. We show such concrete capability by proposing the output capability format and current functionality descriptions necessitate manual work to transfer to the level of service offer. Our theoretical model intends to automate this step. The contributions of our work are enumerated as follows:

• Early capabilities extraction and analysis in a formal way from user requirement scenarios,
• Calculating the degree of similarity between user scenarios to bounding them
• Exploration software capabilities using the proposed capability querying algorithm
• Presenting a new technique to the bounding of software capabilities as a significant measure in separating the concerns and modularization,
• Adopting changes. e.g., adding new functional modules according to user scenarios.

From this point forward, the paper is arranged in this way: The problem definition of the proposed approach is presented in section 2. Section 3 presents a detailed description of the proposed method, and a running example is presented in Section 4. Section 5 reports approach analysis, and Section 6 presents related work. Finally, we will discuss lessons learned, and risks to validity and will summarize the conclusions and directions for future work.

2. Problem statement by an example

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process [19]. There are various modeling approaches to show system requirements, e.g., Unified Modeling Language (UML), Systems Modeling Language (SysML) requirement diagram [20], Architectural Analysis and Design Language (AADL) [21] and Modeling and Analysis of Real-Time Embedded Systems (MARTE) profiles [22] as general-purpose modeling approaches and Business Objectives Model (BOM) [23] and MetaEdit+ as domain-specific modeling (DSM) approaches. Usually, modeling language provides modeling constructs to represent requirements and relate them to other modeling elements to introduce an actual requirement node, which contains information about requirements such as identifier, text, source, and method of verification.

From the software system specification point of view, such techniques exploit the standard extension mechanisms of UML (i. e. , UML profiling). UML profiles enable software engineers to add non-functional properties to the software model, in addition to the functional ones in various manners, e. g. , model-driven [24], agile [25, 26]. Requirements are usually presented as graphs and tree hierarchies, particularly for the team-based design. As shown in Table 1,

common requirement nodes in a hierarchy structure can be organized to show the dependency between requirements. Each node contains the requirement ID, description of a requirement includes functionality and properties, link between requirements, level of the requirement, and bounded domain. Regardless of the domain, the nodes will signify distinct requirements. Some logical organizations are done by domain, which may characterize requirements from different stakeholders, various requirement scenarios (mechanical, electrical, or physical specifications), or requirements external to the technical area (project staffing, project cost, technical specifications, or project schedule requirements). Today, most modern products and software systems are considered complex systems, and extracting and analyzing their capabilities as soon as possible can lead to productive competitive marketing for software. Capabilities are activities that a software can do using its processes, persons, and technology. In determining the capabilities of software, the focus is on what the software is doing (or should do) regarding the performing quality. On the other hand, specific capabilities for innovation and development of complex and successful systems and systems are necessary for detecting the actions that the stakeholders of your product can take to get value from the product. This is more in line with the use cases you want to solve for each stakeholder. Extracting and mining the capabilities of complex software systems with a diverse range of functional and non-functional units are essential to the domain-independent analysis of software. Despite the importance of capability discovery, current approaches do not model it properly as the early phases of software development to continually handle the ever-growing requirement scenarios in practice. In actual settings, it is fairly difficult to model statically tangible capabilities because concrete level delays in extracting all likely explosions of capabilities, and for software projects, developers cannot disclose the values of sensitive qualities as part of the static function description. Developers need to be involved in a capability property before disclosing the actual value of a sensitive feature.

We provide an example related to smart home product development. In the development of smart home products, electrical engineering features prominently in ensuring the successful integration and functionality of these advanced systems. The expertise of electrical engineers is crucial in designing and implementing the power distribution infrastructure, automation systems, energy-efficient measures, sensor integration, IoT connectivity, and safety and security systems that form the foundation of smart homes. Their contributions enable the seamless operation and interconnectivity of various devices, allowing homeowners to enjoy the convenience, efficiency, and enhanced living experience that smart home technology offers. Customer satisfaction comprises comprehending, defining, assessing, and managing customer's needs to satisfy his/her expectations. Conformance to the requirements is thus necessary to make sure that the project's output is up to the expectation. Defining good requirements has a greater payoff, while not doing so will result in a more intense penalty. In future work, with the help of crowdsourcing, we will improve the requirements gathering phase and the preprocessing, so we go on to improve the quality. In this example, the customer-collected requirements (See Table 2, although seen in this table, there are just a few user requirements, the actual model holds more than 250) are preprocessed with the requirement tokenization and capability tuple retrieval for extracting the capabilities available in the requirements. Each capability is added to the CARG (figure 1. a, c, e, and g) and the CARG is arranged during the context bounding process with the use of algorithm 2 (figure 1. b, d, f and h). figure 1.a shows the extracted capabilities from Req1 in Table 2 that are added to CARG and figure 3.b shows the arranged colored CARG. Each color shows a bounded context. After adding six available requirements in Table 5, as shown in figure 1.h, according to the proposed algorithm (Algorithm 1) seven contexts bounded are discovered. It considers a defined similarity metric to create/update various bounded contexts. The concepts provided in this paper define a solution to answer questions such as:

RQ1. How is it possible to automatically perform early discovery and analysis of software capabilities from user requirement scenarios?

RQ2. How can a developer explore and evolve software capabilities using automatic approaches?

RQ3. Can software capabilities be defined as a significant measure in separating the concerns and modularization?

3. Proposed method

Assuming a broad variety of requirements for large systems, the method proposed suggests the extraction and analysis of system capabilities as a new solution for analyzing these systems. The main objective of the framework is to determine the bounds of the system requirements based on extracted capabilities from the requirement scenarios as soon as possible and with an early analysis. After extraction, the software capabilities of a system in the proposed formal graph, different analyses, and improvements in the graph

Table 1. The information about requirement data and relations.

Requirement fields	Description and usage
I_d	Requirement I_d
Text	Description of a requirement includes functionality and properties
Trace	Link between requirement
Refine	Level of the requirement
Derive	Hierarchy of a requirement
Domains	Initial bounded context

Table 2. Requirements and capabilities Table.

Requirement	Capabilities		Figure no.
	Action	**Properties**	
Req1:Set the smart temperature between a and b, support it in software and hardware, and notify temp via sms.	Set smart temp	between a and b	Figure 1.a
	Support	hardware and software	
	Notify temp	Via sms	
Req2: Set smart light between a and b, support in software and hardware notify light via sms or Email, and ship it in the afternoon.	Set smart light	between a and b	Figure 1.c
	Support	hardware and software	
	Notify light	Via sms or Email	
	Ship	time = afternoon	
Req3: Produce a new toy product for children 3 years old, price between 100 $to 150 $, package in the pink box	Produce a new toy product	children age = 3	Figure 1.e
	Price	between 100 $and 150 $	
	Notify light	Via sms or Email	
	Package	box color = pink	
Req4: Produce a new tool for carpenter above 50 years old with remains turned on notification system via telephone call and deliver it with at least 5 years supplier support and price it between 600 $to 800 $.	Produce a new tool product	carpenter age $\geq$50	
	Notify remains to turn on	via a telephone call	
	Deliver	supplier support $\leq$5	
	Price	between 600 $and 800 $	
Req5.: Set smart windows to close the window when the temperature is lower than a and open the door when the temperature is higher than b and notify closing the window and opening the door via a telephone call.	Set smart window	close the window when the temperature is lower than a and open the door the temperature is higher than b	
	Notify closing the window	via telephone call	
	Notify opening the door	via telephone call	
Req6. Evaluation system to include product name,photo, UPI	Evaluation system	include = product name and photo and UPI	Figure 1.g

can be applied to analyze the boundaries of the system. The boundaries of the system with a significant separation of the concerns enable parallel teamwork and also outsourcing of the system to be used for agile applications.
Figure 2 shows the steps of the proposed framework, which are described in this section in detail. Also, Appendix A lists the symbols used in the paper.

