UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Tales Lopes Silva

An Architectural Framework for Expert Identification Based on Social Network Analysis

Juiz de Fora

2022

Tales Lopes Silva

An Architectural Framework for Expert Identification Based on Social Network Analysis

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Victor Ströele de Andrade Menezes

Coorientadora: Prof. D.Sc. Regina Maria Maciel Braga Villela

Juiz de Fora

2022

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Silva, Tales Lopes.

An Architectural Framework for Expert Identification Based on Social Network Analysis
/ Tales Lopes Silva. – 2022.
83 f. : il.

Orientador: Victor Ströele de Andrade Menezes
Coorientadora: Regina Maria Maciel Braga Villela
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2022.

1. Global Software Development. 2. Expert Recommendation System. 3. Syntactic Analysis. 4.Semantic Analysis. 5.Collaboration Network Model. 6.Machine Learning. 7.Ontology. 8.High-volume Social Network. I. Menezes, Victor Ströele De Andrade, orient. II. Villela, Regina Maria Maciel Braga, coorient.

Tales Lopes Silva

An Architectural Framework for Expert Identification Based on Social Network Analysis

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Tales Lopes Silva

An Expert Recommendation System Architecture Based on Social Network Analysis

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 05 de dezembro de 2022.

BANCA EXAMINADORA

Prof. Dr. Victor Ströele de Andrade Menezes - Orientador
Universidade Federal de Juiz de Fora

Profª. Dra. Regina Maria Maciel Braga Villela - Coorientadora
Universidade Federal de Juiz de Fora

Prof. Dr. José Maria Nazar David
Universidade Federal de Juiz de Fora

Profª. Dra. Celia Ghedini Ralha
Universidade de Brasília

Juiz de Fora, 01/11/2022.

Documento assinado eletronicamente por **Victor Stroele de Andrade Menezes**, **Professor(a)**, em 08/12/2022, às 09:30, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Jose Maria Nazar David**, **Professor(a)**, em 08/12/2022, às 11:55, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Regina Maria Maciel Braga Villela**, **Professor(a)**, em 08/12/2022, às 18:55, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Celia Ghedini Ralha**, **Usuário Externo**, em 13/12/2022, às 23:56, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **TALES LOPES SILVA**, **Usuário Externo**, em 02/03/2023, às 08:33, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **1021020** e o código CRC **7B08A9E2**.

AGRADECIMENTOS

A Deus, pela dádiva da vida e pelas bênçãos de cada dia. Aos mestres do invisível, pelo auxílio no caminho.

A minha família, em especial aos meus pais que sempre fizeram o possível e impossível, se sacrificaram e dedicaram a vida para que seus filhos tivessem acesso à educação e ensino.

A minha querida irmã Paula e avó Laide, por serem sempre fonte de inspiração, força e resiliência.

Aos amigos de verdade, por estarem sempre ao meu lado.

Ao Professor Victor, não só pela orientação e colaboração em todos os trabalhos que realizamos, mas também por todos os anos que esteve ao meu lado em minha jornada acadêmica. Pelos ensinamento, pelos momentos de compreensão, pela força nas dificuldades, e pela amizade que será levada por toda a vida.

A professora Regina, por toda colaboração e ajuda essencial nesse trabalho. E por ser uma verdadeira mãe, sempre me orientndo e me guiando pelo melhor caminho.

Aos professores que contribuíram para o desenvolvimento desse trabalho, Professor José Maria, Professor Mario Dantas, Professor Jean-françois (Université Granoble), Professor Michael Bauer (University of Western), além dos demais professores do PPGCC que foram fundamentais para minha formação profissional e acadêmica.

RESUMO

A análise de redes sociais tem sido amplamente utilizada em diferentes contextos de aplicação. Por exemplo, em Desenvolvimento Global de Software, onde vários desenvolvedores com diversos conhecimentos e habilidades estão envolvidos, o uso de modelos de redes sociais ajuda a entender como esses desenvolvedores colaboram. Encontrar especialistas que possam ajudar a abordar elementos ou problemas críticos em um projeto é uma tarefa desafiadora e crítica. Isso é especialmente verdade em projetos no contexto de Desenvolvimento Global de Software, onde desenvolvedores com habilidades e conhecimentos específicos geralmente precisam ser identificados. Nesse sentido, buscar membros essenciais é uma tarefa valiosa, pois eles são fundamentais para a evolução da rede. Este trabalho propõe um framework arquitetural para a identificação de especialistas como uma solução híbrida que inclui análise sintática e semântica em redes sociais. Buscamos abordar desafios de pesquisa relacionados ao projeto de sistemas de recomendação que envolvam a análise de estruturas sociais no contexto de Desenvolvimento Global de Software. Nesta solução, definimos um modelo para a rede social capaz de capturar a colaboração entre desenvolvedores, incorporamos estratégias de análise temporal da rede, exploramos a rede usando algoritmos de aprendizado de máquina, propomos uma ontologia para enriquecer os dados semanticamente e consideramos uma abordagem performativa para métodos de análise de redes de grande volume. Realizamos quatro estudos de caso usando dados extraídos do GitHub para avaliar a abordagem proposta, bem como um conjunto de dados de grande volume para os estudos de performance. Os estudos de caso fornecem evidências de que o método proposto pode identificar especialistas, destacando sua expertise e importância para a evolução da rede social.

ABSTRACT

Social network analysis has been widely used in different application contexts. For example, in Global Software Development (GSD), where multiple developers with diverse skills and knowledge are involved, the use of social networking models helps to understand how these developers collaborate. Finding experts who can help address critical elements or issues in a project is a challenging and critical task. It is especially true in the context of Global Software Development projects, where developers with specific skills and knowledge often need to be identified. In this sense, searching for essential members is a valuable task, as they are fundamental to the evolution of the network. This work proposes an architectural framework for expert identification as a hybrid solution that includes syntactic and semantic analysis in social networks. We seek to address research challenges related to designing recommendation systems when analyzing social structures in the Global Software Development context. In this solution, we define a model for the social network capable of capturing collaboration between developers, incorporate strategies for temporal analysis of the network, explore the network using machine learning algorithms, propose an ontology to enrich the data semantically, and consider a performative approach for high-volume social network analysis methods. We conducted four case studies using data extracted from GitHub to evaluate the proposed approach, as well as a more extensive dataset for the performance studies. The case studies provide evidence that our proposed method can identify specialists, highlighting their expertise and importance to the evolution of the social network.

Keywords: Social Network Analysis. Global Software Development. Expert Recommendation System. Syntactic Analysis. Semantic Analysis. Collaboration Network Model. Machine Learning. Ontology. High-volume Social Network.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GSD	Global Software Development
RQ	Research Question
SRQ	Secondary Research Question
Q&A/QA	Question and Answer
CQA	Community Question Answering
TTEA	Temporal Topic Expertise Activity
TET	Temporal-Expert-Topic
ETL	Extract, Transform, and Load
fp	false positives
tp	true positives
fn	false negatives
tn	true negatives
API	Application Programming Interface
DBLP	Digital Bibliography & Library Project
eps	epsilon
minPnts	minimum points
Q	Question
e.g.	exempli gratia; for example
i.e.	id est; that is
PR	Pull Request
dev	developer
LR	Logistic Regression
LDA	Linear Discriminant Analysis
NB	Naive Bayes
KNN	K-Nearest Neighbors
SVM	Support Vector Machines
size/XS	Extra Small Size
size/M	Average Size
size/L	Large Size
ORSD	Ontology Requirement Specification Document
OWL	Web Ontology Language
SWRL	Semantic Web Rule Language
R	Rule
ORSD	Ontology Requirements Specification Document
CQ	Competency Question
IQR	Interquartile Range
CN	Core Nodes
mst	Minimum spanning tree
hd	highest degree

LISTA DE SÍMBOLOS

$*$	multiplication
$=$	is equal to
$+$	plus
$-$	minus
$\in$	is member of
$>$	is greater than
$<$	is less than
$\%$	percentage
$\wedge$	and (ontology rules)
$\rightarrow$	property value assignment (ontology rules)
$\sum$	summation

SUMÁRIO

1 Introduction

The world is evolving fast, and a large volume of information is becoming increasingly available. Data is generated every moment from different sources, such as IoT devices and social media (1). Furthermore, the growing complexity of software development processes and demand for fast software delivery bring challenges associated with the need for efficient methods capable of extracting knowledge from data. In this sense, data analysis is valuable as it aims to transform datasets into valuable information, converting them into insights for decision-making (2).

Global Software Development (GSD) has arisen driven by the growing importance of software as a vital component in almost every business (3, 4). It is a complex sociotechnical system motivated by various advantages: reduced software development time and cost, access to a large multi-skill pool of individuals, and responsiveness to customers and market needs (5). Furthermore, GSD promotes the emergence of social structures with numerous individuals worldwide working towards the same goal (6). As a result, we have the appearance of version control platforms where teams of developers collaborate to produce working software (7).

The complexity, effort required, and unpredictability in large-scale software development (8) raise challenges associated with finding qualified professionals to assist with specific tasks. Experts can be seen as experienced developers capable of solving complex tasks to achieve project goals and help other developers (9). Moreover, they are essential individuals in GSD, as they are often responsible for most of the relevant contributions in software development (10). However, identifying those suitable to assist with project needs is difficult, especially considering that various characteristics, such as technical knowledge, previous experiences, collaborative skills, and availability, may influence the search for the right person (11). Other important characteristics should also be considered when looking for suitable developers in global contexts, such as cultural factors, geographical differences, and communication barriers (12).

Recommendation Systems (RS) are software applications that seek to provide suggestions to meet users' needs and preferences (13). They are fundamental in GSD as they reduce the human effort of finding suitable developers by providing suggestions based on the analysis of developers' past behavior and interpersonal relationships (14). Furthermore, its use makes the software development process more effective by supporting decision-making activities, such as recommending developers to assist in specific tasks (15) or allocating entire teams to work on project components (16).

However, there are challenges in designing recommendation systems when analyzing social structures in GSD:

(i) **Temporal information**: Among existing contextual aspects of data, temporal

information can be considered one of the most valuable. It facilitates tracking the evolution of individuals' preferences by modeling habits and interests over time (17). Taking the time dimension into account is essential as people's interests are constantly changing, impacting how they interact with each other and their ranking in recommendations (18). Not considering temporal information may result in giving greater importance to individuals who were but are no longer involved in a particular subject, e.g., a developer who has contributed to a specific topic in the past but has lost interest in that topic over time.

(ii) **Work overload**: Traditional approaches often lead to the design of analytical models that recommend the same group of popular developers (19). Such a naive system could drastically increase the requests for assistance from these individuals, who could end up overloaded with numerous tasks to solve (20). Moreover, the excessive workload faced by some developers promotes a stagnant collaboration scenario with low knowledge dissemination, reduced productivity, and increasing software defects (21). Thus, identifying developers who are not obvious but have similar skills to those considered experts expands the diversity of recommendations by identifying additional potential individuals who could contribute to project needs (22).

(iii) **Semantic Analysis**: Several studies have been developed over the years to support data analysis approaches (23). However, learning from data is more than just analyzing data. Some authors (24) suggest that there is no meaningful knowledge discovery without understanding the context information in which data is generated. Thus, a range of strategies can be used separately or advantageously integrated into data analysis methods to incorporate semantics (25). Including semantic analysis in data analysis systems can increase the quality of the study by bridging semantic gaps between data (26, 27).

(iv) **Data Volume**: Finally, in GSD, we have millions of registered developers, repositories, and projects, not to mention information obtained from collaboration features that reach billions of available data (28). The growing volume of data expands the size of social structures while their complexity evolves with the number of connections to be analyzed (29). Therefore, evaluating the system's performance when developing network analysis approaches is crucial if we want to produce optimized methods that perform in a feasible time (30).

1.1 Proposal

Considering the four research challenges related to analyzing social structures in GSD, we present a new architectural framework for expert identification. It is a comprehensive solution for syntactic and semantic analysis in social networks in the Global Software Development context. Our approach combines analysis techniques as different components of a system to explore distinct aspects of the network, helping us identify those with specific knowledge. In this solution, we define a temporal model for social networks

capable of capturing collaboration interactions between developers over time. Thus, we seek to identify and classify experts by investigating the evolution of their communities in social networks. Furthermore, in order to boost decision-making and improve the overall process, we explore the network using machine learning algorithms and propose an ontology to enrich the data semantically. The approach also addresses issues related to the volume of data, including a performative solution for executing social network analysis methods.

Figure 1 highlights the contributions of our work along with the proposed analytics process. First, we model the data as a social network, representing the relationship between individuals that best aligns with the goals we seek to solve. Then we consider temporal information and model the network over sequential periods to analyze how its structure evolves. Finally, we analyze the evolution of the network formed among developers who collaborate to achieve project goals.

Syntactic and semantic analyses are implemented separately to explore the data more comprehensively. In linguistics, syntactic analysis is related to processes that examine sentential structures, and here the term is used to address the study of network structures (31). Therefore, in the syntactic analysis step, we characterize the network and use a clustering algorithm to investigate how the influence of developers changes over time. We work to identify the individuals considered essential for network evolution and the development of projects. Furthermore, we seek to identify less obvious developers with similar skills to those considered influential using both clustering and classification approaches. To that end, we propose and integrate a diversity-based study focused on reducing the work overload faced by highly requested individuals.

On the other hand, semantic analysis is the process of drawing meaning from data. When combined with syntactic analysis, semantic-based methods allow us to delve deeper into the analysis context, extracting knowledge from semantic structures. In this sense, we create an ontology to categorize relations between the entities in the network. Ontologies are logic models that explicitly represent concepts' meaning (semantics) and relationships. We use the model to create a knowledge graph by inferring implicit connections between objects. Finally, we focus on finding individuals with advanced knowledge or skills in the network by analyzing the semantic structure.

We also employ a parallelism study to investigate the performance of our approach when dealing with large data sets. We propose a graph partitioning strategy to reduce the complexity of large network structures by dividing them into smaller, less-connected parts to be processed in parallel. Our goal is to improve the processing time of network analysis techniques, particularly density-based clustering algorithms, as they are widely used in detecting and analyzing communities in social networks (32). In addition, the parallelism study is responsible for supporting all other steps of the analysis process.

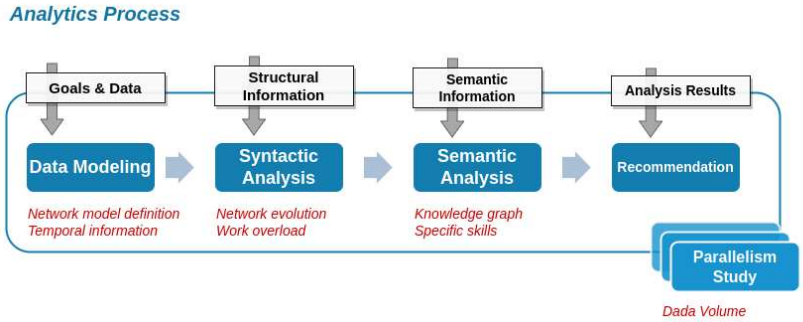

Figure 1: Outlining the analytics process. The research challenges we address are in red italics.

Thus, we summarize the main contributions of our approach:

- We perform a temporal characterization of the network to explore how its structure evolves.
- We introduce a diversity-based analysis to address the problem of potential work overload of certain experts.
- We create a semantic model using an ontology to explore implicit aspects of the network.
- We propose a parallelism study and a new network partitioning strategy focused on investigating and improving the performance of clustering algorithms.
- We propose an architectural framework combining syntactic and semantic analysis techniques to identify experts in GSD.

To guide our study, we highlight a research question (RQ1) that should be answered by the end of this work:

- (RQ1) *How to develop an approach integrating the proposed analytical strategies?* We present a new architectural framework for expert identification that includes solutions to the four research challenges related to analyzing social structures in GSD.

Furthermore, in order to answer RQ1, we define four secondary research questions (SRQ).

- (SRQ1) *What characterizes individuals as essential for network evolution?* We present a temporal characterization approach to identify active members (nodes) and search for individuals considered crucial to the projects. We investigated several

overlapping structures corresponding to consecutive timestamps, with the aim of tracking the behavior of central nodes in the network and understanding how the project evolves over time.

- (SRQ2) *How to implement an approach that allows increasing diversity of recommended developers?* We propose a diversity-based approach as a solution to the work overload problem. We seek to identify developers who are not obvious but have similar skills to those considered experts. We compared some popular classification algorithms to explore individuals recognized as *false positives*; individuals who may have characteristics similar to those identified as *true positives*. In addition, we apply some centrality algorithms seeking in order to understand the role of *false positives* individuals in the network.

- (SRQ3) *How can an ontology assist in identifying developers with specific expertise?* We create an ontology to explore implicit aspects of the network. The ontology is used to infer the expertise of developers by creating relationships between keywords and knowledge topics. Individuals are classified according to their degree of expertise in the network and considering temporal aspects to explore the expertise interest over time, prioritizing more recent project activities. The semantic model is supported by an initial step focused on creating a knowledge domain to define the project's expertise.

- (SRQ4) *How develop a network partitioning strategy that optimizes the execution time of clustering algorithms?* We presented a new and unprecedented parallel graph partitioning algorithm as well as its implementation details and obtained results in an extensive data set. Sequential and parallel approaches were developed and compared to evaluate the algorithm's feasibility. Considering the execution time and memory consumption, we carried out a performance analysis.