3.1 System input preparation

The system input can be considered as any type of requirement model while we consider the textual one. Named Entity Recognition (NER) [25] will be used to identify the who part in requirement model. NER can identify the names of persons, organizations, locations, etc. As we will discuss in the capability tuple retrieval section, we do not need to who part in the requirement model and we will remove this part.
The elements contained in what and why parts of a requirement are similar. Both contain goal and task elements. The difference is that the what part has a capability element. The contrast between goal, task, and capability lies in the level of detail. In this stage, there is no need to categorize phrases in terms of what or why. This is because, in some cases, the what part can act as the why part [27]. The assignment rules of these parts will be discussed in the "Capability tuple retrieval" section. Identification of what and why parts are carried out by tagging relevant phrases. Relevant phrases are derived from various POS tag patterns from goals, tasks, and capabilities. POS tagging patterns are derived from various sources, for example, POS tag patterns from other related studies or by doing manual tagging. According to the list of POS label patterns, the phrases that are relevant to the software functionality are collected. This includes removing phrases that are not related to the software from the list. This filtering process will be done by comparing it with a dictionary of software functionality. The ability to filter out relevant non-software phrases will also be tested using appropriate machine learning algorithms. This is will be candidates for what and why parts.
In the preprocessing, word tokenization on requirements is

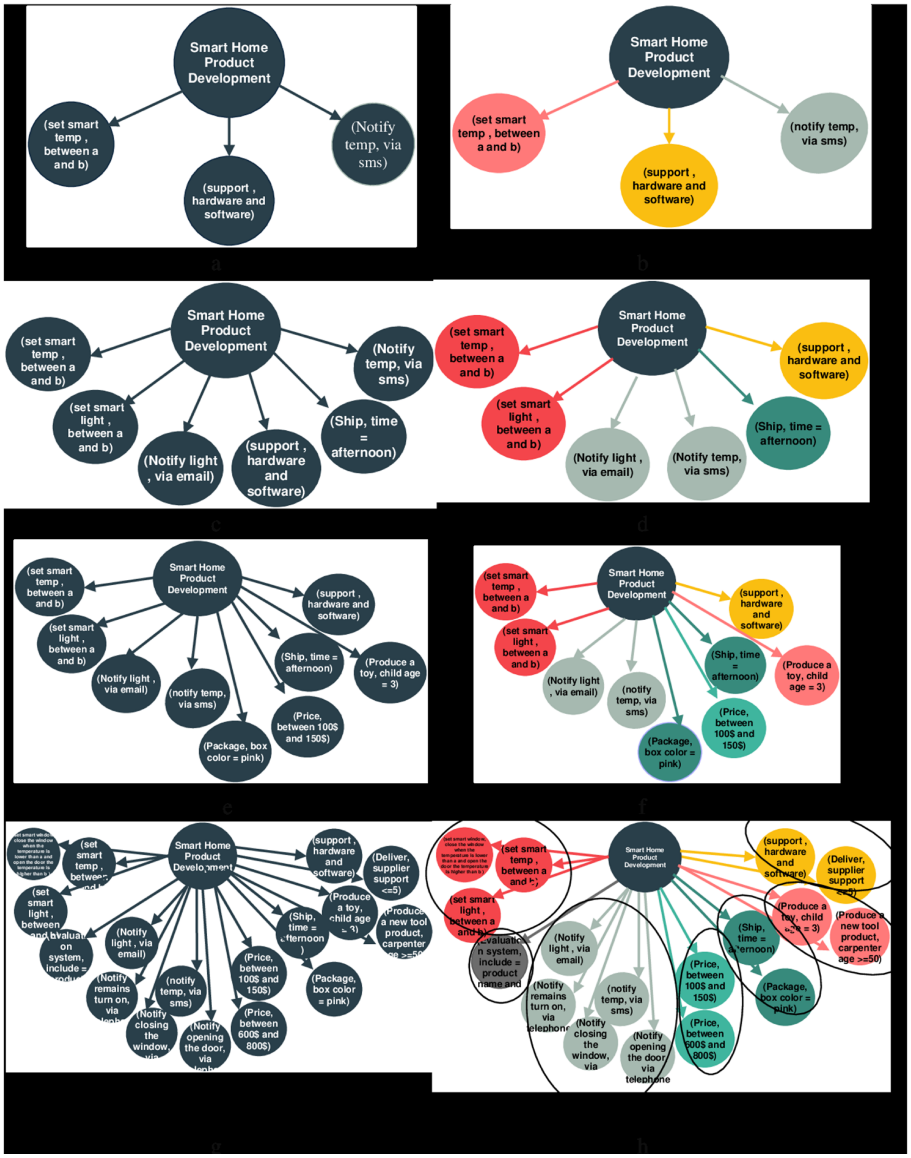

Figure 1. The running example bounding in the step-by-step refinement (Figure 2.a, 2.c, 2.e, and 2.g are the Unbounded graphs and Figure 2.b, 2.d, 3.f, and 2.h are the Bounded capabilities which are shown in the colorful circles according to the context).

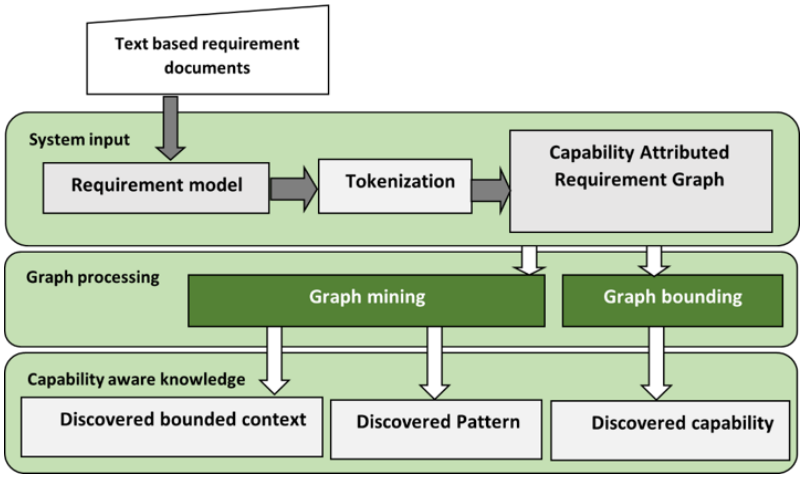

Figure 2. The architecture of the proposed framework

performed and tuples of capability in the form of (Action, Property) are constructed. The capability construction and text extraction process for each requirement is an insignificant matter including the retrieval of verb and noun tokens. For each requirement sentence, to extract property we used all logical operation keywords as defined in the natural language processing dictionary (e. g. , Greater than, Equal to, Greater than or equal to, Less than) and also all synonyms of them by using similar world detection functions the construction of a corresponding capability tuple. Capability tuples will be compiled from phrases (the what/why parts) and properties previously collected. Some of the rules used to identify capability tuples are shown in Table 3.

3.1.1 Tokenization

Without requirement tokenization techniques, a user scenario might appear in the capability format, even though the common unfamiliar users do not know the arguments. Text preprocessing methods are important to provide the necessary process for converting requirement text from natural language to machine analyzable format while the keywords of the capability (action and properties) can be extracted and mined.

The requirement model will go through the pre-processing which includes tokenizer, POS tagging, stop word removal, and WordNet Lemmatization. We perform tokenization based on the method proposed in [26] which specialized using [28]. Before the process is carried out, punctuation, numbers, and special characters in the document are removed. The tokenizer is a process of converting a sequence of characters into a sequence of words. The words are split based on words break exist. Stop words are words that often appear in text that need to be discarded. Lemmatization (WordNet Lemmatization) is the process of grouping the inflected forms of a word that have the same meaning so they can be analyzed as a single item. Lemmatization was chosen because it did not affect the meaning, especially when the POS tagging process was later carried out. The words in the document are marked using Parts of Speech (POS) Tagging. POS tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. The words will be tagged with nouns, verbs, adjectives, and others to find out the sentence pattern.

3.1.2 Capability attributed requirement graph

In this section, a system requirement model transforms into the proposed capability graph, the Capability Attributed Requirement Graph (CARG) while the consistency to its meta-model is preserved. The capability graph is defined based on the attributed requirement graph; thus, each capability graph is an attributed requirement graph that concentrates on existing activities and resources. A Capability Attributed Requirement Graph (CARG) consists of a set of capabilities that each Capability is a set of Actions with their Properties and maybe their constraints. Constraints are a kind of Logical expressions which will appear in the form of pre-conditions and post-conditions. There are some bounded contexts in this graph that are the largest connected semantically bound of a graph. As an example, consider the requirement “Produce a new tool for a carpenter above 50 years old with remains turn on notification system via telephone call and deliver it with at least five years supplier support and price it between 600 $to 800 $”. It can be split into actions (Produce a new tool product, Notify remains to turn on, Deliver, Price) and QoS properties (Carpenter age, via a telephone call, supplier support) and logical expressions ($\geq$,$\leq$, between) between them. In Section 4 a complete case study is described in detail.

Definition 1 (Attributed Graph - AG). An attributed graph is a tuple G = (N, A, E, λ), where N is a set of nodes, A represents an attribute set, E $\subseteq$ N $\times$ N is a set of edges, λ: N $\times$ A $\rightarrow$ R links an actual value to each pair of node attributes as the property value (for functional and non-functional attributes).

Definition 2 (Capability Attributed Requirement Graph - CARG). A Capability-attributed requirement Graph is G = (A, C, E, Cap), in which A is a set of action nodes and the corresponding pre- and post-conditions; C is a connector function that relates with each action node pair a1, a2 a tuple C(a1, a2) = Connector Type(a1, a2): namely, XORjoin, ORjoin, ANDsplit, XORsplit; ANDjoin, and ORsplit; E represents a set of edges for interrelating all the graph nodes; Cap is a function that associates with each action node a tuple Cap(a)=(Action(a), Properties(a)) with each activity node Properties(a) denotes a set of pairs (Property, Value) that signifies the features set of the users or physical resources that links a real value to each node-attribute pair.

Definition 3 (Action). A requirement must perform at least

Table 3. Rules used to identify capability tuples.

Rule	Capability tuple
Ru1e 1: If aspects of who and aspects of what can be identified in the sentence.	Removing the aspect of who part
	Action = aspect of what
	Properties = logical operation tuple
Rule 2: If aspects of what can be identified without aspects of who in the sentence.	Action = aspect of what
	Properties = logical operation tuple
Rule 3: If two or more aspects of what can be identified in the sentence.	Action1 = aspect of what1, Action2 = aspect of what2, ...
	Properties1 = logical operation tuple1, Properties2 = logical operation tuple2, ...

an action that operationalizes the requirement according to available capacity. Act(Cap(a)) is a function that gets as input the capability of node a Cap(a) and returns as output Action(a) of the Cap(a).

As two main parts of the proposed approach, we investigate the CARG, to solve two problems (Querying, and context bounding) which enables a better understanding of defined capabilities or filtering the capabilities with common properties.

3.2 CARG processing

In this section, we define two problems and their solutions in capability-driven requirement engineering including graph querying and bounded context creating and updating. Considering a capability set CapSet and a new capability requested for a requirement, the degree of similarity between them is calculated based on the similarity of their attributes.

3.2.1 Similarity measurement

Many metrics have been developed to quantify the strength of connections (or relationships) between two words. Pointwise mutual information (PMI) [29] is a common measure. PMI is a measure of the difference between the actual co-occurrence frequency of two words and the expected co-occurrence frequency of the words assuming independence. Positive pointwise mutual information (PPMI) is a variant of PMI that yields the PMI score if the score is more than zero and zero otherwise. The best co-occurrence-based measure has been reported to be PPMI [30]. The definition of PPMI is as follows:

$$P_d(Cap_i.x) = \frac{f_d(Cap_i.x)}{\sum_{i=1}^{n} f_d(Cap_i.x)} \tag{1}$$

$$P_d(Cap_i.x, Cap_j.x) = \frac{f_d(Cap_i.x, Cap_j.x)}{\sum_{i=1}^{n} f_d(Cap_i.x)} \tag{2}$$

$$PMI_d(Cap_i.x, Cap_j.x) = \log \frac{P_d(Cap_i.x, Cap_j.x)}{P_d(Cap_i.x)P_d(Cap_j.x)} \tag{3}$$

$$PPMI_d(Cap_i.x, Cap_i.x) = \begin{cases} PMI_d(Cap_i.x)PMI_d(Cap_j.x) > 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{4}$$

We customize this metric to measure relationships between two capabilities. P_d (Cap_i.x) is the probability of occurring the capability Cap's attribute and/or action, called Cap_i.x, in a requirement document d in the preceding equations. f_d (Cap_i.x), the frequency of Cap.x occurring in the requirement document d) divided by the total number of words in the all requirements can be used to estimate this likelihood. The number of unique words in the requirement documents is denoted by the letter n. The chance of two capabilities Cap_i and Cap_j appearing together in a sliding window in a document in the corpus is given by P_d (Cap_i.x,Cap_j.x). Given a document d, PMI_d (Cap_i.x,Cap_j.x) and $PPMI_d$ (Cap_i.x,Cap_j.x) are pointwise mutual information and positive pointwise mutual information between Cap_i and Cap_j. Then we use WordSimSE [31], to measure the word similarity between various requirement documents. WordSimSE express two words as vectors and then compute the similarity between these two vectors to compute their similarity. Each word is represented as a feature vector, with each element representing the weight of that word's co-occurrence with other (contextual) words in the corpus. Our weighted positive pointwise mutual information (WPPMI) is used to calculate the co-occurrence weight, which is calculated using Eq. 5.

$$WPPMI_d(Cap_i.x, Cap_j.x) = W(Cap_g.x) \times PMI_d(Cap_i.x, Cap_j.x) \tag{5}$$

where:

$$W(Cap_j.x) = \begin{cases} \alpha & (\text{if } j \text{ is a popular software tag}) \\ \beta & (\text{if } j \text{ is a nonpopular software tag}) \\ \gamma & (\text{otherwise}) \end{cases} \tag{6}$$

If the elements of capabilities Cap_i and Cap_j do not appear together, the value of $WPPMI_d$ (Cap_i.x,Cap_j.x) is 0. W(ap_j.x) is a weight parameter that is used to regulate the contributions of various sorts of semantic anchor words. Popular software tags, other software tags, and other terms influenced by [31] are the three types of words we consider. Software tags are generally software-specific keywords, and as a result, they should be given more weight. Next, we compute the cosine similarity of their sample vectors to determine how similar two phrases are. WordSim^{SE} [31] is the abbreviation for the ultimate similarity score, which is defined as:

$$Wordsim^{SE}(Cap_i.xCap_i.x) = \frac{\sum_{k=1}^{n} WPPMI_d(Cap_i.x, Cap_k.x)WPPMI_d(Cap_j.x, Cap_k.x)}{\sqrt{\sum_{k=1}^{n} WPPMI_d(Cap_i.x, Cap_k.x)^2}} \times \frac{1}{\sqrt{\sum_{k=1}^{n} WPPMI_d(Cap_j.x, Cap_k.x)^2}} \tag{7}$$

The total number of unique words in the corpus is denoted by n in the preceding equation. Using this metric, it is possible to find similar capabilities of a capability or the bound which a new requested capability is belong to. This is dependent completely on the attributes and related values that outline every capability.

Problem 1 (Graph Bounding)

The goal of graph bounding is to find bounded context (Definition 6) at the requirement level to define the boundaries of the biggest possible functionalities that would satisfy the corresponding stakeholders. The approach for graph querying is shown in Algorithm 2.

Definition 6 (Bounded context). A bounded context is a bound of G that *meets structure cohesiveness* (namely, the nodes included in a bounded context are somehow connected). Based on a common concept of structure cohesiveness, the least degree of all the nodes that show up in the bounded context should be equal to or higher than k.

Algorithm 1 for a capability set CapSet and a new capability Cap 1 finds similar capabilities or bounds to which Cap1 can belong. This is dependent completely on the attributes and related values that outline every capability.