The text of this dissertation is organized as follows: Chapter 2 presents some concepts about expert recommendation systems and related work. Chapter 3 introduces the architecture of the proposed solution, including explaining each step involved in the approach, and describes the data and the social network model used in this study. Chapter 4, 5, 6, and 7 present the historical research related to temporal categorization, diversity-based analysis, semantic analysis, and network partitioning, respectively. Finally, Chapter 8 presents conclusions and future works.

The development of this work resulted in the publication of five published works (33)(14)(34)(10)(35).

All code and data used in this paper can be accessed at[1,2,3,4].

[1] https://github.com/Talessil/DRecSys
[2] https://github.com/Talessil/Temporal-Analysis
[3] https://github.com/Talessil/ExFindO
[4] https://github.com/Talessil/parallelized-dc

2 Backgroud and Related Work

Considering the research areas on which this work draws, we highlight in this chapter the main concepts needed for the proposal understanding. The first two sections present the background regarding Social Network Analysis and Expert Recommendation Systems. The last section presents the related work considering research published in these areas.

2.1 Social Network Analysis

Social networks are sets of connected objects represented by a graph in which edges relate to nodes or vertices. A social network reflects a social structure where individuals or organizations and their relationships can be represented. For example, relationships generally represent one or more types of interdependence (such as ideas and religion) or more specific relationships (knowledge sharing, information, and friendship, for example). This social structure allows data and information exchanged between individuals or organizations to be studied and analyzed at different levels of detail (36).

In social network analysis, many works are related to the key topic of Pattern & Knowledge Discovery (37), and a key issue is community detection (38). As members of social networks are very likely to participate in communities, finding them has drawn the attention of many researchers. There are many applications considering this issue in different fields, such as viral marketing (39), expert finding (40), knowledge sharing (41), and others. Furthermore, the interdisciplinary nature of the subject has increasingly stimulated the study and development of algorithms and techniques to analyze network topology, define clusters for communities' identification, and locate influencing elements, connectors, and information diffusers (37).

According to (42), there are different types of people in a Social Network, and we can highlight two: central connectors and information brokers. Central connectors characterize people with many relationships; those influential people who communicate across subgroups, maintain a large connected group, or connect two groups, are considered information brokers. Analyzing these kinds of people is important to characterize the social network to identify responsible people by maintaining communication among people, groups, and the whole network. Semantic meaning can also improve this characterization, and specific analyses can be used to identify semantic connections among people who share similar interests that are not explicitly stated.

Community detection consists of finding groups in the network so that the members of each group are more similar to themselves than members of other groups. Many approaches toward a solution to this problem can be found in the literature (43, 44), and several studies (45, 46) have reviewed the most common methods to deal with the

community detection task.

The problem is that most existing studies focus only on the link analysis or the topological structure of the network (47). Also, they do not consider dynamic social networks with different snapshots of the network (48). Thus, these methods do not use the content available in the network, and the detected communities are often formed by members with different interests (49). Some recent studies have shown that more meaningful communities can be identified by enriching the network semantically (37), showing members with strong connections and similar interests simultaneously (50, 49).

2.2 Expert Recommendation Systems

Finding suitable individuals to meet needs that demand specific knowledge is challenging. Experts are individuals with deep knowledge or skill based on experience in a particular study area. An expert can be defined as an individual with a score higher than a determined threshold in a given topic or subject (51). They can be discovered by analyzing data from user profiles believed to have certain knowledge and experience. These profiles can be obtained from the content of documents related to people's activities (52). However, the concept of "expert"is frequently unclear, and it is difficult to define measures that identify a person as such. Moreover, candidates' areas of expertise are difficult to quantify, while their experience constantly changes over time. Consequently, people often turn to the most influential individuals as experts, ignoring emerging ones (53).

It becomes increasingly necessary to identify the most qualified and who may work on specific needs in their respective areas. For example, in the industry, professionals use networking platforms (e.g., Linkedin) to search for potential candidates to fill job openings in information technology areas (54). In academia, social networks capture previous collaborations between researchers, whose analysis assists in finding those who could carry out article reviews (55) or contribute to different research groups (32). Furthermore, in addition to being required to actively answer questions on Q&A platforms (56), experts are needed to assist with specific tasks as global, distributed software development evolves more and more.

Expert recommendation systems seek to gather the past reputation of candidates, rank them, and provide a ranked list of those whose experience best matches the user's query (56). These systems can be implemented considering both topic-oriented and expertise-oriented searching models. In topic-oriented searching systems, the goal is to find an individual who is knowledgeable in a particular topic (57). On the other hand, expertise-oriented searching systems focus on finding the fields with the greatest similarities to an expert's specialty (58).

Although the first works related to recommendation systems date back to the 1970s (59, 60), expert recommendation systems are an emerging research field that seeks to

provide suggestions on those considered to be specialists (61). More recent recommendation works apply machine learning (deep learning) (62), graph analysis (63), and semantic approaches (64). Also, they are a branch of general recommendation systems, and one of the most significant differences is in the source of knowledge extraction. Different data types can be extracted from specific repositories and used for different recommendations, such as games (65) or movies (66). On the other hand, when we seek to recommend people, collecting information about their knowledge and experiences becomes more complex, as they are often members of various social networks (58).

2.3 Related Work

Social network analysis has been the subject of several recent studies as effective graph analysis provides a deeper understanding of data and can benefit many applications such as node classification, node recommendation, and link prediction.

In research focused on multiple dimensions, social networks are analyzed concerning the big data paradigm dimensions (67, 68). Particularly in (67), Camacho et al. propose the definition of four different dimensions inspired by the popular V model (69) used in the Big Data area. These are based on four essential features in social network analysis, Pattern & Knowledge discovery, Information Fusion & Integration, Scalability, and Visualization, which are used to define a set of new metrics to evaluate the different social network analysis software frameworks and tools. Also, a multiple-dimensional analysis, including spatial, social, temporal, and semantic perspectives, is conducted in (70) in order to understand Twitter users' discussion. They argue that multi-dimensional analysis can reveal complicated patterns and answer questions that cannot be addressed with a single-dimension analysis (70).

However, most graph analytics methods suffer the high computation and space costs. Graph embedding is an effective yet efficient way to solve the graph analytics problem (71). The authors in (72) introduce a new setting for graph embedding, which considers embedding communities rather than individual nodes. They find that community embedding is useful for community-level applications such as graph visualization and provides an opportunity to improve community detection and node classification.

Social network analysis has also been explored in several studies aimed at detecting experts to assist with project issues (73, 74). In this sense, analyzing social networks in search of specialists can be carried out from the investigation of structural data, representing the connections, interactions, and topology of the network (75), as well as content data that is focused on existing and shared information within the network by the users (76).

In recent works (77, 78), collaborative models are structured and explored to identify communities' specialists according to their expertise. In (77), a new approach

to modeling researchers is proposed to recommend experts in scientific communities, considering factors such as topic relevance, expert quality, and researcher connectivity. It addresses a good way to solve expert identification problems by employing a deep learning method that combines user feature representations with question feature representations to compute scores. In (78), the authors propose an expert finding method, named NEWHITS, which considers the topical similarity of the users and can adapt well to the feature of the CQA. Also, while few works aim to find appropriate individuals to help with GSD issues, they are not directly related to the term expert. Instead, they recommend individuals to help with code reviews based on their historical contributions (77) or previously explored code (79).

These approaches are considered static algorithms in community discovery, as they refer to methods that do not take into account the evolution of social networks, i.e., the 'time' variable is not considered in the network modeling. On the other side, dynamic community finding algorithms (76) refer to those approaches that incorporate the variable time into the model. Temporal analysis was used to identify patterns of user contributions (80) and show that analyzing the evolution of their activities can be used to describe changes in the network structure (81). Thus, work with temporal information is considered a significant factor in the analysis process of social networks as it broadens the range of possible analysis, giving an idea about the changes in the relationships between individuals based on any time interval (82).

The evolution of social networks can be described by analyzing them based on some aspects over time, e.g., shared activities, members' associations, the similarity between individuals' attributes, and the closure of network cycles (83). Compared to static graph analysis, temporal models allow better recognition of users' common interests and predictions about their future activity. Furthermore, understanding how members and their interactions evolve in communities can present key insights into identifying experts (84).

When using machine learning models on evolutionary data, the authors in (84) demonstrated that identifying experts by observing their temporal activity outperforms models that use static data snapshots. In (9), a temporal analysis performed on overlapping communities showed that overlapping nodes might be associated with either multidisciplinary developers collaborating on different technologies simultaneously or changing their interests. Also, statistically, significant improvements were observed when incorporating temporal information to model dynamic user expertise (85). They modified two widely used approaches, Answer Count and ZScore, by applying exponential and hyperbolic penalization models to discount the effect of older records. In (73), the authors consider that light knowledge in other areas allows them to collaborate on different aspects of the project. Therefore, they propose a method for T-shaped expert findings based on temporal expert profiling. They regularly take snapshots of expertise trees to learn the

relation between temporal changes in different candidates' profiles. Finally, (74) presents the TTEA (temporal topic expertise activity) model that simultaneously uncovers the topics, activities, expertise, and temporal dynamics to study the user behaviors and topics dynamics. They demonstrated that TTEA shows advantages in topic modeling, question routing, expert recommending, and community life-cycle management.

We could see that only structural data analysis would provide an incomplete view of the information and patterns stored in the network. Therefore, graph metrics, machine-learning algorithms (86), and semantics approaches (73) are becoming more common in recommendation systems, although similarity estimation always proves to be a critical problem. While traditional expert recommendation approaches ignore semantic-based information, in (83) is proposed a novel Temporal-Expert-Topic (TET) approach based on Semantics and Temporal Information for temporal expert finding, which means identifying a person with given expertise for different periods. They claim to be the first to deal with the temporal expert finding problem by proposing a generalized time topic modeling semantic approach. We can see expertise-oriented (9) and topic-oriented (85) approaches used as the basis for such semantic search models.

Some strategies are used to analyze semantically constructed graphs (87, 88). For instance, GraphDBLP is a tool that models a social network as a graph and enriches the data through semantic keyword similarities computed via word embedding. The authors in (89) propose HQExpert, a novel expert-finding method in CQA based on multi-granularity semantic analysis and interest drift. They represent an expert domain based on semantic analysis and an expert ranking strategy based on quality optimization to draw users' time-sensitive interests at different periods. A new deep model for expert findings based on convolutional neural networks is proposed in (90). Their model tries to detect users' expertise by simultaneously learning patterns from users' documents and queries. Also, (32) seeks to verify the level of influence among researchers by analyzing a bidirectional graph-based model. Then, they use ontological terms and rules to identify new connections and promote scientific collaboration.

Furthermore, diversity-based methods aim to recommend non-common people (91, 91) rather than follow traditional strategies that lead to the same group of individuals (92). Diversity in expert recommendation systems consists of *identifying developers who are not obvious but have similar skills to those considered experts* (93). Its authors consider the work in (94) as the first to study diversity in the presence of social networks and demonstrate the more diverse the group members are, the better quality of the group of experts. The lack of diversity may bring situations where experts begin ignoring their expertise and follow the crowd due to social pressure or the information cascade effect (95).

2.3.1 Discussion

Comparatively, what differentiates our work from others are i) we performed the analysis of social networks by exploring structural and semantic aspects. We performed a temporal characterization of the network, searching for communities and identifying essential members for the evolution of the network. Many studies identify individuals that are somehow relevant but not crucial to the evolution of the network structure. ii) We employ machine learning models and apply centrality metrics to classify developers according to their contribution to the network. With this, we further explored the study of the term diversity, showing that the lack of diversity can inhibit the performance of some specialists, bringing situations in which other specialists are overloaded with tasks to solve. iii) We address the use of the term expert in GSD, and for that, we propose a taxonomy categorizing keywords in GSD projects, which extends to future related work. Thus, we propose an ontology to explore implicit aspects of the network through a semantic perspective and use ontological terms and rules to identify experts and expertise. Some studies focus on identifying experts according to their expertise but not on identifying them by linking developers with implicit semantic expertise and their change of interest over time. iv) Finally, while graphical metrics, machine learning algorithms, and semantic approaches are becoming more common in recommender systems, they are barely seen together in the approach. Thus, we put together all the approaches above to perform enriched data analysis and compose an architectural framework for expert identification in GSD. Some studies combine the analysis of different network dimensions to analyze its structure but do not focus on improving the detection of communities.

2.3.2 Parallelism

Be $G = (V, E)$ an undirected graph with a set of nodes $V = \{v_0, v_1, ..., v_n\}$ and edges $e_{ij} = (v_i, v_j)$, with e_{ij} representing a relationship between nodes i and j. The graph minimum cut problem can be defined as the division of V into n smaller parts minimizing the edge cut (96). It can also be extended to the balanced graph partitioning problem (97), where the partitions need to be approximately sized. Furthermore, the particular case $n = 2$ leads to a classic discrete optimization problem: the minimal bisection problem (98, 99). The bisection problem has proved to be an NP-hard (100) since it requires a new cardinality constraint: $|B| = |V|/2$, on each partition (bisection) $B \subset V$.

With advances in infrastructure and the emergence of even larger datasets, some current studies have employed high-performance clusters to obtain high-quality partitions in complex networks with billions of edges. They include graph partitioning and matching methods applicable to large-scale graph analysis (101, 102), adaptations of techniques for parallel graph clustering algorithms (103), and partitioning schemes for graph community detection (104).

However, despite advances in works related to graph partitioning, the lack of research aimed at detecting communities in parallel is still a difficulty (104). We found publications on parallel clustering techniques applied in social networks (104, 105, 106), but none considers parallel partitioning with a focus on community detection by density-based clustering algorithms. In addition, studies are found that consider execution time in their analysis(107), but none include the analysis of memory consumption.

In a structured overview of graph partitioning studies in the literature, (108) highlight the contrast between demands for different approaches to dealing with larger datasets, and the fact that graph partitioning methods are becoming less likely to emerge. Thus, concerns about future improvements in this area grow as analytics networks are getting larger and more challenging to deal with today's partitioners. Moreover, to the best of our knowledge, no graph partitioning work guarantees a process that split the graph into connected subnets (connected components) and allows for overlapping nodes to exist. This further emphasizes the need for more effective parallel partitioning methods for graph approaches.

The aspects that make our proposal different from previous studies are: (i) we propose a new parallel graph partitioning algorithm focused on obtaining balanced partitions along with execution time feasibility in social networks. (ii) Our approach guarantees the connectivity of each partition as well as allows overlapping nodes between different partitions. Thus, the application of density-based clustering algorithms can be performed on the graph partitions in order to detect communities. (iii) Besides time consumption, we also perform algorithm analysis to evaluate its behavior in terms of memory consumption.

3 Proposed Solution

This chapter presents the proposed architectural framework for expert identification. The framework implements a hybrid analysis of social networks by integrating distinct and complementary approaches to deal with the challenges raised in the introductory chapter.

3.1 Architecture Overview

Since we address the study and analysis of social networks, we propose an architectural framework designed to explore collaborative networks in GSD contexts. Figure 2 presents the proposed conceptual architecture, which is defined in seven layers.

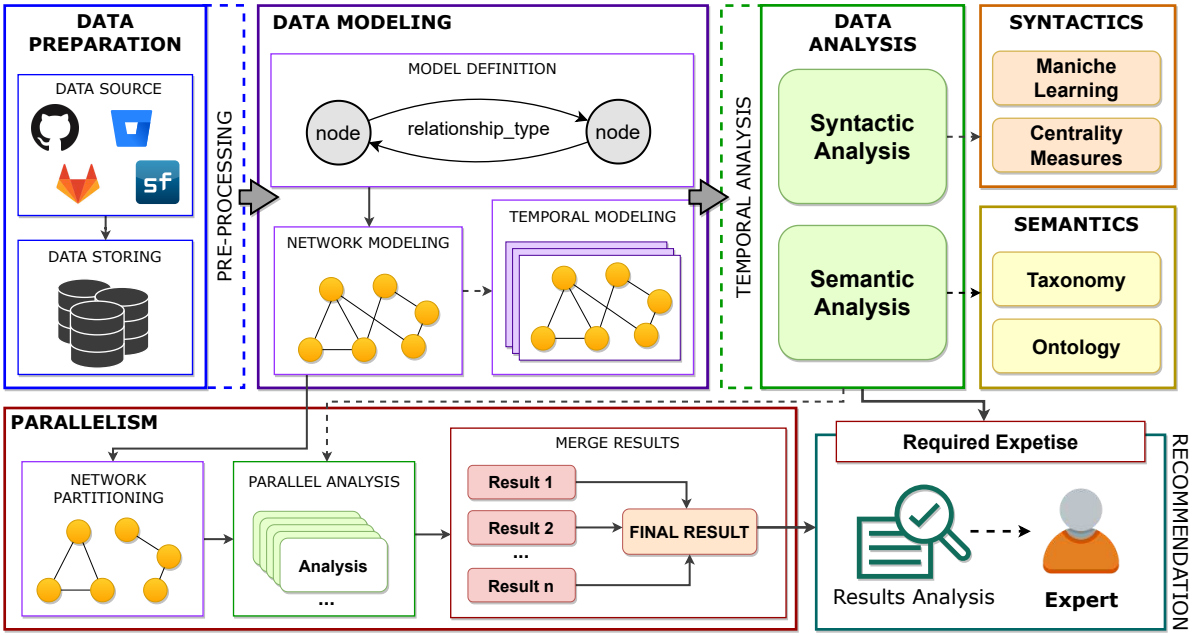

Figure 2: Architecture for an Expert Recommendation System.