Algorithm 1
Input: Graph CARG, Capability C1 with Action A1
Output: Capabilities OrderList: the set of discovered capabilities

```
1.Begin
2.Candidates of the Capabilities (Cands), Neighbors (Nei);
3.Integer i;
4.Cands = RamdomSelection(n, CARG.E);
5.OrderedList = Order(Cands);
6.while (i ≤ it and Wordsi(OrderedList(0), C1) ≤ 1) do
7. i= i+ 1;
8. for each (CN ∈ Cands and ¬ CN.Visited) do
9. CN.Visited = True;
10. Nei = getNei(CN, CARG);
11. foreach (NC ∈ Nei) do
12.if SimScore (CN, C1) then
13. Cands.Add(NC);
14. OrderedList = Order(Cands);
15. end
16. end
17. end
18. end
19. return OrderedList;
20. end
```

Problem 2 (Bounded Context Discovery)
The aim of another class of solutions is to find a bounded context according to a capability query request. Given a CARG, G = (A, C, E, Cap, Cond), a capability request C1∈ A, returns the largest bound similar to C 1. This definition and corresponding algorithm are inspired by research on the community detection problem. The proposed method must first define *K* partitions as bounded contexts. Because the distances between the first *K* capabilities are greater than Eps, the algorithm considers them to be the centers of the *K*-bounded contexts. The technique progressively divides the requirement documents for subsequent capabilities to avoid the scalability problem of scanning the whole search space for the newly inserted capability's neighborhood. The method creates/updates bounded contexts in this partition based on the incremental clustering algorithm [32] after allocating the new capability to its nearest partition. The method then combines the dense bounded contexts of various clusters gradually depending on a specified similarity measure to create/update the final bounded contexts. The suggested technique for constructing and updating bounded contexts is described in Algorithm 1 inspired by [33].
In bounded contexts, the incremental DBSCAN method [32] is inspired to update clusters of capabilities. A dense area is a collection of capabilities that may either constitute a final bounded context or be a portion of one. The incremental insertion module of DBSCAN (incAdd) is used to locate a bounded context at the closest partition that a capability may join. The deletion module (incDel) is used to remove old capabilities from their old dense regions.
According to [32], the threshold value is considered to merge bounds with the shortest distance in the current clustering until all similar capabilities are included in a single cluster. In the analysis section, the performance algorithms are evaluated from various viewpoints including the quality and stability of bounded contexts.

4. Analysis and results

In this section, we analyze our approach from four points of view including quality and stability of bounded contexts, average precision, and query assessment.
In the first and the second problems, the goal is to find an ordered list of capabilities similar to the query capability of a user, a bounded context according to a capability query request, and in the third problem, the graph is context-bounded. In all three problems, the quality of capability modeling is the main issue. For this purpose, the CARG effectiveness metric is introduced. The results are acceptable. The challenge we face in computing this metric is finding the right threshold value. Another experiment is on the time cost of discovering bounds. This time was acceptable on our domains with a large number of requirements and capabilities. Another issue that was important for us is despite the many changes at the system development time, how the changes can affect on discovering the right bounded context.
As the test data, by crawling GitHub, we were able to collect 3,925 requirement scenarios (in text format) and then prepared 1, 105 requirement documents that were chosen based on the domain similarity to construct the capability graph. These requirements are in four category domains: Online collaboration system with 201 requirements and 16 discovered bounds, Smart home online store with 254 requirements and 21 discovered bounds, open-source search engine system with 350 requirements and 14 discovered bounds, and Smart grid software solutions with 300 requirements and 24 discovered bounds. With an Online collaboration system, we mean the systems enable users to

Algorithm 2
Input: Graph CARG, new Capability NewCap, Bound_ centers
Output: Create/Update the bound after insertions of NewCap

1. Begin
2: CapSet ← List of bounded capabilities that may change their centers
3: D ← List of updated dense regions
4: For each NewCap do
5: c← nearest_ centers_ Wordsim^{SE} (pi, bound_centers)
6: CapSet ← update_centers (NewCap, c)
7: Add NewCap to CapSet
8: end for
9: For each ri in CapSet do
10: cn ← ri.new_center
11: co ri.old_center
12: Apply incDel to remove ri from co
13: Apply incAdd to insert ri to cn
14: Add updated dense regions to D (step 11 and 12)
15: end for
16: For each di in D do
17: For each dj in D and i ≠ j do
18: If inter_connectivty (di,dj) > threshold
19: merge(di,dj)
20: end if
21: end for
22: end for
23. End

work together openly. Especially, users may assess, share, make artifacts, network, and perform tasks. Users can share "items" through these systems, such as structured or textual knowledge, services, and products.

4.1 Bounded context assessment

In this section, we assess our bounded context from two points of view: quality and stability of bounded contexts. Measuring bounded context quality is an important issue because our method was unsupervised. There are no commonly recognized best suitable measures in practice. Because there is no expert-specified knowledge about bounds, internal clustering quality identification measures are used in our approach. In these measures, the goodness of a bounded context is evaluated by considering how well the bounds are separated and how compact the bounds are. One of the most important internal cluster quality measures is the silhouette coefficient [34]. For a data set, D, of n objects, suppose D is partitioned into k bounded context, $B_1, \ldots,$ B_k. For each object o $\in D$, we calculate a (o) as the average distance between o and all other objects in the bound to which o belongs (Eq. 8). Similarly, b(o) is the minimum average distance from o to all bounded context to which o does not belong (Eq. 9). Formally, suppose $o \in C_i$ ($1 \leq i \leq k$); then

$$\alpha(o) = \frac{\Sigma_{o' \in C_i, o \neq o'} dist(o, o')}{|C_i| - 1} \tag{8}$$

and

$$b(o) = min_{C_{j:1 \leq j \leq k, j \neq i}} \left\{ \frac{\Sigma_{o' \in C_j} dist(o, o')}{|C_j|} \right\} \tag{9}$$

The silhouette coefficient of o is then defined as Eq. 10.

$$s(o) = \frac{b(o) - a(o)}{max\{a(o), b(o)\}} \tag{10}$$

The value of the silhouette coefficient is between -1 and 1. The value of *a(o)* reflects the compactness of the bounded context to which o belongs. The smaller the value, the more compact the bound. The value of *b(o)* captures the degree

Table 4. The results of bounded context assessment.

	No. of requirement	s(o)	% of correctness of the number of found bounds	No. of found (ms)	Time cost
Online collaboration system	201	0.90	99	16	5
Smart home online shop	254	0.89	97.4	21	8
large enterprise	350	0.78	90	14	4
Smart grid software solutions	300	0.95	97	24	9

to which o is separated from other bounds. The larger *b(o)* is, the more separated *o* is from other bounds. Therefore, when the silhouette coefficient value of o approaches 1, the bound containing o is compact and o is far away from other bounds, which is the preferable case. The silhouette coefficient of four experiment domains is calculated and the results shown in Table 4 indicate that the compactness and separation of the bounds are acceptable.

Another issue is bounded context assessing is bounded context stability. Bounded context obtained from several datasets sampled from the same underlying distribution as *D* should be similar or stable. In this measure, the best number of bounds is judged. In this paper, a bootstrapping approach is used [35]. This approach involves the generation of several "fake" data sets by sampling patterns with replacement in E (bootstrapping). For each number, *K*, of clusters, a measure of stability of the K-cluster partitions over the bootstrap samples is used to characterize the significance of the K-cluster partition for the original data set. The value of K which provides the most stable partitions is the estimate of the number of clusters in E. The bounded context stability of four experiment domains is calculated and the percentage of correctness of several found bounds and number of found bounds is shown in Table 4 which indicates our bounding context is stable.

As another experiment, we calculate the time of bound discovery (figure 3 and 4). In this experiment, we wanted to show the time cost of the proposed approach in discovering bounds with a large number of requirements and certainly a large number of capabilities. The results are quite acceptable as Table 4.

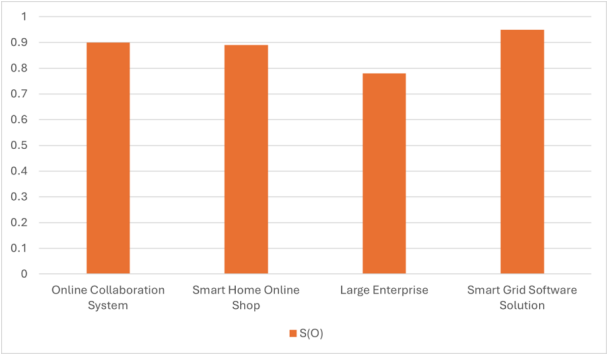

Figure 3. The results for bounded context assessment in terms of s(o).