Initially, fundamental actions, such as data scraping and storage, are employed in the **Data Preparation** layer. This step collects data from code hosting and version control platforms like GitHub, focusing on repositories for software development projects. Information captured includes developers' data and elements describing how they collaborate to achieve project goals. Still, the data is pre-processed in this stage, going through cleaning, integration, reduction, and transformation steps until it is ready to be ingested by the data modeling layer. This is a data integration process combining extraction, transformation, and loading (ETL) (109). It was implemented without any existing framework (e.g., Hevo Data, Airbyte, Apache Kafka, Pentaho Kettle, Apache Camel) but using python for us to develop our own procedures.

Next, in the **Data Modeling** layer, social network structures are generated according to the network model that best represents the relationships in the problem

investigation context. This is done by defining the graph structure: nodes and connections. The network structures are stored using both Neo4j, a graph database management system, and NetworkX, a Python library for studying graphs and networks. Once the graph model is established, the modeling process is extended using a temporal approach to represent the network in specific periods. Including time information in the modeling process allows us to generate and analyze time frames in order to understand how the network evolves.

With the network structures in hand, we can use several methods separately or together to compose analysis strategies. We divided them into two groups: syntactic and semantic analysis approaches, as illustrated in the **Syntactics** layer and **Semantics** layer, derived from the **Data Analysis** layer. Syntactic analysis approaches include methods focused on exploring the structural aspects of networks. In this sense, machine learning algorithms, such as Logistic regression (LR) (110), Linear Discriminant Analysis (LDA) (111), Naive Bayes (NB) (112), K-Nearest Neighbors (KNN) (113), and Support Vector Machines (SVM) (114), and centrality analysis measures, such as Closeness, Betweenness, and Page Rank (115), are examples of approaches for this purpose. In addition, semantic analysis approaches focus on representing and interpreting the meaning of data in a specific context. Therefore, knowledge graphs and their database structure are defined with a focus on the applications we target to build; defined by the task, and the ontology is defined from the domain knowledge, containing the definition of concepts and relationships for a given domain as well as the domain rules.

Furthermore, when dealing with large and more complex structures, a good strategy is to partition the network into less-connected structures. Therefore, the architecture also implements a parallelism strategy that optimizes network analysis approaches' execution time. That approach is represented in the **Parallelism** layer, where the network structures are divided into smaller partitions, the new partitions are analyzed in parallel, and the partial results are merged to generate the final result.

Finally, regardless of the approach adopted, the results are gathered and analyzed in the **Recommendation** layer in order to identify and recommend specialists according to the required expertise.

In the next sections, we detail how the architecture can be used to address the research challenges related to the search for experts presented in Chapter 1. Then, we define strategies from the combination of components present in the architecture and name them according to the challenges of research they aim to solve (see Figure 3).

3.2 Temporal Characterization

In this subsection, we present the network's temporal characterization method. It is proposed as a solution to deal with the research challenges related to using temporal information in expert recommendation systems.

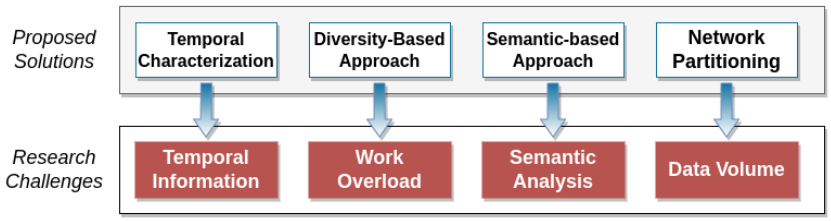

Figure 3: Research challenges and proposed solutions.

As illustrated in Figure 4, we consider the temporal characterization to be a two-stage approach. In the first stage, we model the data as a social network and employ temporal modeling to represent the network over sequential periods, defined by monthly snapshots. The temporal modeling seeks to generate overlapping structures that can be analyzed sequentially, that is, in chronological order to observe how the network evolves. In the context of a network representing collaborative relationships, investigating the network evolution allows us to track the contribution activity between developers. Therefore, this stage can identify developers who are considered influential due to their importance to the network evolution by analyzing their contribution degree. In summary, this first stage aims to reduce the initial number of developers to a group of more relevant individuals in the network.

The second stage extends the first one by working towards searching for those considered essential for the network evolution. These individuals may play key development roles in the project and, therefore, are critical to the project's growth. Here, we delve further into the concept of what it means to be an influential developer, looking at how their influence degree evolves. This analysis allows us to differentiate individuals considered influential exclusively at specific periods or over a longer period in the project. In addition, we use domain knowledge to enrich our analysis with semantic information, fundamental to assist us in answering the "whys"of questions not possible to be elucidated by only analyzing structural aspects of the network. The domain knowledge is recovered from project documentation related to development resource labels.

3.3 Diversity-based Approach

In order to reduce the workload on highly in-demand individuals, we present the diversity-based approach (see Figure 5) as a solution to deal with the research challenges related to working overload in expert recommendation systems. We seek to reduce the workload of some highly-requested individuals by increasing the diversity of recommended developers by recommending those who are not obvious but have similar skills to those considered experts.

Following the idea proposed in the temporal characterization method, this approach also has a first stage responsible for reducing the initial number of developers to a group

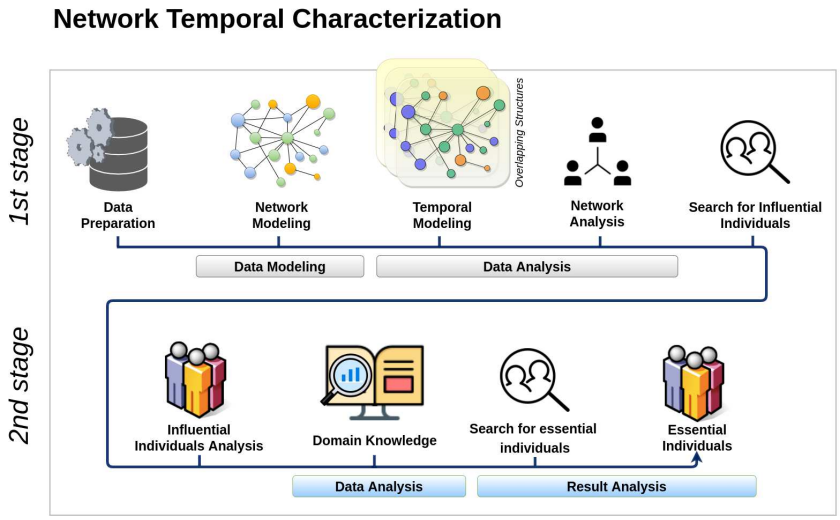

Figure 4: Network Temporal Characterization.

of the most influential individuals in the network. Considering the degree of contribution as a metric to find influential individuals, it is likely that they have a greater and more active participation in contributions to the network. They are identified by analyzing the developers' contribution degree. Therefore, studying these individuals is considered valuable when identifying potential individuals to assist in project tasks.

In the second stage, we propose an analytics approach that trains different classification models: LR, LDA, NB, KNN, and SVM, to predict whether or not individuals are requested to assist with project tasks. The models' results are compared and evaluated, and the model with the best results has its values chosen for further analysis. We focus on analyzing individuals classified as false positives (fp) because they are individuals incorrectly classified as positive. In this case, they are supposed to have similar characteristics to those classified as true positives (tp). In other words, we investigate if recommending false-positive individuals can be used as an alternative to increasing the diversity of recommended developers and reducing the workload on those who excel in the network.

3.4 Semantic-based Approach

Figure 6 shows the approach proposed to address the research challenges related to semantic analysis in expert recommendation systems. Here again, the first stage comprises the strategy of reducing the number of individuals to be analyzed to a group of more influential individuals in the network, according to their contribution degree.

We employ a semantic analysis approach in the second stage to extract meaning from data. First, we create a taxonomy to provide the categories by which a given entity within the given context can be described. The taxonomy is created using both domain knowledge and information obtained in the first stage (syntactic analysis step).

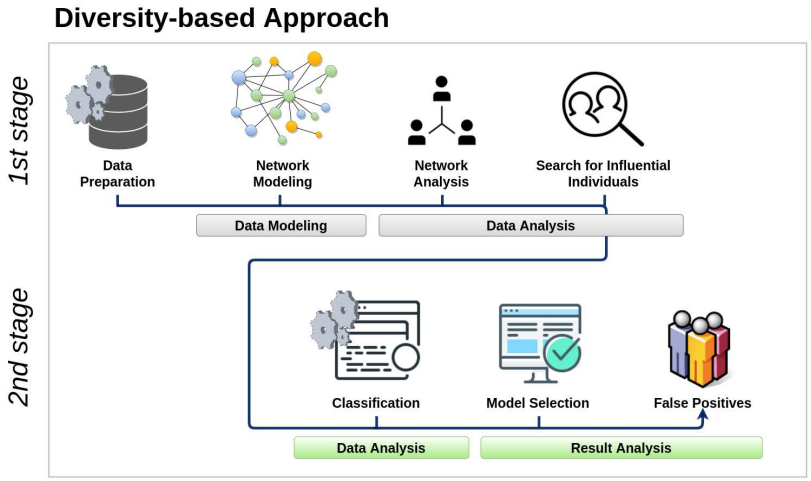

Figure 5: Diversity-based Approach.

Next, we use an ontology to explore the semantics behind the collaborative context in GSD projects. The ontology is constructed following the Methontology Framework (116) and its steps will be presented in Chapter 7. The ontology takes a step further from the taxonomy, allowing us to classify and specify the entities by describing the data structure as classes, relationships, and constraints that act on the concepts and entities. Therefore, we use ontology to infer implicit connections between experts and expertise in order to create a knowledge graph. A knowledge graph represents a semantic network and can be used to identify, rank, and query individuals with specific skill levels.

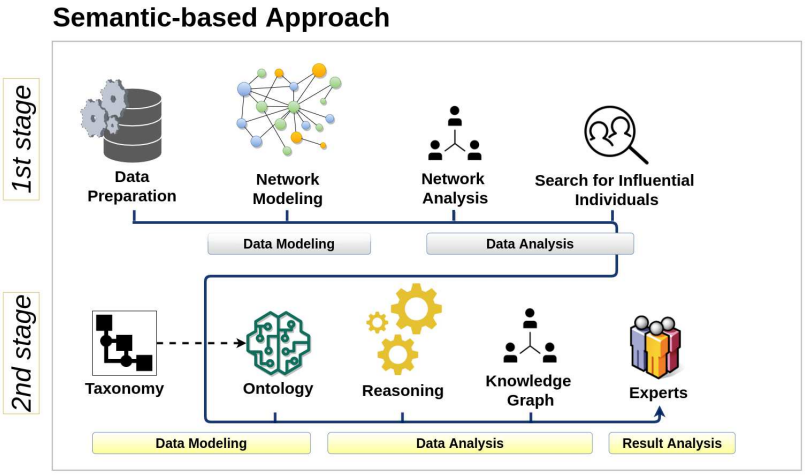

Figure 6: Semantic-based Approach.

3.5 Network Partitioning Strategy

Finally, this subsection presents the network partitioning strategy (see Figure 7), which is proposed to address the research challenges related to data volume. Our focus is on improving the processing time of community detection methods in social networks: the

syntactic step we performed in the previous cases (stage 1). Although the syntactic step is a common step in our architecture modules, it consists of costly operations, which makes it crucial to improve its performance. Therefore, we propose a new network partitioning algorithm and use it to reduce the structural complexity of large networks, dividing them into balanced subnetworks to be processed in parallel. Lastly, the smaller structures are analyzed in parallel, and the partial results are merged to produce the final result.

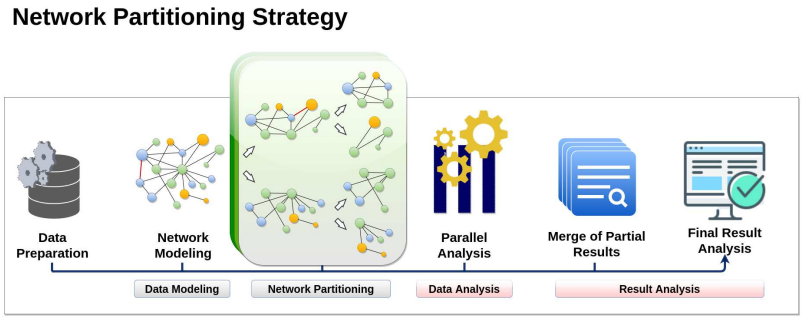

Figure 7: Network Partitioning Strategy.

To evaluate our proposal, we applied our architecture to a collaborative network of developers working on GSD projects. Therefore, in the next chapter, we present the data as well as describe the social network model used in this study.

3.6 Social Network Data and Model Description

GSD is supported by many platforms that enable collaboration between developers. GitHub (117) is one of the biggest code hosting platforms, responsible for millions of software development projects. It offers Git functionality, such as distributed version control and source code management. The platform has been a target of recent studies (118, 119) since it allows the integration of large amounts of data through its RESTful API. Furthermore, GitHub offers collaborative features[1] supported by pull-based methods (120), which works with *Issues*, and *Pull requests*. Issues are trackers of bugs, enhancements, requests, and ideas, while pull requests are a mechanism for developers to notify team members that a feature or fix is ready. With this kind of system, a developer can let everyone know that they can review the code, providing a forum to discuss the implementation of the proposed feature.

Therefore, in order to evaluate our approach in a real-world scenario, we analyzed some of the most starred($\star$) projects[2] on GitHub: **Node.js**[3] (86,445 $\star$), **Kubernetes**[4] (86,761 $\star$), and **Symfony**[5] (26,626 $\star$). These are popular projects with new features being

[1] https://github.com/features
[2] https://gitstar-ranking.com/repositories; accessed 24 Mar 2022
[3] https://github.com/nodejs/node
[4] https://github.com/kubernetes/kubernetes
[5] https://github.com/symfony/symfony

continuously developed and supported by a large community with various active developers worldwide. The projects are briefly described below:

- **Node.js** is an open-source, cross-platform JavaScript runtime environment that lets developers use JavaScript to write command-line tools. It is primarily used for non-blocking, event-driven servers due to its single-threaded nature. *Initial release*: May 27, 2009; *Stable release*: March 22, 2022.
- **Kubernetes** is an open-source container orchestration system for automating computer application deployment, scaling, and management. It aims to provide a platform for automating deployment, scaling, and application containers across clusters of hosts. *Initial release*: 7 June 2014; *Stable release*: December 16, 2021.
- **Symfony** is an open-source PHP web application framework (a set of reusable PHP components/libraries) that aims to speed up the creation and maintenance of full-featured web applications. *Initial release*: 22 October 2005; *Stable release*: December 29, 2021.

3.6.1 Dataset Description

All data used in this research were obtained through the GitHub RESTful API. However, it is important to emphasize that, as the work progressed, new data were obtained and used in our analysis. The data obtained from the projects correspond to the period from January 2014 to June 2018.

Table 1 shows the different tables used in the experiments and the number of records in each table. The different tables are described as follows. (i) *User*: table with data about distinct developers contributing to the projects; (ii) *Pull request*: table with data about pull requests that were opened in projects; (iii) *Review comments*: table with data of comments related to proposed changes in pull requests; (iv) *Discussion comments*: table with data of comments in discussions forums; (v) *Label*: table with distinct tags used to categorize pull requests.

Table 1 – Dataset's Tables and Number of Records.

Table	**Value**
Users	15,762
Pull Requests	79,375
Review Comments	265,714
Discussion Comments	226,628
Labels	425

3.6.1.1 Network Model

A social network model can represent different types of interactions depending on the connections between individuals we want to analyze. Therefore, seeking to explore the collaboration context in GSD projects, we propose a collaboration network model created from implicit contribution relationships between developers.

As illustrated in Figure 8, a contribution relationship is created from User v_i to User v_j when user v_i creates a review comment on a pull request created by user v_j. Therefore, we say that v_i contributes to v_j. Review comments were chosen as they can be seen as contributions that involve specific knowledge since they are comments related to code changes proposed in pull requests. Conversely, other collaborative features on GitHub, such as discussion comments, were not considered at this point when modeling the network because they are not explicitly related to any specific knowledge. Furthermore, it is important to highlight that the proposed collaboration model will be used in the data modeling stage in the other approaches developed in this work. Next, we formalize the proposed model.

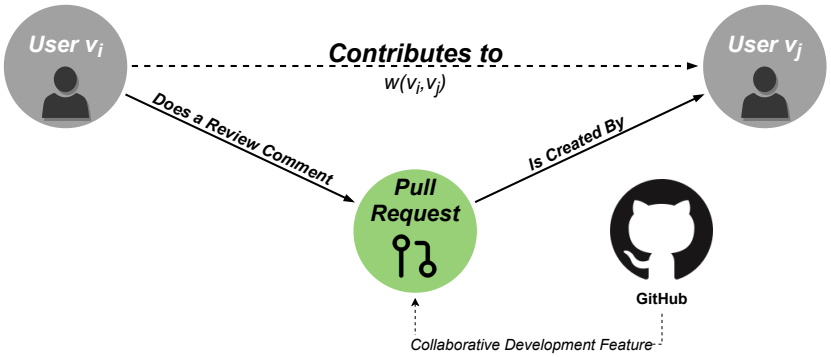

Figure 8: Collaboration Network Model.