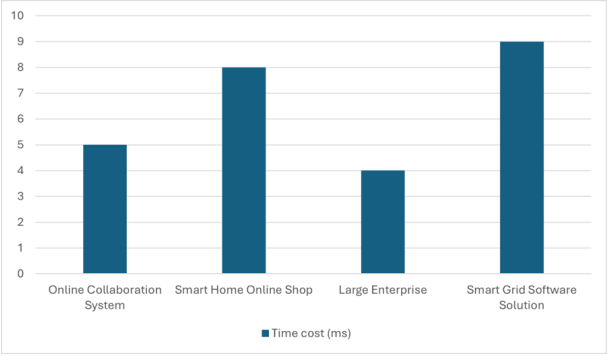

Figure 4. The results for bounded context assessment in terms of time (ms).

4.2 Average precision (AP) of the approach

This metric aims to assess the precision of the suggested capability retrieval method. We adopted the word expansion method proposed in [36] to include capabilities whose similarities to the original capability are larger than a given threshold (0.9). Then, to confirm the performance of capability retrieval (defined in Eq. 11), we applied the average Top-5 precision indicator. The value of K was set from 1 to 5 in this work.

$$\text{Precision}^{k}(R_i) = \frac{|Rel(R_i) \cap Rank^{k}(R_i)|}{|Rank_k(R_i)|} \qquad (11)$$

where R_i is the ith user requirement, $Rank^K\ (R_i)$ is the Top-K bounds returned (the 1st returned K bounds), and $Rel(R_i)$ is a set of relevant requirements with R_i (i.e., answers to the ith user requirement). We compute the average of Top-5 discovered bounds precision for 20 selected requirements for each case study which results are shown in figure 5. According to the results, the Top-1 Top-5 precisions are decreasing while the larger bound of similarity is selected. It is possible to detect the best bounds of a requirement using this metric. For the large enterprise system, the Top-4 and Top-5 are the same while for others by increasing K, the Top-K bounds decrease, so provide lower precision in bounded concerns.

The problems related to retrieving distinct sub-domains from a requirement model can generally be classified into context detection and context search. The solution enables an automatic approach to how and why a context is formed. Our results show that the approach is effective and efficient in retrieving the contexts from requirement models in various domains. More tightly-connected vertices with similar contexts or backgrounds show a better bounderization. So, developers can focus on the particular capabilities or activities in a domain in the evolution and maintenance phases.

4.3 Queries assessment

A detailed time analysis for the proposed method described in the provided text requires access to specific implementation details and algorithmic complexities of the referred techniques in Table 5. Since the specific complexities are not mentioned, it is not possible to provide a precise time complexity analysis.

However, it is possible to provide a general overview of the time complexity considerations for the main steps described:

• Defining K partitions: The time complexity of this step depends on the number of desired partitions (K) and the size of the input data. Typically, this step involves iterating over the data once and assigning each capability to a partition, resulting in a time complexity of O(N), where N is the number of capabilities.

• Identifying centers of bounded contexts: This step involves selecting the first K capabilities as the centers of the bounded contexts. As it requires iterating over the first K

Table 5. The results for queries assessment.

	No. of queries	Avg. Time cost (ms)
Online collaboration system	39	2.9
Smart home online shop	53	3.4
large enterprise	88	7.1
Smart grid software solutions	107	12.4

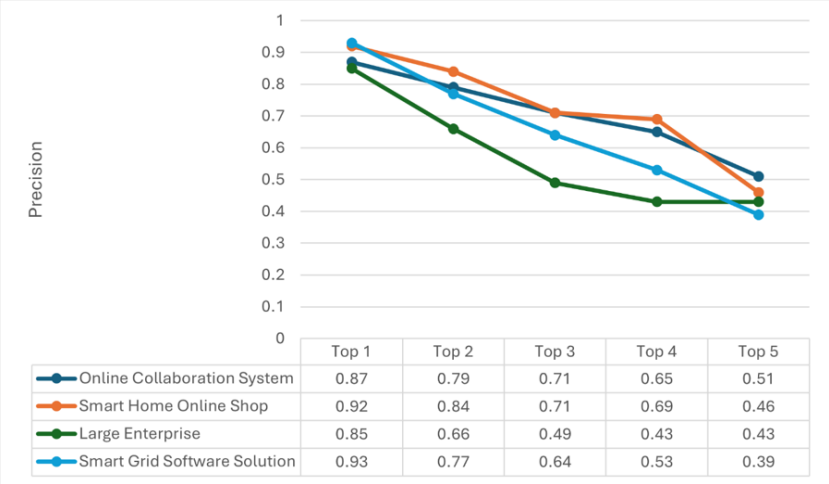

Figure 5. The average precision of Top-5 discovered bounds.

capabilities, the time complexity is O(K).
• Progressive division of requirement documents: The time complexity of this step depends on the number of subsequent capabilities and the size of the requirement documents. If the division involves scanning the entire requirement documents for each subsequent capability, the time complexity could be O(M * N), where M is the number of subsequent capabilities and N is the size of the requirement documents.
• Creating/Updating bounded contexts: The time complexity of this step depends on the implementation details of the incremental clustering algorithm and the number of capabilities being allocated to their nearest partitions. The complexity will vary based on the specific algorithm used.
• Combining dense bounded contexts: The time complexity of this step depends on the number of dense bounded contexts and the similarity measure used. If merging involves comparing all pairs of dense contexts, the time complexity could be $O(D^2)$, where D is the number of dense bounded contexts.
• Incremental DBSCAN for updating clusters: The time complexity of this step depends on the implementation details of the incremental DBSCAN algorithm and the number of capabilities being added or removed. The complexity will vary based on the specific algorithm used.
It's important to note that the specific complexities and efficiency of the incremental clustering and DBSCAN algorithms are critical factors in determining the overall time complexity of the proposed method.

4.4 Discussion

The proposed framework, in response to RQ1, takes a set of user requirements as input, builds the corresponding attributed capability configurable graph conformed to the proposed meta-model, and, in response to RQ2, outputs a capability aware domain for querying and discovering (using Algorithm 1), as well as bounding and updating software capabilities (using Algorithm 2) at the early stage of the development process. The resultant capabilities (in response to RQ3) can be utilized during the RE phase to modularize requirements for a variety of purposes, such as outsourcing software projects to a parallel team.
Extracting and mining the capabilities of complex software systems with a broad variety of functional and non-functional elements is the major finding of this case study. Despite the significance of capability discovery, the suggested technique models it appropriately in the early stages of software development in order to continuously manage the ever-growing requirement contexts in practice.
Various threshold values are taken into account to merge bounds with the shortest distance in the present bounding until all comparable capabilities are grouped together in a single bound. Choosing the optimal value for merging to bound is a practical experiment based on domain features in assessing bound separation and appropriate distance between any two bounds, or distance between bound centers. These measure quite a well-defined concept of separation. There are also different software entity similarity criteria that may be used in clustering evaluation, as well as an overview of suitable clustering algorithms and clustering methodologies. Later in the literature, several deterministic and intelligence-based techniques to solving the re-modularization problem as a clustering problem were presented and will be addressed in future studies. Finally, the limitations of this research are discussed in the different

mapping study steps.

Threats to selection and data extraction consistency. In this paper, as the test data, by crawling GitHub, we collect requirement scenarios, and then prepare 1105 requirement documents selected based on the domain similarity to construct the capability graph. However, research has shown that volunteers do not have the same characteristics as the general population (e. g. , [37]). As the first threats to external validity, the real requirements that a user enters in a particular domain may have little impact on the results because of the frequency of focus on the particular context and so on.

Threats to the quality of the used tokenization algorithm. The quality of the requirement tokenization and capability tuple retrieval algorithm can affect the results and lead to more precise synthesis and results. Our method calculates employs merely the lexical expansion with domain-specific semantic expansion. Choosing the right domain can also influence the results of the algorithm.

Threats due to immaturity of work and lack of them in this field. Being new in the field makes the analysis harder and limits comparability between different analysis results.

Threats to set a similarity threshold for different domains. Setting a similarity threshold for different domains is a threatening work. This measure enables determining the degree of similarity between two capabilities (regarding them as structured entities).

5. Related works

RE needs to precisely determine "WHAT" the system should do and also how to do it. Various methods are used for different RE processes through the system goals, for example, knowledge acquisition for automated specifications (KAOS) method [38], GOIG method, NFR framework, i* framework, AGORA method [39], and some other methods e.g. the pattern-based method [40], hybrid methods [41], [42], [43] and the differential method [44]. AND/OR graph is employed in GORE to model functional and non-functional requirements of the software. Several AND/OR graphs have been created in the GORE. Kaiya et al. [4]suggested the AGORA goal graph, for the subsequent activities: "(i) establishing initial goals as customer's needs, (ii) decomposing and refining goals into sub-goals (iii) choosing and adopting goals from the alternatives of the decomposed goals, (iv) detecting and resolving conflicts on goals". To model the non-functional and functional goals, Mohammad et al. [40] developed a FAGOSRA graph . Fuzzy contribution values and fuzzy preference matrices in this graph are linked to the graph nodes. The softgoal interdependency graph was made to model the non-functional requirements. Kaiya et al. [39] found that goal-oriented methods such as GRL, i*, and KAOS are the top-down approaches for decomposing and refining the stakeholders' needs by satisfying the customer's needs. The AND/OR graph was the resultant structure. The developers indicated that current goal-oriented methods do not support these activities: "(i) selecting the goals to be decomposed (ii) prioritizing and solving the conflict of goals and the conflict of stakeholders on a goal (iii) choosing and adopting a goal out of the alternatives of the goals as a requirements specifications (iv) analyzing the impact when requirements change, and (v) improving the quality of the artifacts developed by the method". Sadiq and Jain provide methods, such as AGORA, i*, GRL, and KAOS to be used for the choosing and prioritization of the software requirements in fuzzy settings [45].

The suggested technique in [46] uses the textual semantics of software functional requirements (FRs) to infer probable quality restrictions enforced in the system. Specifically, we perform a systematic study of a variety of word similarity approaches and clustering techniques to produce semantically coherent clusters of FR terms.

Natural Language Processing Requirements Engineering (NLP4RE) applies NLP techniques, tools, and resources to different requirements documents in a wide range of research. Different linguistic processing tasks like detecting language problems, defining core domain principles, and creating ties between specifications in terms of traceability occur at different RE stages. As a survey and systematic study [47] in this field, the authors provide systematic search results as follows:

NLP4RE has amassed a large number of publications and attracted widespread attention from diverse communities having only been evaluated using an experiment or an example application.

According to the information provided in [47], The selected studies also suggested 130 new methods to enable a variety of linguistic research activities, 140 NLP strategies (e.g. POS tagging and tokenization), 66 NLP methods (e. g. Stanford CoreNLP and GATE), and 25 NLP instruments (WordNet and British National Corpus) from the reviewed articles.

However, most novel NLP strategies and tools are not commonly used; NLP technologies, on the other hand, are frequently used as general-purpose software analysis techniques. In this paper, we propose using NLP as a tool to automate the requirement analysis phase in software engineering.

Some mining methods have been designed to analyze the software requirements, e. g. , [48], [49], [50] and [51] as a recent survey in this area. In [48], a design theory based on two design principles is proposed for systems of requirement mining: (1) using retrieved and imported knowledge, and (2) automatic requirement mining. This theory enables requirements engineers to find and organize requirements in natural language, and to assess the viability of the artifact and the conceptual reliability of the proposed design. Natural language processing methods were used in [49] for automatic generation of developing Parametric Property-Based Requirements from semi-structured and unstructured specifications. A mechanical ring was designed through our approach. the A technique was proposed in [50] to build a body of different kinds of incoherence as well as a categorization for defining patterns to mine incoherent requirements.

There are some works in extracting domain models [52], [53] or software features [54] from Natural-Language requirements. In [55], authors presented an automated ap-

proach based on Natural Language Processing for extracting domain models from unrestricted requirements. The main technical contribution of their approach is in extending the existing set of model extraction rules. Authors in [56] conducted a systematic literature review for feature extraction approaches from NL requirements for reuse in SPLE.

In their study [57], the authors have rigorously explored a range of architectural scenarios on database transaction failures and resilience within the realm of microservices at an enterprise scale. Their research specifically zeroes in on the resilience of transactions in microservices across defined bounded contexts.

In the paper referenced as [58], the authors introduce a structured method for Software Architecture Reconstruction (SAR) specifically tailored to software systems built using Microservice Architecture (MSA). This approach utilizes a suite of modeling languages designed for model-driven MSA engineering, enabling the precise capture of architectural details within models tailored to specific viewpoints. As a result, the application of this method results in the creation of detailed reconstructions including models of the domain, technology stack, services, and operations specific to the microservice architectures under analysis. Further, they meticulously analyzed a curated set of input files to extract domain concepts. These identified domain concepts, encapsulated within bounded contexts, are then documented within the reconstruction domain models using a bespoke Domain Data Modeling Language.

In [27], the researchers have proposed a methodology that analyzes an application's source code to generate recommendations for structuring microservices. This proposed technique involves the extraction of specific keywords succeeded by a Breadth First Search traversal guided by predefined rules. By employing this structured approach, the methodology effectively decomposes a monolithic application into multiple clusters, wherein the predominant number of components within each cluster share a common business domain.

The method introduced in [59] takes an automated stance in segmenting microservices, leveraging graph clustering paired with combinatorial optimization strategies to strike an optimal balance between maximizing internal coherence and minimizing external dependencies.

In [60], the authors have detailed an innovative metamodeling strategy, named the Software Pattern MetaModel (SoPaMM). This approach is designed to synchronize the requirements specification with the testing phase by employing requirement patterns alongside test patterns. The primary ambition of the authors is to identify recurring motifs within system requirements that can be cataloged and repurposed in subsequent projects.

The capability definition, discovering, and managing are important various domains for business processes, enterprise information systems as well as service-oriented architecture e. g. , [55], [61], [56], and [13]. While, according to the best knowledge of the author, there is no similar work that tackles the concept of capability as a first-class entity of software functionally in a domain-independent RE approach. Also, the approaches to explicitly discovering, bounding querying the software capabilities at the requirement engineering phase via analyzing informal user requirements.

6. Future direction

The paper proposes a framework that in response to RQ1 takes as input a set of user requirements, builds the corresponding attributed capability configurable graph conformed with the proposed meta-model, and in response to RQ2 outputs a capability-aware domain for querying and discovering, (using Algorithm 1), and bounding and updating the software capabilities (using Algorithm 2) at the early phase of the development process. The resulting capabilities (in response to RQ3) can be used at the RE phase to modular the requirements for various goals e.g., outsourcing the software projects as the parallel team working. The framework has been evaluated using real-world software scenarios that have been manually collected and annotated within GitHub. The evaluation was done considering three main dimensions: precision, time, and cohesiveness rate of discovered bounds before and after changes. Results show that the approach is promising to reach a certain level of maturity to facilitate the RE in software settings. As future work, we propose the following extensions: planning to examine more relations that may be valuable for the capability descriptions graph, planning to provide the required environment for tracing and transformation of the graph to analysis and design phases, and establishing a maintenance mechanism for the trained model to automatically extract the domains of requirements with enough accuracy and precision. Finally, we will extend the approach to define various semantic relations among capabilities. Capability holders may quickly and simply describe new capabilities using these relations by recycling earlier definitions.

7. Conclusion

An original meta-model was presented in this paper for defining capabilities. This model has many advantages; it has functional and business characteristics, which are specified in consumers' requests and typical desires. The meta-model may handle capabilities at different levels of abstraction in the same manner. Furthermore, this model builds relationships among capabilities at different levels of abstraction. Above all, the model offers the needed declarative specification to vigorously create concrete capabilities. Also, we discussed the idea of early integration of capabilities in requirement engineering to be used for creating a capability-driven development process. The considered method for requirement analysis in this paper applies tokenization based on the method proposed which keeps nouns, verbs, and logical conditions, and removes stop words to filter out the capability records in an output file according to the conditions described in the user requirement in any format which may be defined as feedback, user's mails, or Twitter, or sensors, or even from user stories.

8. APPENDIX A

Table 6 shows the used symbols in the paper and the corresponding meanings.

Table 6. Symbols and meaning.