Collaboration Model: Let $B = (U_b, P_b, E_b)$ be a bipartite bidirectional graph (see Figure 9), with U_b, P_b, and E_b denoting the set of users, pull requests and edges connecting the nodes, respectively. Each edge can be labeled with: 'created_by' (from P to U), or 'created_review_comment' (from U to P). Also, each node $p_i \in P_b$ can have N input edges, with N equal to the number of review comments, however, it can have only one output edge (only one author). In order to model the collaboration network, we extract a bidirectional graph $D = (V_d, E_d)$ from B, where $D_d = \{d_0, d_1, ..., d_n\}$ is the set of developers (nodes), connected by a set of edges $E_d = \{e_{ij}, ..., e_{km}\}$, representing contributions between pairs of developers. A relationship $e_{ij} = (v_i, v_j, w_{ij})$ is established when a developer v_i creates a review comment on developer v_j's pull request. In addition, the weight value w_{ij} is calculated by summing all v_i's review comments on all pull requests opened by developer v_j.

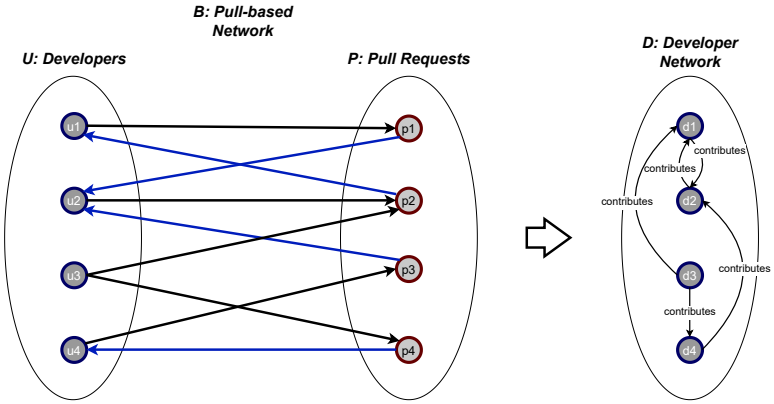

Figure 9: Developer Network.

3.6.1.2 Evaluation Structure

The data and the social network model presented (in this chapter) in figure 8. were used in developing historical researches that evaluate this work's proposal. Considering the four pillars of the proposed solution, we divided the evaluation into four historical researches, presented in the following chapters.

Historical Research 1 was conducted to verify the proposal's feasibility regarding the temporal characterization of the social network. In historical research 1, a clustering algorithm was used to identify experts, and a temporal analysis was conducted to analyze the evolution of the network over time. We also use domain knowledge to understand which experts are responsible for the growth of the network.

Historical research 2 verifies the proposal's feasibility to identify alternative specialists, i.e., to reduce the workload of some developers who are widely requested at certain times. We used clustering and classification algorithms in this case study to identify alternative specialists. The results were analyzed to see if the identified developers could be considered alternative experts.

In Historical researches 1 and 2, we used information about the social network structure for the analyses carried out. In historical research 3, a semantic analysis is conducted to enrich the data through the proposal of an ontology. This ontology was instantiated using GitHub data and the network structure described in this chapter.

Finally, historical research 4 aims to investigate the structural complexity reduction in large graph arrangements, considering the graph partitioning problem as a fundamental algorithmic operation on large-scale parallel methods (121) and its impact on social network analysis. Considering that we seek to address the data volume in social networks, here we use a different dataset, with a greater number of entities and connections: the **Digital Bibliography & Library Project (DBLP)** database (122). As this dataset will only be used in this historical research, it will be described later in Chapter 7.

4 Historical Research 1: Temporal Characterization

Considering time dimension is essential in expert recommendation systems as people's interests constantly change, which may impact their ranking in recommendations. In this chapter, we carry out experiments to investigate how the proposed temporal characterization approach contributes to the search and recommendation of experts. The temporal characterization is an instantiation of the proposed architecture with a focus on syntactic analysis of the network. We consider temporal information to model the network over subsequent periods in order to investigate the evolution of the network formed among developers who collaborate to achieve project goals. To achieve our goal, we seek to understand what defines developers as essential for the evolution of the network.

4.1 Network Characterization

This section presents the analysis and results related to the temporal characterization of the network. The data is analyzed using the proposed temporal network approach (see Figure 4). In the first stage, we perform syntactic analysis on the overlapping structures in order to find influential individuals in the network. In the second stage, we use domain knowledge (semantics) to enrich our investigations in search of individuals considered essential for the evolution of the network.

In order to reduce the initial number of developers to a group of individuals with greater relevance in the network, we used a density-based clustering algorithm to analyze the network and identify influential individuals in communities. A density-based clustering algorithm was chosen as we want to identify developers who are well connected, which are represented by core nodes. Also, since we don't know the number of groups, we need an algorithm that doesn't require this number as a parameter. Density-based clustering methods usually have two main parameters: (*eps* and *minPts*). The parameter *eps* defines the influence (connection weight) for a node to be considered an influencer of its neighbors, and *minPts* indicates the minimum number of neighbors that a node must have to be considered a core node. Therefore, we can see core nodes as influential developers because of the minimum number of connections they must have.

Since we want to identify influential individuals in a collaborative network, we used NetSCAN to search for core nodes. NetSCAN (32) is density-based and has been used in previous studies to detect overlapping communities in social networks. In addition to the parameters mentioned above, NetSCAN also has an optional parameter (*radius*), which allows the search for elements influenced by the *core* in a greater depth.

After preliminary analysis of the network, we set the NetSCAN parameters to be $eps = 1$ and $minPts = 4$. The $eps = 1$ corresponds to the out-degree mode value above zero in the network, and $minPts = 4$ corresponds to the out-degree average value rounded

down in the network. With these values, we give more value to developers who contribute to a different number of individuals (minPnt) than the weight of each relationship (eps). Thus, we consider influential those with many contributions to different people greater than or equal to the average number of contributions to different people in the network. The third parameter (radius) was kept with its default value equal to 1.

Considering the data presented in subsection **3.6.1**, we generated overlapping structures to analyze the network over consecutive periods. Although all obtained data correspond to the period from January 2010 to June 2018, given the data quality, we opted to work with the most recent period, **from January 1, 2017, to June 30, 2018**. We modeled each month as a contribution network and ran the NetSCAN algorithm on each network to find core nodes for each period. Table 2 shows the number of nodes and edges of the network for each period, as well as the number of clusters identified by NetSCAN and the nodes and cores that make up these clusters.

Table 2 – NetSCAN results in each overlapping structure

Period	**Nodes**	**Edges**	**Clusters**	**Cores**
2017-1	436	987	9	66
2017-2	494	1117	6	62
2017-3	489	1016	8	66
2017-4	477	1034	10	67
2017-5	547	1299	7	70
2017-6	551	1137	6	62
2017-7	522	1110	10	63
2017-8	595	1337	8	70
2017-9	557	1186	5	63
2017-10	704	1519	4	73
2017-11	624	1367	9	85
2017-12	489	873	11	39
2018-1	530	1043	6	59
2018-2	573	1168	12	76
2018-3	497	1032	9	59
2018-4	528	1020	8	60
2018-5	493	1012	11	60
2018-6	481	926	11	44

Once we have defined and identified influential developers as being Core Nodes, we now move further in analyzing these individuals. We seek to investigate the role of Core Nodes as influential individuals, analyzing their degree of contribution over different periods in the network.

Considering the temporal characterization goal addressed in historical research 1, figure 10 shows (A) the number of Core Nodes for each period to which they contributed and (B) the contribution degree of each Core Node. As we can see, the number of Core

Nodes contributing does not change much over the months until 2017-11. At that moment, the number of Core Nodes contributing reaches a maximum in 2017-11, followed by a minimum in the next month, 2017-12. Furthermore, although 2017-11 is the month with the most Core Nodes contributing, Figure 10-B shows that 2017-11 has less intense contributions (darker colors) than in the previous month (2017-10), which has the most specific contributions; Core Nodes contributing more intensely (lighter colors).

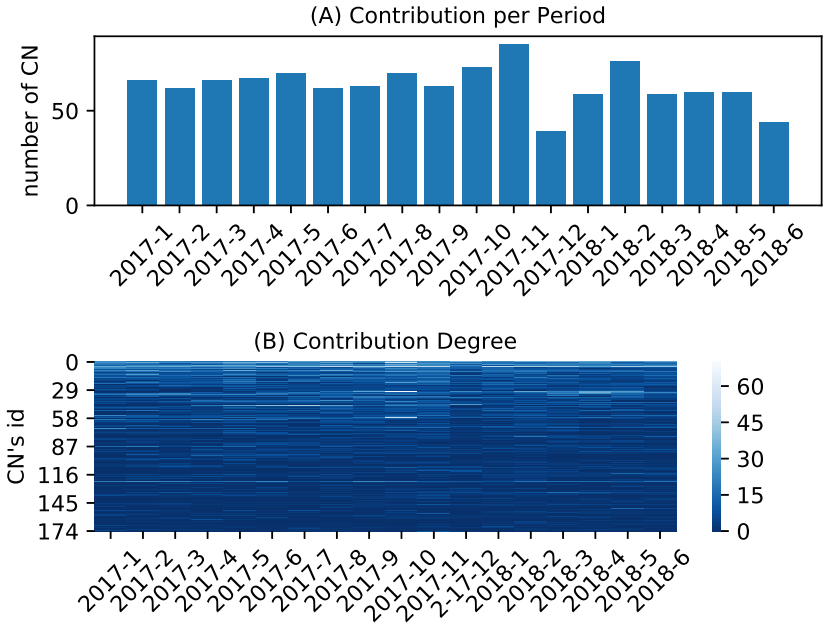

Figure 10: (A) the number of Core Nodes contributing in each period. (B) Core Nodes' contribution degree per period.

Figure 11 presents boxplots in order to further illustrate the Core Nodes' contribution degree per period. It shows that 2017-10 is the month with the highest number of individual contributions, indicating that this is a month that demanded a greater degree of contribution from some specific Core Nodes (outliers). These analyses may indicate the growth of project demands that require specific expertise, with emphasis on 2017-10, where there is the most significant number of individual contributions.

It is also possible to observe another pattern, where outlier nodes contribute more and more from 2017-1 up to 2017-10. Although 2017-11 is the period with the most Core Nodes contributing when compared to the other months, this is also a period where only one individual stands out with their high contribution degree. Consequently, this draws our attention to a question (Q1) to be investigated in the domain knowledge step (next subsection):

Q1: What characterizes periods with a greater number of individual contributions?

Considering all the Core Nodes identified in the analyzed periods, in Figure 12, we illustrate (i) the number of Core Nodes related to the number of months they contributed (i.e., how many Core Nodes contributed in 1, 2, 3, ..., 18 months), and (B) who are the

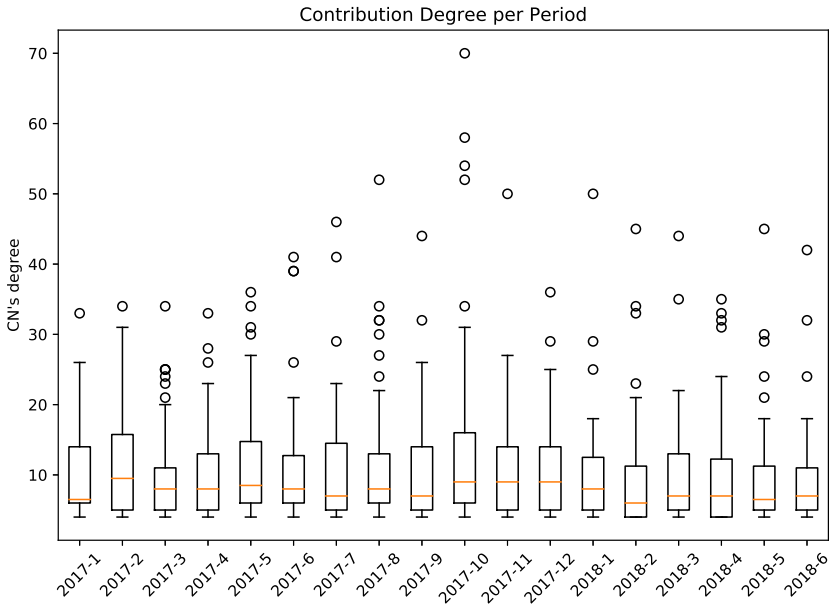

Figure 11: Boxplot: Core Nodes' contribution degree per period.

developers identified as Core Nodes in each month (the y-axis on the graph represents each individual's ID). By analyzing the graphs, it is possible to see that many Core Nodes participated in just a few months. Most of them contributed in just one month, while others contributed for short periods, such as 2, 3, or 4 months. For longer contribution periods ($>$ 4 months), the number of Core Nodes contributing stabilizes. Furthermore, we can see that some have contributed over several months and probably have a more significant role in the network. Therefore, we also employ domain knowledge to answer Q2 in the next subsection:

(Q2) How can Core Nodes be characterized as those who contribute during all periods in the projects?

Finally, we also analyzed sequential periods of contribution and idle periods of contribution. Similar to Figure 12-A, Figure 13 contains graphs showing the number of Core Nodes related to (A) sequential periods of contribution, (B) the maximum sequential period of contribution, (C) sequential periods of inactivity (idle periods), and (D) the maximum sequential period of inactivity. It is worth explaining that both 13-A and 13-C represent the totality of cumulative periods, i.e., a developer who contributed for a period, stopped, and contributed again after some idle period, counts in both periods. According to 13-A and 13-B, many Core Nodes have contributed for one month or short sequential periods, and a few have contributed for longer sequential periods. Furthermore, 13-C and 13-D show that some Core Nodes were absent for 12 and 13 months between periods of contribution. Therefore, we extend the discussion of the next subsection with a final question, Q3.

(Q3) What characterizes Core Nodes who make contributions in specific periods?

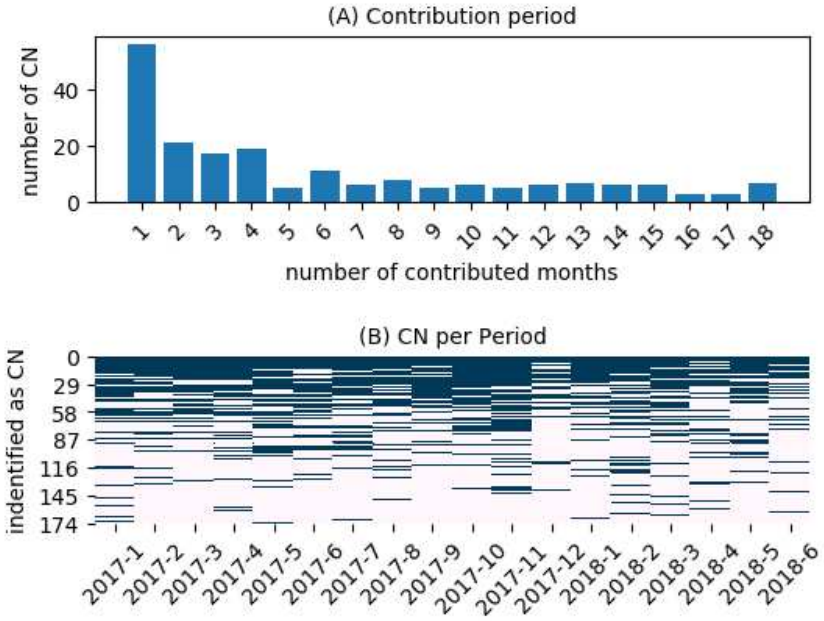

Figure 12: (A) Number of Core Nodes who contributed by the number of contributed months. (B) The ID of individuals identified as Core Nodes in each period.

In conclusion, we were able to identify influential individuals in the network by analyzing their contribution degrees. Furthermore, some may be essential for the network evolution due to their high contribution degree in specific periods. However, despite some individuals' high contribution degree, it is impossible to affirm that they are essential for network evolution by analyzing the network structure alone. Therefore, in the next subsection, we include domain knowledge in our analysis to help us search for essential developers in the network. We seek to answer questions Q1, Q2, and Q3, which can help us to identify these individuals.

4.2 Analysis using domain knowledge

At this stage of the temporal characterization approach, we extend our investigation by using domain knowledge in order to characterize individuals considered essential for network evolution.

Projects hosted on GitHub repositories are equipped with a label system that provides several tags for annotating pull requests and issues. Tags are responsible for improving access and understanding of documents, capturing progress history points, marking release versions, and assisting users in navigating to critical parts of the project (123). Likewise, developers are also related to tags that categorize the pull requests they contributed to. Therefore, we use domain knowledge to explore the tags responsible for labeling pull requests.