Symbol	Meaning
G = (N, A, E, λ)	Definition 1
A	Set of actions
C	Connector function **input:** nodes a1, a2 **output:** ANDsplit, ANDjoin, ORsplit, ORjoin, XORsplit, and XORjoin
E	Set of edges
Cap	Capability function **Input:** node a **Output:** (Action(a), Properties(a))
Cond	Condition function **Input:** node a **Output:** a set of pre and post conditions of node a
Act(Cap(a)) Prop(Cap(a))	Action function **Input:** capability of node a Cap(a) **Output:** Action(a) of the Cap(a) Property function **Input:** capability of node a Cap(a) **Output:** Properties(a) of the Cap(a)
ActionSet	All actions in a CARG
PropertySet	All properties in a CARG
CapSet=(ActionSet, PropertySet)	The set of tuples (ActionSet, PropertySet)
Wordsim ^{SE}	similarity function

Authors contributions
All authors have contributed equally to prepare the paper.

Availability of data and materials
The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflict of interests
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] P. Loucopoulos and V. Karakostas. "**System requirements engineering**.". *McGraw-Hill, Inc.*, 1995. DOI: https://doi.org/10.1049/ic:20050-131.

[2] G. Kotonya and I. Sommerville. "**Requirements engineering: processes and techniques**.". *Wiley Publishing*, 1998.

[3] I. Sommerville. "**Software engineering,9th Edition**.". *Mater. Today*, 10(42), 2007. DOI: https://doi.org/ISBN-10137035152, 2011.

[4] P. Loucopoulos and E. Kavakli. "**Enterprise modelling and the teleological approach to requirements engineering**.". *International Journal of Cooperative Information Systems*, 4(01):pp. 45–79, 1995. DOI: https://doi.org/10.1142/s0218843095000032.

[5] A. Van Lamsweerde. "**Requirements engineering: From system goals to UML models to software**.". *Chichester, UK: John Wiley and Sons*, 2009. DOI: https://doi.org/10.1109/icse.2003.1201266.

[6] J. Siddiqi. " **Requirement engineering: The emerging wisdom**.". *IEEE Software*, (2):pp. 15, 1996. DOI: https://doi.org/10.1109/ms.1996.506458.

[7] M. N. Abadeh. "**Knowledge-enhanced software refinement: leveraging reinforcement learning for search-based quality engineering.**.". *Automated Software Engineering*, 31(2):pp. 57, 2024. DOI: https://doi.org/10.1007/s10515-024-00456-7.

[8] S. Berzisa, G. Bravos, T. C. Gonzalez, U. Czubayko, S. Espana, J. Grabis, M. Henkel, L. Jokste, J. Kampars, H. Koc, and J. C. Kuhr. "**Capability driven development: an approach to designing digital enterprises**.". *Business and Information Systems Engineering*, 57(1):pp. 15–25, 2015. DOI: https://doi.org/10.1007/978-3-319-90424-5-13.

[9] M. H. Danesh, P. Loucopoulos, and E. Yu. "**Dynamic capabilities for sustainable enterprise IT–a modeling framework**. ". *International Conference on Conceptual Modeling, Springer*, pages pp. 358–366, 2015. DOI: https://doi.org/10.1007/978-3-319-25264-3-26.

[10] P. Loucopoulo, C. Stratigaki, M. H. Danesh, G. Bravos, D. Anagnostopoulos, and G. Dimitrakopoulos. "**Enterprise capability modeling: concepts, method, and application**.". *International Conference on Enterprise Systems (ES)*, pages pp. 66–77, 2015. DOI: https://doi.org/10.1109/es.2015.14.

[11] P. Oaks, A. H. Ter Hofstede, and D. Edmond. "**Capabilities: Describing what services can do**.". *International Conference on Service-Oriented Computing,Springer*, pages pp. 1–16, 2003. DOI: https://doi.org/10.1007/978-3-540-24593-3-1.

[12] I. s. Standards Board. "**IEEE Recommended Practice for Software Requirements Specications**.". *IEEE Standard*, 830:pp. 04–09, 2000. DOI: https://doi.org/10.1109/ieeestd.1998.88286.

[13] W. Derguech, S. Bhiri, S. Hasan, and E. Curry. "**Using formal concept analysis for organizing and discovering sensor capabilities**.". *The Computer Journal*, 58(3):pp. 356–367, 2015. DOI: https://doi.org/10.1093/comjnl/bxu088.

[14] J. Dabrowski, E. Letier, A. Perini, and A. Susi. "**Mining and searching app reviews for requirements engineering: Evaluation and replication studies**.". *Information Systems*, 114:pp. 102181, 2023. DOI: https://doi.org/10.1016/j.is.2023.102181.

[15] T. Cardona, E. A. Cudney, R. Hoerl, and J. Snyder. "**Data mining and machine learning retention models in higher education**.". *Journal of College Student Retention: Research, Theory and Practice*, 25(1):pp. 51–75, 2023. DOI: https://doi.org/10.1177/1521025120964920.

[16] L. Wijerathna, A. Aleti, T. Bi, and A. Tang. "**Mining and relating design contexts and design patterns from Stack Overflow**.". *Empirical Software Engineering*, 27(1):pp. 8, 2021. DOI: https://doi.org/10.1007/s10664-021-10034-0.

[17] P. Sangaroonsilp, H. K. Dam, M. Choetkiertikul, C. Ragkhitwetsagul, and A. Ghose. "**A taxonomy for mining and classifying privacy requirements in issue reports**.". *Information and Software Technology*, 157:pp. 107162, 2023. DOI: https://doi.org/10.1016/j.infsof.2023.107162.

[18] W. Abdeen, X. Chen, and M. Unterkalmsteiner. "**An approach for performance requirements verification and test environments generation**.". *Requirements Engineering*, 28(1):pp. 117–144, 2023. DOI: https://doi.org/10.1007/s00766-022-00379-3.

[19] R. S. Wahono. "**Analyzing requirements engineering problems**.". *IECI Japan Workshop*, 2003.

[20] M. Hause. "**The SysML modelling language**.". *Fifteenth European Systems Engineering Conference*, 9:pp. 1–12, 2006. DOI: https://doi.org/10.1049/ic:20050131.

[21] P. H. Feiler, D. P. Gluch, and J. J. Hudak. "**The architecture analysis and design language (AADL): An introduction**.". *Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst*, 2006. URL 10.21236/ada455842.

[22] O. MARTE. "**UML profile for modeling and analysis of real-time and embedded systems (MARTE)**.". *ed: OMG*, 2011. DOI: https://doi.org/10.1016/c2012-0-13536-5.

[23] B. SISO. "**Guide for base object model (BOM) use and implementation. SISO**.".

[24] S. Bernardi, J. Merseguer, and D. C. Petriu. "**Model-driven dependability assessment of software systems**.". *Springer*, 2013. DOI: https://doi.org/10.1007/978-3-642-39512-3.

[25] B. Mohit. "**Named entity recognition**.". *Natural language processing of semitic languages: Springer*, pages pp. 221–245, 2014. DOI: https://doi.org/10.1007/978-3-642-45358-8-7.

[26] E. Brill. "**A simple rule-based part of speech tagger**.". *Proceedings of the third conference on Applied natural language processing. Association for Computational Linguistics.*, pages pp. 152–155, 1992. DOI: https://doi.org/10.3115/1075527.1075553.

[27] S. Rochimah and B. Nuralamsyah. "**Decomposing monolithic to microservices: keyword extraction and BFS combination method to cluster monolithic's classes**.". *Jurnal RESTI*, 7(2):pp. 263– 270, 2023. DOI: https://doi.org/10.29207/resti.v7i2.4866.

[28] I. K. Raharjana, D. Siahaan, and C. Fatichah. "**User story extraction from online news for software requirements elicitation: A conceptual model**.". *16th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages pp. 342–347, 2019.

[29] R. M. Fano. "**Transmission of information: A statistical theory of communications**.". *American Journal of Physics*, 29(11):pp. 793–794. DOI: https://doi.org/10.1119/1.1937609.

[30] J. A. Bullinaria and J. P. Levy. "**Extracting semantic representations from word co-occurrence statistics: A computational study**.". *Behavior research methods*, 39(3):510–526, 2007. DOI: https://doi.org/10.3758/bf03193020.

[31] Y. Tian, D. Lo, and J. Lawall. "**Automated construction of a software-specific word similarity database**.". *Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages pp. 44–53, 2014. DOI: https://doi.org/10.1109/csmr-wcre.2014.6747213.

[32] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. "**Incremental clustering for mining in a data ware housing**.". *University of Munich Oettingenstr*, 67.

[33] A. M. Bakr, N. M. Ghanem, and M. A. Ismail. "**Efficient incremental density-based algorithm for clustering large datasets**.". *Alexandria Engineering Journal*, 54(4):pp. 1147–1154, 2015. DOI: https://doi.org/10.1016/j.aej.2015.08.009.

[34] J. Han, M. Kamber, and J. Pei. "**Data Mining: Concepts and Techniques**.". *Morgan Kauffman*, 2011. DOI: https://doi.org/10.1109/icmira.2013.45.

[35] A. K. Jain and J. Moreau. "**Bootstrap technique in cluster analysis**.". *Pattern Recognition*, 20(5): 547–568, 1987. DOI: https://doi.org/10.1016/0031-3203(87)90081-1.

[36] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. "**Influential community search in large networks**.". *Proceedings of the VLDB Endowment*, 8(5):pp. 509–520, 2015. DOI: https://doi.org/10.14778/2735479.2735484.

[37] R. L. Rosnow and R. Rosenthal. "**The volunteer subject revisited**.". *Australian Journal of Psychology*, 28(2):pp. 97–108, 1976. DOI: https://doi.org/10.1080/00049537608255268.

[38] A. Van Lamsweerde, A. Dardenne, B. Delcourt, and F.Dubisy. "**The KAOS project: Knowledge acquisition in automated specification of software**.". *in Proc. of the AAAI Spring Symposium Series, Design of Composite Systems*, 1991:pp. 59–62, 1991.

[39] H. Kaiya, H. Horai, and M. Saeki. "**AGORA: Attributed goal-oriented requirements analysis method**.". *Proceedings IEEE joint international conference on requirements engineering*.

[40] S. Ghaedi Heidari and S. Ajoudanian. "**Automatic pattern-based consistency checking in model refactoring: introducing a formal behavioral preserving method**.". *Innovations in Systems and Software Engineering*, 20(1):pp. 65–84, 2024. DOI: https://doi.org/10.1007/s11334-022-00525-8.

[41] A. Afrin and M. Sadiq. "**An integrated approach for the selection of software requirements using fuzzy AHP and fuzzy TOPSIS method**.". *in 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, pages pp. 1094–1100, 2017. DOI: https://doi.org/10.1109/icicict1.2017.8342722.

[42] M. Sadiq and S. Nazneen. "**Elicitation of software testing requirements from the selected set of software's requirements in GOREP**.". *International Journal of Computational Systems Engineering*, 5(3):pp. 152–160, 2019. DOI: https://doi.org/10.1504/ijcsyse.2019.10022447.

[43] S. Khan, C. W. Mohammad, and M. Sadiq. "**Generating patterns and sub-Patterns of pairwise comparison matrices for the selection of software requirements**.". *International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages pp. 91–97, 2018. DOI: https://doi.org/10.1109/icacccn.2018.8748860.

[44] M. Nooraei Abadeh and M. Mirzaie. "**DiffPageRank: an efficient differential PageRank approach in MapReduce**.". *The Journal of Supercomputing*, 77(1):pp. 188–211, 2021. DOI: https://doi.org/10.1007/s11227-020-03265-3.

[45] M. Sadiq, T. Hassan, and S. Nazneen. "**AHP-GORE-PSR: Applying analytic hierarchy process in goal oriented requirements elicitation method for the prioritization of software requirements**.". *3rd International Conference on Computational Intelligence and Communication Technology (CICT)*, pages pp. 1–5, 2017. DOI: https://doi.org/10.1109/ciact.2017.7977366.

[46] A. Mahmoud and G. Williams. "**Detecting, classifying, and tracing non-functional software requirements**.". *Requirements Engineering*, 21(3):pp. 357–381, 2016. DOI: https://doi.org/10.1007/s00766-016-0252-8.

[47] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E. V. Chioasca, and R. T. Batista-Navarro. "**Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study**.". *ACM Computing Surveys (CSUR)*, 54(3):pp. 54, 2004. DOI: https://doi.org/10.48550/arXiv.2004.01099.

[48] H. Meth, B. Mueller, and A. Maedche. "**Designing a requirement mining system**.". *Journal of the Association for Information Systems*, 16(9):pp. 799, 2015. DOI: https://doi.org/10.17705/1jais.00408.

[49] R. Pinquie, P. Veron, F. Segonds, and N. Croue. "**Natural Language Processing of Requirements for Model-Based Product Design with ENOVIA/CATIA V6**.". *IFIP International Conference on Product Lifecycle Management,Springer*, pages pp. 205–215, 2015. DOI: https://doi.org/10.1007/978-3-319-33111-9-19.

[50] P. Saint-Dizier. "**Mining incoherent requirements in technical specifications: Analysis and implementation**.". *Data and Knowledge Engineering*, 117:pp. 290–306, 2018. DOI: https://doi.org/10.1016/j.datak.2018.05.006.

[51] D. Janssens. "**Natural language processing in requirements elicitation and requirements analysis: a systematic literature review**.". *ACM Computing Surveys*, 54(3), 2019.

[52] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. "**Extracting domain models from natural-language requirements: approach and industrial evaluation**.". *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages pp. 250–260, 2016. DOI: https://doi.org/10.1145/2976767.2976769.

[53] C. Arora, M. Sabetzadeh, and L. C. Briand. "**An empirical study on the potential usefulness of domain models for completeness checking of requirements**.". *Empirical Software Engineering*, 24(4):pp. 2509–2539, 2019. DOI: https://doi.org/10.1007/s10664-019-09693-x.

[54] H. Sadeghi and S. Ajoudanian. "**Optimized feature selection in software product lines using discrete bat algorithm**.". *International Journal of Computational Intelligence and Applications*, 21(1), 2022. DOI: https://doi.org/10.1142/s1469026822500031.

[55] K. Karlapalem, H. P. Yeung, and P. C. Hung. "**CapBasED- A M S- framework for capability-based and event-Driven Activity Management System**.". *CoopIS*, 95:pp. 205–219, 1995.

[56] S. Bhiri, W. Derguech, and M. Zaremba. "**Modelling capabilities as attribute-featured entities**.". *International Conference on Web Information Systems and Technologies. Springer*, pages pp. 70–85, 2012. DOI: https://doi.org/10.1007/978-3-642-36608-6-5.

[57] S. Jung, J. Yoo, and S. Malek. "**A systematic co-engineering of safety and security analysis in requirements engineering process**.". *International Journal of Critical Infrastructure Protection*, 43, 2023. DOI: https://doi.org/10.1016/j.ijcip.2023.100642.

[58] F. Rademacher, S. Sachweh, and A. Zündorf. "**A modeling method for systematic architecture reconstruction of microservice-based software systems**.". *Lecture Notes in Business Information Processing. Springer. Cham.*, 387, 2020. DOI: https://doi.org/10.1007/978-3-030-49418-6-21.

[59] G. Filippone, N. Qaisar Mehmood, M. Autili, F. Rossi, and M. Tivoli. "**From monolithic to microservice architecture: an automated approach based on graph clustering and combinatorial optimization**.". *IEEE 20th International Conference on Software Architecture (ICSA), L'Aquila, Italy*, pages pp. 47–57, 2023. DOI: https://doi.org/10.1109/ICSA56044.2023.00013.

[60] T.N. Kudo, R. d. F. Bulcão-Neto, and V. V. G. Neto and. "**Aligning requirements and testing through metamodeling and patterns: design and evaluation**.". *Requirements Eng*, 28:97–115, 2023. DOI: https://doi.org/10.1007/s00766-022-00377-5.

[61] W. Derguech and S. Bhiri. "**Modelling, interlinking and discovering capabilities**.". *ACS International Conference on Computer Systems and Applications (AICCSA)*, pages pp. 1–8, 2013. DOI: https://doi.org/10.1109/aiccsa.2013.6616444.