Figures 14, 16, and 15 show Symfony, Kubernetes, and Nodejs projects, respectively, with the monthly distribution of the main tags used in each project. Also, the graphics

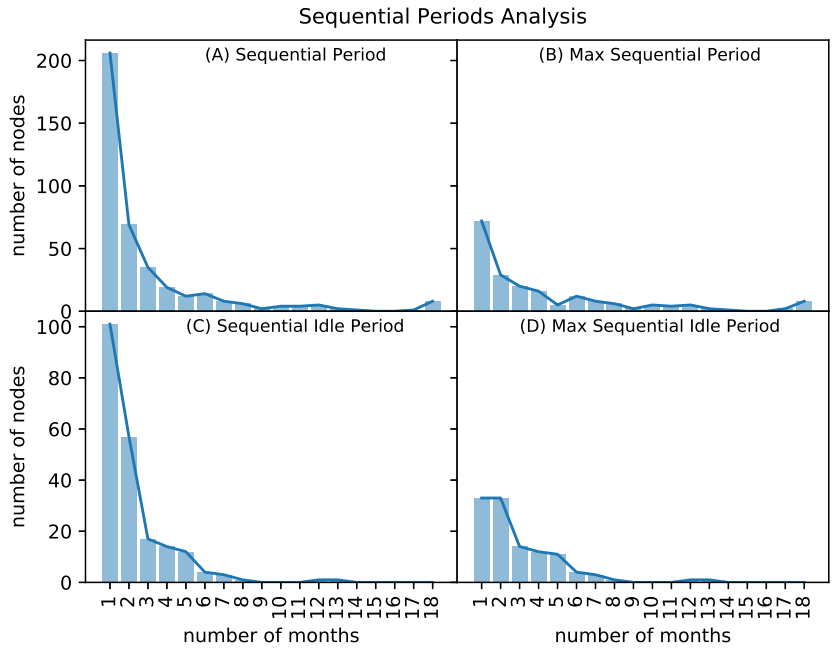

Figure 13: (A) Number of Core Nodes related to sequential periods of contribution. (B) The number of Core Nodes related to the max sequential period of contribution. (C) The number of Core Nodes related to idle periods of contribution. (D) The number of Core Nodes related to the max sequential idle period of contribution.

show trend lines that may represent developing directions that projects are heading, e.g., with this information, it is also possible to observe how projects are evolving, indicating higher stability or a greater development effort.

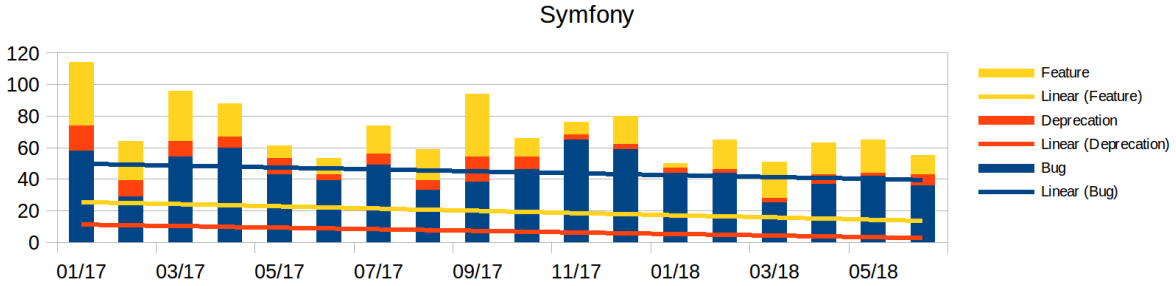

Figure 14: Symfony Project: Tag Distribution.

In the Symfony project (Figure 14), *bug* and *feature* are the most used tags. We can associate these two tags with efforts related to corrective maintenance and the development of new features, respectively. In addition, a smooth fall of all trend lines indicates project stabilization.

On the other hand, Node.js (Figure 15) appears to be growing due to the increase in the number of tags related to evolutionary maintenance, such as *lib/src* and *doc*. Node.js has a larger number of distinct tags distributed over the periods, demonstrating the importance of various areas being developed within the project. Despite this, the project

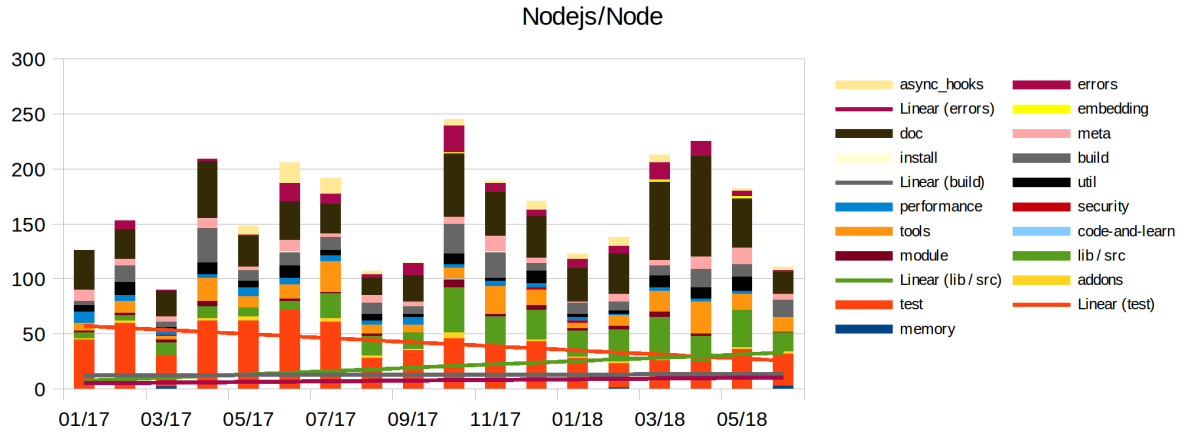

Figure 15: Node.js Project: Tag Distribution.

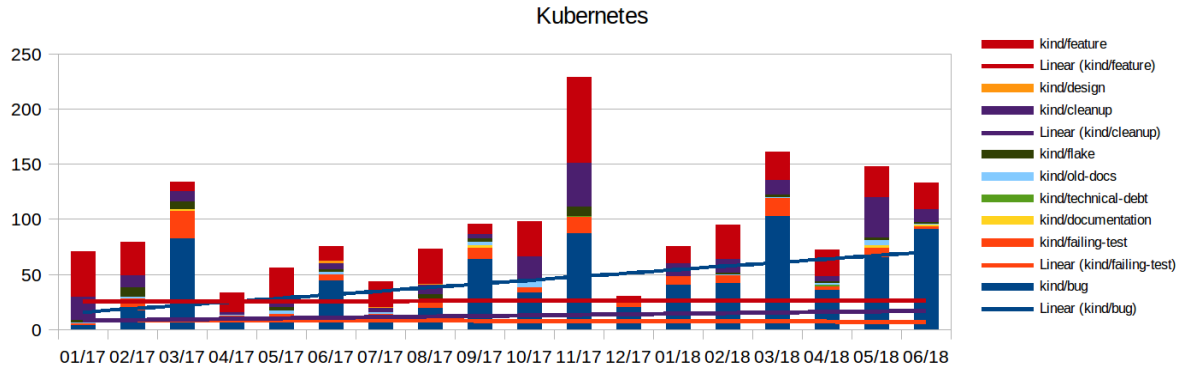

Figure 16: Kubernetes Project: Tag Distribution.

also shows signs of stability due to the decrease in the number of test-related labels, indicating the consolidation of its main components.

Lastly, Kubernetes (Figure 16) shows an upward trend in the number of bugs over the months. It is possible to observe a particular situation in 2017-11 where there is a huge amount of PRs mainly labeled with ‘feature’, ‘cleanup’, and ‘bug’ tags. Similar to Symfony, the appearance of these tags in Kubernetes PR seems to be strongly associated with the development of new features.

These results show that periods with a large number of individual contributions (identified in the structural analysis of the network), such as 2017-10 and 2017-11, match with periods with a huge amount of PR labeled according to the development of new features or the appearance of bugs. Therefore, answering Q1, the periods with greater individual contributions can be characterized by periods of high specific demand in projects.

In the projects, we also identify different developers’ profiles. These contributed over all periods, while others made specific contributions. We observed that these individuals have different contribution profiles characterized by topics of contributions. In this sense, we seek to explore the Core Nodes considering their specialties and the main technologies

adopted in each project.

We randomly chose seven influential Core Node developers to be further analyzed. Table 3 describes these developers considering: (i) **ID**; (ii) **Period** of contribution, with most individuals having contributed during all periods and two in specific periods; (iii) **Project** on which they worked; Furthermore, other information related to PR they worked, including (iv) **Kind**; (v) **Technology**; and (vi) **Size** (code lines), which is information available only for the Kubernetes project. We further extended the analysis of two developers who contributed in all periods in Kubernetes: Dev 980082 (Figure 17) and Dev 730123 (Figure 18), by illustrating the tag distribution of the PRs they participated in.

Table 3 – CNs Profiles

Core ID	Period	Project	Kind	Technology	Size
730123	all	Kubernetes	bug, clean	sig/api-machinery	size/L,size/M
980082	all	Kubernetes	bug	sig/api-machinery, sigh/auth	size/M, size/XS, size/L
439929	all	Nodejs	lib/src, errors	C++,HTTP2	-
2512748	all	Nodejs	test	C++	-
243674	all	Symfony	bug	DependencyInjection, Frameworkbundle	-
826111	2017-3,2017-4	Kubernetes	Feature	sig/node,sig/storage	size/L
610090	2017-7,2017-10,2018-2	Symfony	Bug,Deprecation,Feature	TwigBridge,HttpKernel	-

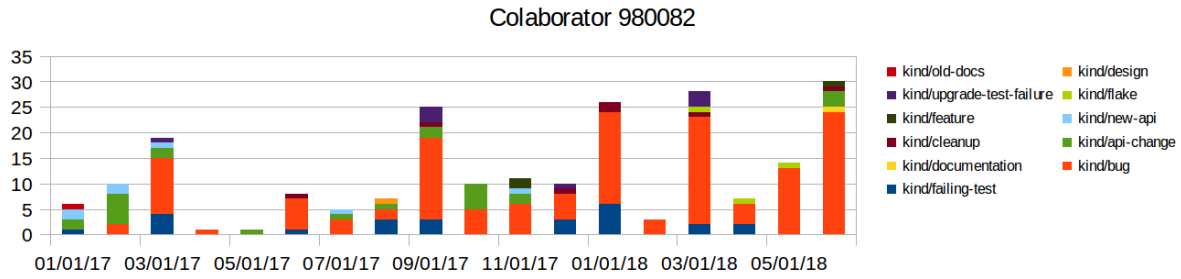

Figure 17: Dev 980082: Tags distribution.

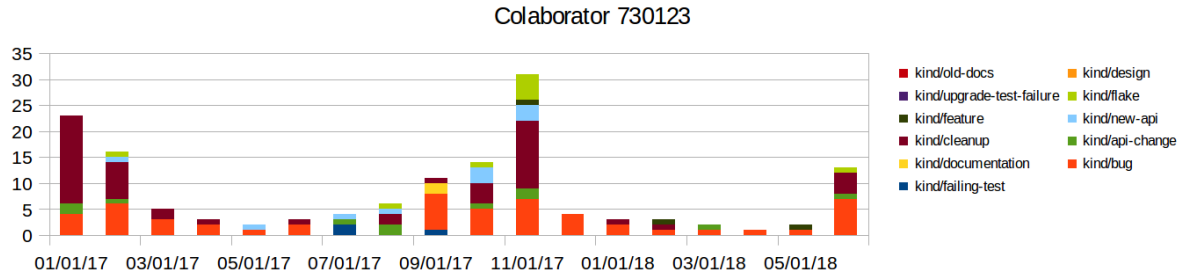

Figure 18: Dev 730123: Tags distribution.

In Table 3, we see that these bugs may be mainly related to small/medium size contributions in *apimachinery*[1], a library for servers and clients to work with Kubernetes API infrastructure without direct type dependencies.

Figure 17 shows that Dev 980082 has a contribution pattern that meets demands related to bugs in Kubernetes, indicating that the author contributed during periods of

[1] https://github.com/kubernetes/apimachinery

significant demand related to bug fixing. In the table 3, we see that these bugs are related to small-large changes and the *apimachinery*, a library for servers and clients to work with Kubernetes API infrastructure without direct type dependencies. Also, according to Figure 18, Dev 730123 was a key individual in contributions related to cleanup, specifically in periods when the project had a great demand on that topic.

Moreover, by analyzing other Core Nodes who contributed in all periods, we discovered that their contributions are usually related to topics concerning the projects' principal technologies, such as C++ in Node.js and corrective maintenance in Symfony. Therefore, in answering Q2, we can characterize individuals who contributed during all the periods as those who work according to the demands of the projects' main technologies. The most frequent contributors work more comprehensively, assisting with specific needs such as resolving bugs, performing software builds, and they have a large number of approved issues.

On the other hand, several developers contributed at only specific periods, such as Dev 826111 and Dev 610090. Dev 826111 with large contributions between 2017-3 and 2017-4 in the Kubernetes project, working on new features related to components that support the controlled interactions between pods and host resources[2], and block storage files[3], according to their related technologies. In addition, Dev 610090 contributed in isolated periods where there was a need to develop features and fix bugs regarding components like TwigBridge[4] and HttpKernel[5] in the Symfony project.

Also, as exemplified in table 3, we could observe that individuals who work at particular moments in the projects tend to contribute to issues with low priority. They usually operate with more than one project technology, although there is always one technology that stands out, i.e., the one they know best. In addition, they can probably be characterized as inexperienced individuals who work with smaller code, which is often the recommendation of many projects so that new members can become more familiar with the demands of the project. Accordingly, Q3 is also answered.

Finally, we can also answer the first secondary research question **SRQ1** presented in chapter 1. By answering questions Q1, Q2, and Q3, we could characterize individuals considered essential for network evolution.

[2] https://github.com/kubernetes/community/blob/master/sig-node/README.md
[3] https://github.com/kubernetes/community/blob/master/sig-storage/README.md
[4] https://github.com/rcrowe/TwigBridge
[5] https://github.com/symfony/http-kernel

5 Historical Research 2: Diversity-based Analysis

In the previous Chapter, we used the proposed temporal characterization approach to analyze a collaboration network modeled over consecutive periods. We analyzed its evolution by considering structural aspects and exploring domain knowledge to expand our investigations. From the analysis, we were able to identify influential individuals as well as those who can be considered essential for network evolution. However, it was observed that the high demand in specific periods causes some developers to be highly requested. Therefore, this section presents the diversity-based analysis approach as a solution to the work overload problem.

For the analysis, we keep working with the data from projects on GitHub, introduced in subsection **3.6.1**. In addition, as we do not intend to study isolated periods, we extended the analyzed data to the entire period present in the database, from January 1, 2014, to June 30, 2018.

5.1 Network Analysis

The diversity-based approach also has an initial network analysis step focusing on finding influential individuals. This is done to reduce the number of developers examined to a smaller group of more relevant individuals in the network. For this purpose, we used the NetSCAN (*eps = 1, minPts = 4, radius = 1*) clustering algorithm as described in the preceding Chapter to search for core nodes.

In Table 4, we present information about the number of (i) nodes, (ii) edges, and (iii) connected components in the network. Also, we present data related to the results obtained with the clustering that includes the number of (i) clustered nodes, (ii) found clusters, and (iii) nodes identified as core nodes.

Table 4 – Values that describe the network and clustering results.

Nodes	**Edges**	**Components**
5591 (5536*)	24563	28
Clusters	**Clustered Nodes**	**Core Nodes**
54	1853	900

*Biggest connected component.

5.2 Classification

In GSD, requested individuals are those who are usually asked to assist with some specific project tasks. In this sense, we seek to work with classification models to find one that we can use to classify experts. Therefore, five popular algorithms were chosen to classify the developers as **required developers** or not required ones: Logistic regression

(LR), Linear Discriminant Analysis (LDA), Naive Bayes (NB), K-Nearest Neighbors (KNN), and Support Vector Machines (SVM). These algorithms were chosen because they are easy to use to implement classification problems since they are available in python using the Sklearn library.

Furthermore, we defined the target class according to the binary attribute: *Requested*, which is a field of the User table. This attribute defines whether a developer has already been requested, at least once, to review a pull request. Moreover, considering the network collaboration context, four other numerical attributes were chosen and calculated to compose the instances to be analyzed by the models. The attributes are the total number of (i) pull requests, (ii) review comments, (iii) discussion comments, and (iv) distinct tags that a developer is associated with.

In the context of classification models, a *false positive* is an error in a binary classification method where a test result incorrectly indicates a condition. For instance, in expert classification systems, false positive elements may represent individuals incorrectly classified as experts. However, as these individuals are classified as positive, it may be that, among the true negatives, they are the ones that have characteristics most similar to the true positives. Therefore, in the next subsection, we seek to investigate whether the recommendation of false positives is adequate as an alternative to increasing the diversity of individuals that are required to assist in project demands.

5.3 Results

We used the 900 individuals identified as core nodes in our analysis since they are a group of more relevant developers in the network and are considered influential. In order to evaluate the classification models, we adopted the k-fold cross-validation technique (124), a re-sampling procedure used to evaluate machine learning models. The procedure has a single parameter called k, which refers to the number of interactions that a given data sample is split into test and training data sets. In this regard, we took the value k equal to 10, and for the size of test and training data, we used the percentages 30% and 70%, respectively. Furthermore, the same s_i seed is used during the i-th interaction of each algorithm so that we can compare the models properly.

The models were compared and analyzed to choose the one that presented the best results for our problem. Figure 19 illustrates boxplots representing the accuracy scores of each model in the ten runs performed. The accuracy was chosen as the evaluation metric since it defines the proportion of true results (true positives + true negatives) over the total number of instances. Also, we are looking for a model with high accuracy and one with false positive values. The main idea is that the algorithm should be good for finding experts (true positives) and have the flexibility to find people with characteristics similar to experts (false positives).

In analyzing the boxplots (Figure 19), we see that the LDA model obtained the highest accuracy score when compared with the results obtained by the other models. The average accuracy score of the LDA was 76%, while the others were: LR at 59%, NB at 58%, KNN at 55%, and SVM at 54%. Furthermore, the LDA model was also able to classify some instances as false positives. Therefore, we chose one of the results of the LDA model to be investigated further. The instance chosen was the one that achieved 80% accuracy.

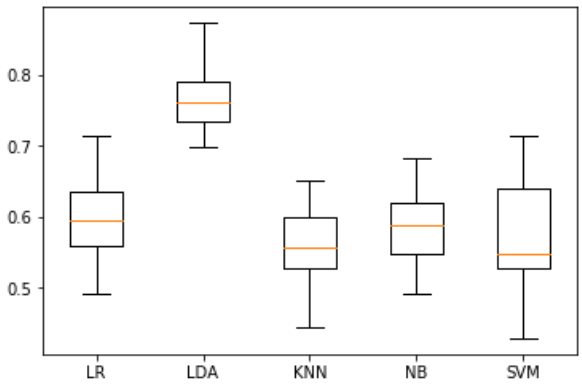

Figure 19: Average classification accuracy for five classifiers.

Figure 20 presents the confusion matrix for the chosen LDA. The 0 value represents the predicted condition, meaning that values that fall into this class are classified as required individuals, while the 1 represents individuals classified as not required. According to the confusion matrix, 49 individuals were correctly predicted as required (true positives), and 172 were correctly predicted as not required (true negatives). Since the test samples have 275 individuals, this LDA model reached 80% accuracy (221/275). Furthermore, 39 individuals were identified as false negatives, while 15 were incorrectly predicted as positives (false positives). As discussed, in order to reduce the request overload to the same group of influential developers on the network, we further investigate the 15 individuals classified as false positives.

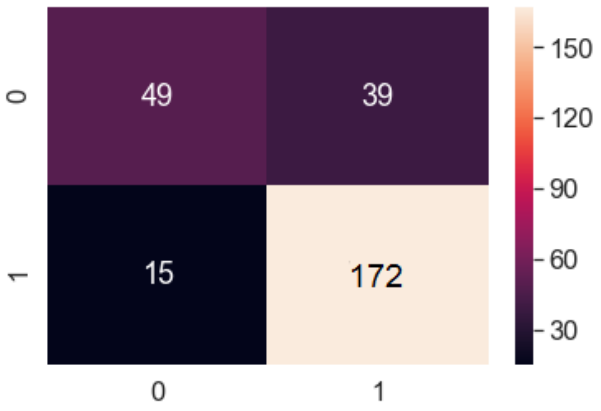

Figure 20: Confusion matrix.

When searching on GitHub, we found four project-related roles: *Member*, *Contributor*, *Collaborator*, and *None*. As illustrated in Table 5, we matched each of these roles with the individuals classified as false positives to specifically identify what role they play in the network. As we can see, most of those in the group of false positives are contributors, those who have previously committed to repositories (different from collaborators: those who can pull - read - the contents of the repository and push - write - changes to the repository). In addition, one of the false positives was also identified as a member of a project or organization.

Table 5 – Developers' Role

False Positives	None	Contributor	Collaborator	Member
15	0	14	0	1

To evaluate whether the individuals in the false positive group not only make isolated contributions but are also active in projects, we examined the contribution period of each of these developers; this is illustrated in Figure 21. The blue and green dots represent the first and the last year of contribution to the projects, respectively. We can see that many of the individuals in the false positive group contributed over several years, having significant and active activity in the network. Some worked throughout all observed periods, while others were still active until the last year (2018). These results reinforce the importance of considering those individuals identified as false positives due to their role and contribution time within the projects, even though they were not necessarily asked to help with the project tasks.

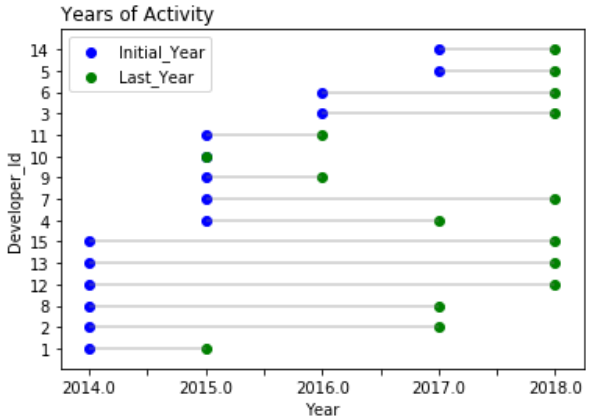

Figure 21: False positives' contribution timespan.

In graph theory, centrality indicators assign ranks to nodes within a graph corresponding to their position in the network. Therefore, as the last analysis metric, we also calculated the centrality indicators for the false positive individuals. Three boxplots

are shown in Figure 22, containing the comparison of the Closeness, Betweenness, and Page Rank centrality measures. The graphs highlight the scores obtained by only the false positive group compared to the score of all the other nodes in the network.

When analyzing the boxplots, we can see that all the false positives have Page Rank and Betweenness scores that define them as outliers. The high Betweenness values show that these individuals operate as information bridges between different groups in the network. Likewise, the Page Rank values indicate the increased relevance of false positive individuals due to the importance and influence of the nodes they are connected to. Besides that, all proximity scores of false positives are above the third quartile, indicating that these are individuals with a high degree of contribution to the network; they are among individuals with the shortest distances to all other nodes, capable of disseminating information efficiently through the graph.

Thus, considering the adopted metrics, we can say that the individuals classified as false positives are among the ones with the highest centrality scores on the network. In addition, when comparing with the results presented in Figure 21, even the false positive individuals that contributed for a short period to the network have significant measures of centrality, demonstrating their importance in the period they worked.

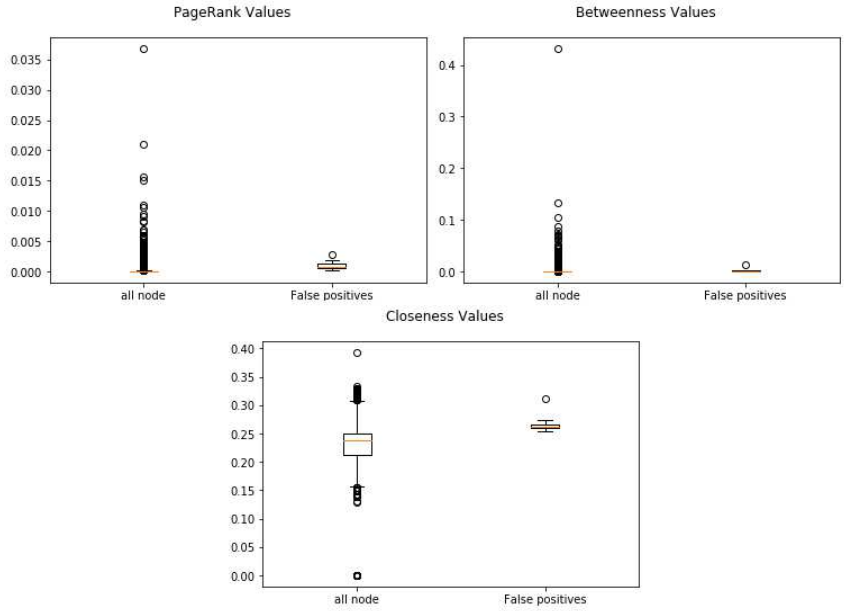

Figure 22: Centrality Analysis.

Finally, recommending the individuals identified as false positives to assist with project tasks can be used to reduce the workload of highly requested individuals. Therefore, we can answer the second secondary research question **SRQ2** since we were able to implement an approach that allows increasing the diversity of recommended developers.

6 Historical Research 3: Semantic Analysis

In Chapter 5, we identified essential individuals by exploring the network evolution. In addition, the analyses allowed us to identify some overworked developers due to the high number of requests they receive to contribute to tasks in projects. In Chapter 6, we extended our investigations by proposing a diversity-based approach to identify additional individuals who could also be recommended as alternatives to overloaded experts since these other individuals had similar skills to those considered experts.

The studies carried out so far are considered syntactic in their analyses because they focus on the network's structural investigation. However, there is no meaningful knowledge discovery without attributing meaning to data in the context in which it is embedded. Therefore, in this Chapter, we propose a semantic analysis approach to enhance our analysis of the data on individuals. Specifically, we create an ontology to extract topics from keywords in order to discover implicit relations between developers and their expertise. We also consider temporal aspects in order to investigate how the developers' interests change over time.

The data from the three projects on GitHub was used for the analysis. We conducted a preliminary ontology construction based on all projects. However, as already illustrated in Chapter 5, we mapped the terms available in the projects and observed that the projects are very varied in terms of “tags"and “labels". In this sense, we would have to create specific taxonomies for each project. We realize it is important to define a very detailed taxonomy to better understand the terms and build a higher-quality ontology.

Therefore, we focused our study on the project Node.js. As introduced in subsection **3.6.1**, Node.js is an open-source, cross-platform JavaScript environment and one of the most starred repositories on GitHub. It is supported and continuously developed and maintained by a large and active community, with various developers contributing worldwide. We used GitHub RESTful API to acquire more recent data and extended the analyzed periods from January 1, 2016, to July 01, 2020.

Initial findings identify 1513 developers among the 3143 users, that fit the contributor role according to our proposed network model. The NetSCAN clustering algorithm was again used as a preprocessing step with $eps = 1$ and $minPnts = 4$ (see Section 4.1). Table 6 shows values describing the data, which includes the number of (i) users, (ii) contributors, (iii) pull requests, (iv) review comments, (v) distinct labels, (vi) a total of labels used in all analyzed pull requests, and the date of the (vii) first and (viii) last pull request analyzed.

Table 6 – Database Information.

Data	Info
Pull requests	21600
Review comments	75365
Initial date	2016-01-01
Last date	2020-07-01
Users	3143
Unique tags	166
Pull tags	47854

6.1 Ontology

We developed the ontology following the *Methontology* Framework (116), an accepted methodology that defines the ontology development process. The framework includes four phases: (i) **Specification**, (ii) **Conceptualisation**, (iii) **Formalisation**, and (iv) **Evaluation**. Next, we describe each phase.

6.1.1 Specification

The specification is considered an essential step in system development. Therefore, this step established the Ontology Requirement Specification Document (ORSD). The ORSD allows us to identify the ontology's knowledge and define the requirements the ontology must cover. In this document, we describe the ontology's: (i) purpose, (ii) scope, (iii) implementation language, and (iv) intended End-Users. The ORSD is shown in Table 7.

6.1.2 Conceptualization

The conceptualization phase focuses on organizing and structuring the semantic meaning of data. Therefore, we developed a taxonomy[4] to categorize the knowledge in the context addressed. We first identified and classified all available keywords related to the project Node.js on GitHub. Then, since many tags correspond to project components, we reviewed the Node.js documentation[5] to categorize the tags. According to the study and compilation of the keywords, it was possible to define six different categories: (i) Type, (ii) Module, (iii) Status, (iv) Versioning, (v) Platform, and (vi) Directory. The tags available in the Node.js' Pull Request label system were distributed into these six categories, as described in Table 8.

[4] A Taxonomy is a hierarchical structure representing the formal organization of classes or types of objects within a domain.

[5] https://nodejs.org/api/

Table 7 – ORSD

Ontology Requirements Specification Document
1 Purpose
To extract topics and specific topics of knowledge from key development terms in order to discover implicit relations between developers and expertise.
To consider temporal information to compute the individuals' change of interest over time.
To identify and rank the developers according to a given specific skill.
2 Scope:
The ontology focuses on recommending experts in the context of GSD.
The level of granularity is directly related to the competency questions that are defined in Subsection **6.1.4** (Evaluation); Table 11.
3 Language:
The ontology was implemented in OWL language[1] using the Protégé ontology tool[2] and Owlready2[3], a module that that can manage ontologies and knowledge graphs in Python.
4 Intended End-Users:
User 1. Individuals who want to know what expertise is present in a project.
User 2. Individuals who want to know what expertise a developer is associated with.
User 3. Individuals who want to know a developer's expertise level.
User 4. Individuals who are searching for experts.
User 5. Individuals who want to rank developers according to to some expertise.
User 6. Individuals who want to know how developers' interest in specific expertise has changed over time.

Further, we expanded the category *Type* into the *Topic* and *SpecificTopic* categories, as illustrated in Figure 23. This is because *Type* is associated with distinct software development knowledge, including *Testing*, *Integration*, *Project Management*, *Security*, *Performance*, *Memory*, *Diagnostics*, *Updating*, *Errors*, *Features*, and other technologies, such as *Programming Languages*. We can say that topics and specific topics represent project expertise. A topic represents a more general subject, while a specific topic represents a more specific subject, i.e., saying that a developer is familiar with the topic *programming language* is more comprehensive and not as accurate as saying that they are familiar with the specific topic, for example, *Python*.

Finally, in Figure 24, we present the taxonomy model created from two knowledge sources: the network model, as detailed in subsection *3.6.1.1*, and the keyword study. The main entities of the taxonomy are: (i) Developer, (ii) Pull Request, (iii) Keyword, (iv)

Table 8 – Tags from pull-requests labels of Nodejs Project.

Node.js Keywords
Type
build, C++, Cl/flaky test, confirmed-bug, debugger, errors, experimental, feature request, memory, meta, performance, postmortem, python, refactor to ES6+, report, security, semver-major, semver-minor, test, wasm
Module:
addons, asm.js, assert, async_hooks, async-wrap, brotli, buffer, cares, child_process, cli, cluster, console, coverage, crypto, deprecations, dgram, dns, domain, dtrace, ES Modules, events, eventtarget, fs, gyp, http, http_parser, http2, https, i18n-api, inspector, install, libuv, module, n-api/n-api-semver-major, net, npm, os, openssl, path, perf_hooks, process, promises, punycode, querytring, quic, readline, regression, repl, source maps, stream, string_decoder, timers, tls, trace_events, tty, url, util, url whatwg, V8 Engine, v8 module, V8 Platform, vm, wasi, worker, zlib
Status:
abandoned, author_ready, baking-for-lts, blocked, dependencies, discuss, fast-track, help wanted, invalid, investigating, known limitation, landed, needs-benchmark-ci, needs-ci, needs-citgm, notable-change, stalled, on hold, pending, review wanted, wontfix, work in progress (WIP)
Versioning:
backport-blocked-v.X, backport-open-v.X, backport-requested-v.X, backported-to-v.X, dont-land-on-v.X, land-on-v.X, v.X
Platform:
aix, android, arm, freebsd, ibmi, linux, macOS, mips, ppc, s390, SmartOS, windows, Windows Subsystem for Linux (WSL)
Directory:
benchmark, doc, lib/src, tools

Keyword Category, and (v) Expertise. Developer, Pull Request, and Keyword entities, as well as their respective relationships, are derived from the network model. The Keyword Category entity, including the Topic and SpecificTopic entities, are associated with the keywords characterization.

6.1.3 Formalization

The formalization phase is when we convert the conceptual model (taxonomy) into a computable model. We used the Protégé[6], a tool that uses the Ontology Web Language (OWL)[7] to define an ontology by specifying its classes, properties, and semantic rules.

In OWL, classes are interpreted as a set of individuals or objects. For example, all classes are subclasses of the class Thing, which represents a set of all individuals. Figure

[6] https://protege.stanford.edu/
[7] https://www.w3.org/TR/owl-features/

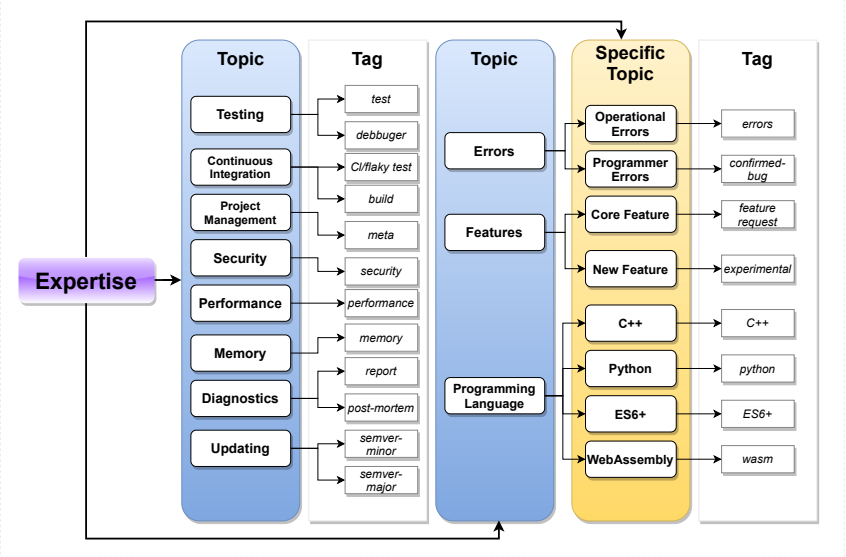

Figure 23: Topics and Specific Topics.

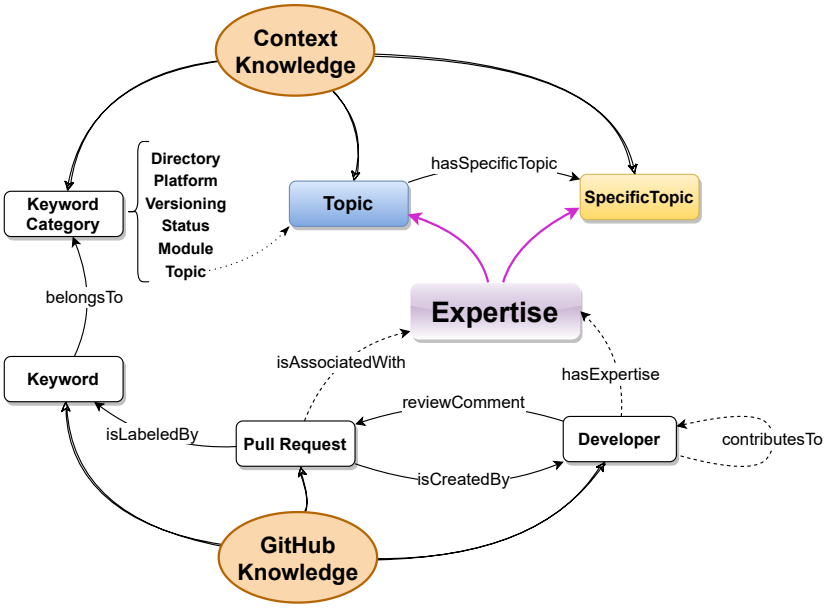

Figure 24: Nodejs project taxonomy.

25 shows the classes defined in the ontology.

Properties in OWL represent relationships between individuals. There are two types of properties: *object property* and *data property*. An object property links an object, such as an individual, to another object, while data properties link objects to a data type, such as Integer, String, or Date values. Table 9 shows the properties defined in the ontology, including their type, domain, and range.

SWRL is a language for the Semantic Web used to express semantic rules and logic. Table 10 shows the SWRL rules created to infer implicit relationships. The rules R1 and R2 are responsible for associating keywords with topics and specific topics, checking if the keyword is included in the keyword set defined by the respective topic or specific

Figure 25: Ontology classes and properties.

Table 9 – Ontology properties and their types, ranges, and domains.

Property	Type	Domain	Range
hasTopic	ObjectProperty	Keyword	Topic
hasSpecificTopic	ObjectProperty	Keyword	SpecificTopic
hasKeyword	ObjectProperty	PullRequest	Keyword
hasPullRequest	ObjectProperty	Developer	PullRequest
hasExpertise	ObjectProperty	Developer	Expertise
keywordName	DataProperty	Keyword	String
weight	DataProperty	Expertise	Integer
year	DataProperty	PullRequest	Date

topic. Since pull requests are also related to keywords, we use R3 and R4 to associate pull requests to the same topics and specific topics associated with the keywords they are related to. Also, developers are related to the pull requests they contributed to. Therefore, we created rule R5 to associate developers with expertise.

Table 10 – SWRL rules.

SWRL Rules
R1: Keyword(t1) ∧ Topic(t2) ∧ *name*(t1, n1) ∧ *name*(t2, n2) ∧ *equal*(n1, n2) → hasTopic(t1, t2)
R2: Keyword(t1) ∧ SpecificTopic(st) ∧ *name*(t, n1) ∧ *name*(st, n2) ∧ *equal*(n1, n2) → hasSpecificTopic(t1, st)
R3: PullRequest(pr) ∧ Keyword(t1) ∧ hasKeyword(pr, t1) ∧ hasTopic(t1, t2) → hasTopic(pr, t2)
R4: PullRequest(pr) ∧ Keyword(t1) ∧ hasKeyword(pr, t1) ∧ hasSpecificTopic(t1, st) → hasSpecificTopic(pr, st)
R5: Developer(d) ∧ hasPullRequest(d, pr) ∧ hasExpertise(pr, ex) → hasExpertise(d, ex)

Thus, we can say that a given developer d can be connected to a set of distinct keywords $K_d = \{k_0, k_1, ..., k_n\}$ through contributions made in pull requests. Of these, each keyword $k \in K_d$ can be associated with a set of one or more pull requests $P_d = \{p_0, p_1, ..., p_y\}$, to which d has contributed. Furthermore, each keyword $k \in K_d$ is associated with one expertise $e \in E_d = \{e_0, e_1, ..., e_z\}$, being E_d the set of all expertise associated

to developer d. Therefore, the proposed ontology is also used to calculate the expertise weight, considering temporal information to compute its variation over time. For this purpose, we employ equation 6.1 and equation 6.2.

We used Equation 6.1 (125) to calculate W_k^d, the weight of the keyword k, for developer d considering the y Pull Requests for which d contributed. BY is the base year used in this work, 2021 (BY must be greater than the maximum value of CY, otherwise we would have 1/0 in the exponent of the equation), and CY is the pull request p creation year, which the keyword k is associated with. Thus, Equation 6.1 gives greater weight to the more recent contributions associated with the keyword.

$$W_k^d = \sum_{y \in P_d} exp^{1/(BY-CY)} \tag{6.1}$$

Consequently, WEX_e^d, the total weight of expertise e for developer d can be calculated using Equation 6.2. In Equation 6.2, we summarize the total expertise weight e associated with Pull Requests P_d.

$$WEX_e^d = \sum_{k \in K_d} W_k^d \tag{6.2}$$

6.1.4 Evaluation

The evaluation activity is carried out during all phases in the traditional *Methontology* Framework (126). In our work, we considered this activity as another phase in the proposed methodology, consisting of carrying out the following tasks: *verification* and *validation*.

The verification step consists of carrying out a technical judgment of the ontology, according to the ORSD, by verifying the correctness and validating the ontology. The correctness of the ontology is done through a verification process using the Pellet plugin reasoner on Protégé, a piece of software able to infer logical consequences from a set of asserted facts or axioms.

The validation is a step to ensure that the ontology fulfills its purpose. Suarez-Figueroa et al. (127) present guidelines based on using the Competency Questions (CQ) and the existing methodologies to build ontologies. Competency Questions specification is vital since it allows us to determine its scope and validate the ontology (128). Therefore, the verification step is done with the ontology responding correctly to the CQ (129). The CQs designed to aid in the validation activity are shown in Table 11 and are answered in the next subsection, where we present our results.

Table 11 – Main Competency Questions

Competency Questions (CQ)
CQ1: What expertise are present in the project?
CQ2: Who are the developers related to the expertise e?
CQ3: What is the expertise weight of a developer d?
CQ4: How are the developers ranked according to the expertise e?
CQ5: How has the expertise weight of a developer changed over time?
CQ6: Who is the most qualified developer to be recommended to assist with a task requiring expertise e?
In order to answer the CQ, queries were developed using python and Owlready2. The code can be found at[8,9]

6.2 Results

Initially, the database described in subsection **3.6.1** was loaded into the ontology and the SWRL rules were processed on the data to infer new relationships. According to the developed taxonomy, we have keywords associated with topics and specific topics. Thus, we use rules R1 and R2 to connect each tag to topics and specific topics (tag *test* connected to topic *Testing*, for example). Furthermore, we have PRs associated with keywords, and since keywords are associated with topics and specific topics we were able to infer new connections between PRs and topics (e.g. Testing, Security,..) and specific topics (e.g. New Features, C++,...) using rules R3 and R4. Finally, we have developers connected to PRs and associated with specific topics and topics by inference (rule R5). Therefore, we were able to find out implicit relationships between developers and expertise, defined by specific topics and topics.

In the analyses, the ontology was first used to identify all the expertise in the project Node.js - Competency Question *(CQ1)*. To do so, we look at developers associated with the project and their respective expertise associated with the collaboration. The identified expertise encompassed the following topics: *Memory*, *Performance*, *Security*, *CI*, *ProjectManagement*, *Testing*, *Updating*, *C++, Python*, *ES6+*, *NewFeature*, *CoreFeature*, *OperationalErrors*, and *ProgrammerErrors*.

Figure 26 shows the weight of each identified expertise, which corresponds to the sum of that expertise weight for all developers. According to these results, the most common expertise in the project is related to the topic *Updating*, followed by the specific topics *C++* and *Testing*. To answer Competency Question *(CQ2)*, we identified the developers having one or more of this expertise; there are 24 core and 1489 non-core individuals. Developers with expertise on *Updating* are more involved in releases (minor and major) in the project, working directly on feature development or bug fixes. On the

other hand, those with expertise on *Testing* are those individuals with unit test automation, debugging, and QA. Finally, C++ is a specific topic within programming languages and is one of the primary languages used in development projects. Our subsequent analyses focus on these three areas of expertise.

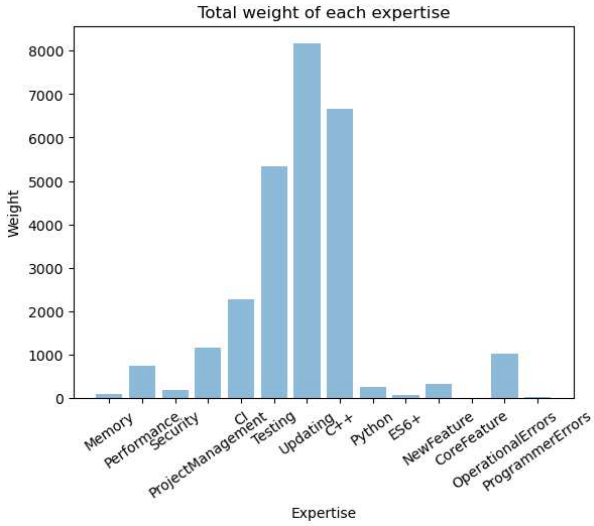

Figure 26: Total weight of each expertise in the network.

We investigated the weight difference of *Updating*, *C++*, and *Testing* expertise between core and non-core individuals. Regarding *(CQ3)*, Figure 27 shows box plots with the total weight percentage comparing the 24 core and 1489 non-core nodes related to each expertise. We can see that the individuals with the highest weight values are among the core nodes. Some are outliers as their expertise weight is significantly above the maximum value ($maximum = Q3 + 1.5 * IQR$). Again, these numbers reinforce the existence of workload in a small group of individuals. However, although most non-core individuals' weight is less than 0.2, a few non-core individuals have an expertise weight above the median of core individuals, indicating potential non-core individuals to be recommended.

For Competency Question *CQ4*, we ranked the developers considering their weight in the *Updating*, *C++*, and *Testing*. Table 12 shows the normalized score values of the top ten ranked developers. Table 12 is divided into two tables showing scores in two distinct periods. Table 12-A shows the scores considering the period from 2016-01-01 to 2019-12-31. On the other hand, Table 12-B considers the most recent period, from 2020-01-01 to 2020-07-01. We adopted this division as a strategy for evaluating the recommendations. We analyzed the developers considering their contributions between 2016 and 2019 and used their activity in 2020 to verify whether the individuals were still active in 2020 or not.

Looking specifically at recommendations for expertise in *Testing*, the three best-classified developers are Dev718899, Dev2512748, and Dev439929. When we check Table 12-B, we can see that Dev718899 and Dev439929 are suitable to be recommended because they are still maintaining a high performance (top-two developers) in 2020. On the other

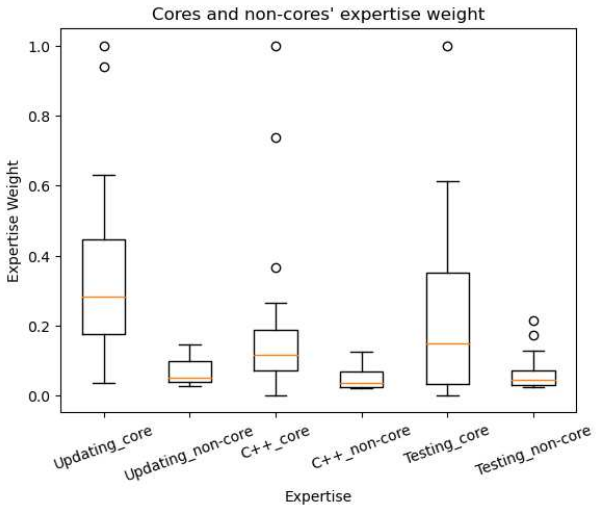

Figure 27: Box plots of Expertise Weight Values.

hand, Dev2512748 significantly fell in their rank, dropping from second to tenth. In this sense, we sought to explore the variation of developers' expertise weight over time to improve our recommendations by addressing the situation where individuals drop their scores significantly. We conducted a further analysis considering the five top-ranked developers in *Testing*.

Figure 28 shows (for *CQ5*) the expertise weight in each period (2016 to 2019), including the trend lines of each individual. The weight values are relative to the overall total weight of the expertise *Testing* each year, e.g., if only one individual has some expertise, their weight would be equal to 1. We can say that these values indicate the relevance of developers regarding the demand for tasks related to *Testing* in each period.

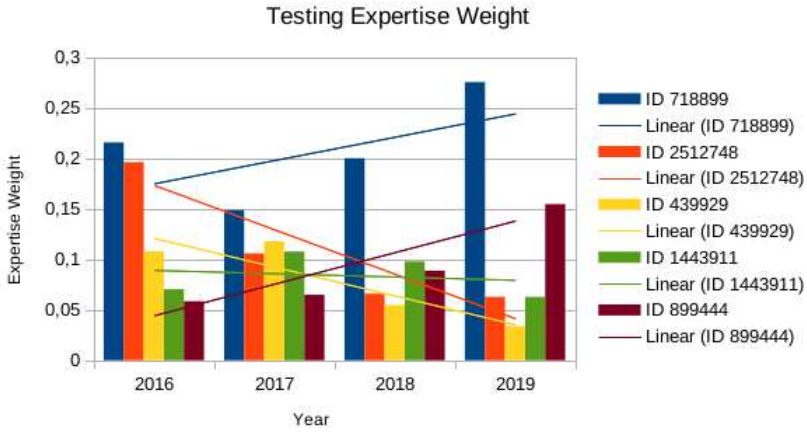

Figure 28: Trend lines of Expertise Weight Values.

Returning to the analysis of Table 12-A and -B, we can explain the rank drop of Dev2513748 by his reduction in interest in *Testing* questions over the years, demonstrated by his negative trend line. In contrast, the positive trend line supports the high rank of Dev718899 and the rise in the ranks of Dev899444. Therefore, for Competency Question

Table 12 – Recommendation Results

Developer	Updating	Developer	C++	Developer	Testing
Table II-a: Core Nodes Recommendations from 2016-01-01 to 2019-12-31					
3065230	1	899444	1	718899	1
439929	0.9376	275871	0.7358	2512748	0.6108
238531	0.6300	439929	0.3645	439929	0.4754
899444	0.5489	4299420	0.2647	1443911	0.4509
275871	0.4546	96947	0.1935	899444	0.3814
8822573	0.4449	2512748	0.1897	96947	0.3579
505333	0.4424	1538624	0.1852	8822573	0.3434
54666	0.4066	54666	0.1654	275871	0.3057
718899	0.3198	17607	0.1523	696611	0.2806
2512748	0.2929	10393198	0.1262	54666	0.1684
Table II-b: Core Nodes Recommendations from 2020-01-01 to 2020-07-01					
899444	1	899444	1	439929	1
439929	0.8889	439929	0.5158	718899	0.9827
3065230	0.5542	275871	0.4183	899444	0.8281
8822573	0.3786	5952481	0.1476	8822573	0.4524
52195	0.3762	4299420	0.1385	3065230	0.1780
5952481	0.3399	54666	0.1122	275871	0.1313
275871	0.3209	2512748	0.1118	1443911	0.1292
54666	0.2685	8822573	0.0600	17607	0.0848
718899	0.2502	17607	0.0598	54666	0.0417
2352663	0.1492	2352663	0.0426	2512748	0.0411

CQ6, we can say that Dev718899 and Dev899444 are more suitable to be recommended for tasks that require expertise in *Testing*. Furthermore, these analyses reinforce the importance of considering the individuals' change of interest in recommendation approaches. We can improve the results by prioritizing individuals with a positive contribution tendency rather than those with a drop in interest.

Therefore, we can answer **SRQ3** since we could use an ontology to assist in identifying developers with specific expertise. With the ontology, we inferred implicit relations between developers and expertise, in addition to giving weight to these relationships and ranking developers according to each expertise. Also, in order to improve the analytical process, we investigated the trend of developers' interest in expertise over the years, allowing us to identify developers with a high weight of expertise and a positive interest trend.

7 Historical Research 4: Network Partitioning

This chapter addresses the fourth challenge related to data volume in designing recommendation systems and analyzing social structures. We propose a balanced subtree-splitting strategy as a new parallel graph partitioning algorithm. Furthermore, the partitioning approach has a focus on density-based clustering algorithms for social network analysis. We can highlight two main goals of our proposal: (i) the proposed method preserves the partitions as connected components, so this is considered valuable for connection-dependent methods (130), such as density-based clustering approaches, to be subsequently processed over the partitions, and; (ii) the presence of overlapping nodes on different partitions is allowed, supporting new clustering approaches such as NetSCAN, which also allow the presence of overlapping nodes in different clusters (32).

7.1 Partitioning Algorithm

In this section, we describe the proposed algorithm for a balanced partitioning of a graph in parallel. First, it is important to note that we do not intend to propose a combinatorial optimization approach since we do not seek to find an optimal result or use an objective function to measure the quality of solutions. Instead, the program focuses on the time optimization of density-based clustering algorithms through the use of a partitioning approach.

The partitioning method was designed to split a graph into two partitions. If more than two partitions are required, the function is invoked recursively up to the desired number of partitions. Therefore, as our focus is on parallel partitioning, the whole process was designed to split the data set into 2^n partitions using multiple processors simultaneously. When a partitioning execution ends, it immediately calls the partitioning function for each new subset.

The partitioning strategy will be explained in the following subsections and can be defined by three main algorithms: (i) preprocessing, (ii) partitioning, and (iii) recursion. The first takes care of setting and initializing the parameters. The second defines the balanced subtree-splitting strategy. While the third is related to the parallel approach, which includes the partitioning function recursion. The line number in the algorithm will often be referred to in the text for a better explanation.

7.1.1 Pre-processing

Algorithm 1 shows the preprocessing operations necessary for structuring data and defining parameters. As expected input, we have: (i) *db*, a dataset that can be modeled as a graph $G = (V, E)$ with no edge directionality constraint required; (ii) *pt*, the number of desired partitions; and (iii) *per*, the percentage value for defining limits.

First, *db* is modeled as a network (**line 1**), and its minimum spanning tree (*mst*) is calculated (**line 2**). Working with a *mst* structure is a crucial step, as it allows the network to be split into two subnetworks by removing only a single edge. Next, in **line 3**, we search for the node with the highest degree (*hd*) in the *mst*, that is, the node (or one of the nodes) with the highest number of connected subtrees. This node will be the starting point in the partitioning process.

The value *per* is used to define limits (**line 4** and **line 5**), which will be used in the network partitioning process. The *upper limit* defines the desired maximum number of nodes that a partition can have, while the *lower limit* defines the minimum number of nodes. We use threshold values as they allow flexibility in the size difference between partitions, e.g., if *per* were equal to 0, the algorithm would always try to split the network into partitions of the same size strictly. If that were the case, the algorithm would not work when it was not possible to break a graph into connected components of the same size.

Finally, the *partitioning* function is called (**line 6**), passing the defined parameters as input.

Algorithm 1 Initialization

Input: database: *db*; partitions: *pt*; percentage: *per*
1: *g = graph creation (db)*
2: *mst = minimum spanning tree (g)*
3: *hd = highest degree node (mst)*
4: *upper limit = number of nodes (mst) * (0.5 + per/100)*
5: *lower limit = number of nodes (mst) * (0.5 - per/100)*
6: *partitioning (mst, hd, upper limit, lower limit)*

7.1.2 Graph Partitioning

Algorithm 2 defines the main loop of the graph partitioning algorithm. First, we initialize two vectors (*vector1, vector2*), where each partition's subtrees is stored, and two variables (*sum1, sum2*), responsible for tracking the number of nodes in each partition over the program iterations.

At each step of the proposed subtree splitting strategy, we can have one of four possible scenarios (A, B, C, D), as illustrated in Figure 30. Starting with the root node (*rt*), the algorithm will traverse all its subtrees (*st*) until it reaches one of the four scenarios. The scenarios are described below.

- **Scenario A**: This scenario occurs when a subtree st_i with size within the limits is found (**line 4**). Thus, the *cut* function is invoked to split the graph into two sets (**line 5**). One set with the subtree st_i, and another set with the node rt and all its subtrees but st_i.

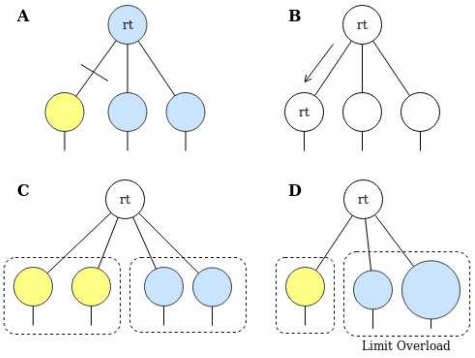

Figure 29: Subtree-Splitting Strategy Scenarios.

- **Scenario B**: This scenario occurs when a given *rt*'s subtree (st_i) has a number of nodes greater than the *upper_limit* (**line 6**). In this case, the root node (*rt*) goes down to the st_i's root node position (**line 8**), and the next loop starts with a new root node *rt*.

- **Scenario C**: This scenario can be divided into two parts (C1 and C2). In both cases, we know that all *rt*'s subtrees sizes are below the *lower_limit* because the condition of scenarios A and B were not met. In this way, the algorithm employs a sequential method to define a subtree combination for partitioning.

  - **C1**: First, it is checked if the size of the subtree st_i plus the size of one of the partitions reaches a value within the limits (**line 10 and 13**). If the check passes, the subtree st_i is added to the respective partition (**line 11 and 14**), node *rt* is added to both partitions as it is the node connecting all the structure, and function *cut* is called (**line 12 and 15**) as one of the partitions has reached a number of nodes within the defined limits.
  - **C2**: As long as C1 is not satisfied (**line 16 and 20**), the subtree st_i is placed in the smallest partition (**line 18** and **line 22**), the number of nodes in the partitions is refreshed, and a new comparison (C1) is performed on the next subtree st_j (**line 19** and **line 23**).

- **Scenario D**: This is considered an overload situation and can even be seen as the third part of case C. However, we define this as a new scenario to demonstrate the case when none of the above cases (C1 and C2) are satisfied (**line 24**). It occurs when the size of st_i plus the size of any of the partitions is greater than *upper_limit* (**line 25** and **line 28**). Thus, st_i is added to the smallest partition (**line 26** and **line 29**) and node *rt* is added in both partitions for the same reason explained in scenario C. Finally, the partitioning procedure is done (**line 27** and **line 30**).

Algorithm 2 Partitioning

Input: *m*, *hd*, *upper limit*, *low limit*
Output: *2 partitions*
Initialisation : vector1, vector2, sum1, sum2
1: *st = first subtree (hd)*
2: **while** not call cut function **do**
3: *cont = count number of nodes (st)*
4: **if** ($lowlimit \leq cont \leq upperlimit$) **then**
5: *cut (m, st, rt)*
6: **else if** ($upperlimit < cont$) **then**
7: *initialises (vector1, vector2, sum1, sum2)*
8: *rt = st*
9: *st = next subtree (rt)*
10: **else if** ($lowlimit \leq cont + sum1 \leq upperlimit$) **then**
11: *vector1 = append (st)*
12: *cut (m, st, vector1)*
13: **else if** ($lowlimit \leq cont + sum2 \leq upperlimit$) **then**
14: *vector2 = append (st)*
15: *cut (m, st, vector2)*
16: **else if** ($sum1 + cont \leq lowlimit$) **then**
17: *sum1 = sum1 + cont*
18: *vector1 = append (st)*
19: *st = next subtree (rt)*
20: **else if** ($sum2 + cont \leq lowlimit$) **then**
21: *sum2 = sum2 + cont*
22: *vector2 = append (st)*
23: *st = next subtree (rt)*
24: **else**
25: **if** $sum1 \geq sum2$ **then**
26: *vector2 = append (st)*
27: *cut (m, vector1, vector2)*
28: **else**
29: *vector1 = append (st)*
30: *cut (m, vector1, vector2)*
31: **end if**
32: **end if**
33: **end while**

7.1.3 Recursion

In Algorithm 3, we have defined the program recursion. It shows the *cut* function overview and the beginning and end of the parallel zone. Let graph g and two subgraphs sg_1 and sg_2 be the function's input; the partitioning function will be invoked continuously by tasks/threads until it reaches the stop condition (desired number of partitions). As long as the condition is not satisfied, the highest degree node (hd) is recalculated, and the *upper limit* and *lower limit* for each of the partitions sg_1 and sg_2. Finally, the partitioning function will be invoked for each partition by passing the necessary parameters.

At the end of all processes, the respective analysis procedures, such as a clustering procedure, can be performed on each obtained partition.

Algorithm 3 Cut

Input: graph: g, subgraphs: sg_1, sg_2
1: sg_1,sg_2 = *split (g)*
2: *free (g)*
parallelism zone: start
3: **if** (did not reach the desired number of partitions) **then**
4: *recalculate hd, upper limit, low limit for* sg_1 *and* sg_2
5: *partitioning (mst(*sg_1*), hd, upper limit, low limit)*
6: *partitioning (mst(*sg_2*), hd, upper limit, low limit)*
7: **end if**
parallelism zone: end
8: *analysis procedures (*sg_1*,* sg_2*, ...,* sg_n*)*

7.2 Results

This section presents partitioning studies, including performance results concerning the proposed algorithm. In order to evaluate our approach, we seek to investigate whether the partitioning method can split a graph-modeled database into n balanced partitions, keeping the two particulars defined in this chapter beginning. Furthermore, as the main goal is to allow the partitioning to be processed in a viable time, we also conducted a performance study comparing its sequential and parallel executions. The objective is to evaluate gains and losses in terms of memory and time consumption.

Below are described the specifications of the hardware and software used for tests:

- Hardware: 3.10 GHz CPU Intel I5 second generation with 16GB RAM.
- Operating System: 64-bit Debian GNU/Linux 10.
- Software: OpenMP API was used for the parallel implementation with the maximum number of threads available (*four*) in a shared memory multiprocessing programming in C/C++ language.

7.2.1 Datasets Description

The experiments are divided into two steps. First, we apply the partitioning method on smaller datasets and perform the analysis of the obtained partitions. We seek to verify the feasibility of finding balanced partitions, considering the adoption of limits for calculating the partitions. The small networks used to evaluate our approach are: (i) Zachary's Karate Club (131), (ii) Protein Network (132), (iii) artificial and (iv) 200data (32), as well as the GitHub network used in this work (see Section **3.6.1**).

However, the main goal is to investigate the feasibility of our application in social networks with large volumes of data. Therefore, we extend the analysis of the algorithm to a larger dataset in order to explore and detail the partitioning advantages. We also carry out performance analysis considering processing time and memory consumption. We

use data available from a well-known dataset repository [1]. The Digital Bibliography & Library Project (DBLP) database (122) is used and modeled as a network of scientific citations between researchers. For network modeling, an "isCitedBy" connection is created from researcher A to researcher B when B cites a publication by A. The network has a total of 1,100,505 nodes and 9,915,146 edges.

7.2.2 Partitioning Results

The limit percentage chosen for this experiment was about 5%. Therefore, the *low limit* is equal to 45%, and the *higher limit* is equal to 55%. This is not to say that only partitions within this limit will be found, but this was considered an adequate value to look for to obtain balanced partitions. Furthermore, a previous empirical experiment showed that in some cases, the partitions' size does not change much with higher limit values, and for lower values, it takes longer to find partitions within the limits.

Table 13 shows the partitioning result into the five smallest instances. In the table, we have each instance's name, size, and number of partitions (p). The number of partitions varies from 2 to 32 depending on the instance size. In order to analyze obtained partitions' size and their proximity to the defined limit, we looked at the maximum obtained cut percentage among all the divisions performed in each step, e.g, 12.5% at p=8 means that in the four partitionings performed (split each of 4 networks into 2 to obtain 8 subnetworks), the partitioning that resulted in the greatest difference in size between the subnetworks obtained was 37.5% of the nodes for one partition and 62.5% of nodes for another one.

These initial results show that while the limit values help to find balanced partitions, there are several cases where the cut percentage exceeds the thresholds due to the graph structure.

Table 13 – Maximum Cut Percentage (in %)

instance	**size**	**p=2**	**p=4**	**p=8**	**p=16**	**p=32**
artificial	16	12.5%	4.54%	-	-	-
karate	34	2.94%	2.63%	12.5%	-	-
protein	21	7.14%	0%	16.66%	-	-
200data	200	3.5%	4.62%	14.4%	7.89%	11.53%
github	5536	13.8%	3.11%	5.05%	6.44%	10.0%

Also, Table 14 details the results regarding the DBLP database. Since it is a large network, we conducted split operations in the instance into up to 16,384 partitions, in which some characteristics were observed: (i) the maximum cut percentage (**cutp**); (ii) the partition with the largest size (**max**); (iii) the partition with the smallest size (**min**); and, (iv) the total of overload cases in the partitions (scenario D).

[1] https://networkdata.ics.uci.edu

Table 14 – DBLP Partitioning Results

partitions	cutp (%)	max	min	overload
2	1	551,799	548,706	0
4	5	307,710	244,089	0
8	5	164,231	120,640	0
16	5	84,634	55,381	0
32	5	46,250	27,065	0
64	5	24,416	12,770	0
128	5	12,492	6,119	0
256	20	6,726	2,790	6
512	18	3,687	1,132	22
1024	16	2,015	521	42
2048	21	1,108	214	80
4096	21	596	102	169
8192	21	324	49	356
16384	22	179	24	742

When we look at the two partitions, we can see that the largest partition is approximately the same size (1.005 times larger) as the smallest partition. Analyzing the 16,384 partitions, we can see that the largest partition is about 7.45 larger than the smallest one. Although the *cutp* is within the range of 5% up to division by 128 partitions, it is possible to see an increasing contrast between the largest and smallest partition as the number of partitions increases. This is due to the *cutp* value that is recalculated based on the size of the partitions at each iteration. Also, after 128 partitions, we have the beginning of overload occurrences. The sum of all overhead cases is 1,417, corresponding to 8.64% of the 16,383 partitioning processes.

According to the result, it was possible to find partitions ensuring connectivity and allowing nodes to overlap. These are considered attractive initial results since finding optimal graph division while maintaining the network connectivity is difficult. Therefore, it is possible to say that the proposed algorithm could split a graph-modeled database into n balancing partitions, maintaining the two particulars proposed.

7.2.3 Performance Analysis

Since the proposed algorithm focuses on processing large data sets in a feasible time, we compare the sequential and parallel execution of the approach, considering the analysis of the algorithm's performance in terms of processing time and memory consumption.

We first highlight some implementation decisions to understand the results better.

- **Decision 1**: A previous analysis of the functions was carried out to understand which ones demand more processing time. We could see that the search functions

are the most expensive. Consequently, an adjacency list was adopted as a structure to represent the graph, and the search in the graph is done through direct access using a vector with the positions corresponding to the nodes' ids. This decision aims to improve execution time in exchange for increased memory consumption.

- **Decision 2**: As we have limited available resources/infrastructure to carry out the tests, we free up all memory spaces (used by the structures) as soon as they are not used to allow the program to run. The function *Cut* (Algorithm 3) shows the deallocation of graph g after being partitioned into g_1 and g_2. Following the same logic, g_1 and g_2 are also deallocated after each recursion step.

The graphs in Figure 30 illustrate the comparison of parallel and sequential approaches concerning execution time and memory consumption.

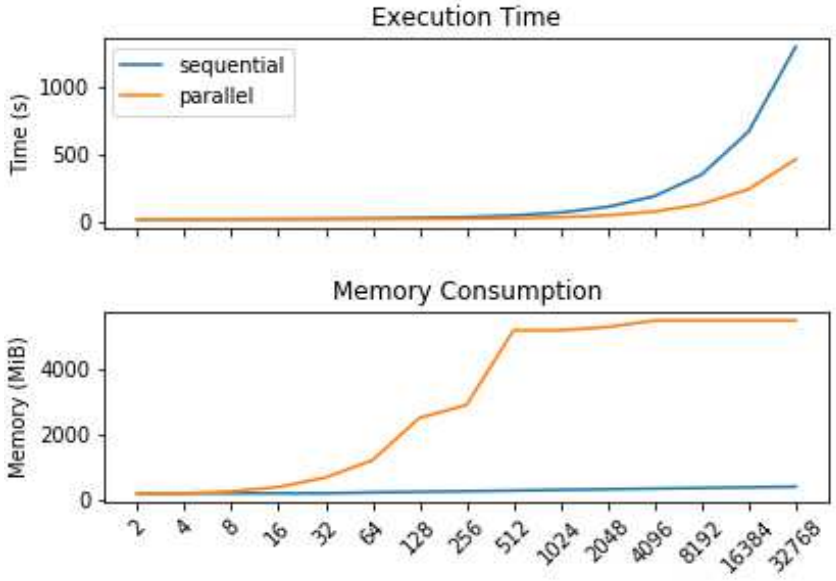

Figure 30: Graphics of Performance Results.

The top graph shows the relationship between the number of partitions (x-axis) and execution time (y-axis). According to the results, the execution time remains approximate for both approaches up to 512 partitions. After that, it is possible to notice an increasing gap between the execution time of the approaches as the number of partitions increases. With 32,768 partitions, we have a peak in the difference of values where the sequential approach takes 1,298s to run, almost three times longer than the parallel approach with 466s.

The bottom graphic shows the relation between the number of partitions (x-axis) and memory consumption (y-axis). The memory value at each point corresponds to the higher consumption peak during the program execution. By analyzing the graphic, we can see that the memory consumption does not change significantly with the increase in

the number of partitions in the sequential approach. However, we can see that the parallel approach presents an increase in memory consumption up to 512 partitions; this was expected since the program allocates access and storage structures while invoked recursively. Furthermore, we can see that memory consumption stabilizes after 512 partitions. This can be due to hardware limitations and implementation decisions regarding the number of threads and the structure allocation/deallocation flow that seems to restrict the total memory consumption at a maximum value.

Finally, it is possible to answer **SRQ4**. Focused on assisting in social network community detection, we developed a new parallel partitioning algorithm for graph structures. Our approach has a balanced subtree-splitting strategy, designed to support density-based clustering algorithms, considering node overlapping among groups and ensuring partition connectivity. We also adopted boundaries to make the partitioning process more flexible since the search for partitions of the same size is not always possible. We also investigated the algorithm behavior in a well-known large data set, aiming to understand the partitioning results and analyze the algorithm performance according to time and memory consumption. Initial findings point out that the algorithm can split a data set into n balanced partitions maintaining the particulars proposed and indicating relevance for further parallel applications.

8 Conclusion and Future Works

GSD contexts comprise social networks with connected developers worldwide. In these structures, the essential individuals have greater relevance than the average member. They consist of the most collaborative and participative members in the network, connecting distinct groups of developers. Also, they can generally be seen as experts: the most experienced developers who can help solve complex tasks to achieve project goals in specific situations. Experts are essential in GSD since they can bring skills and expertise to a project and are likely to collaborate with others to solve distinct tasks. Although identifying these individuals is valuable, recommending experts to help with project tasks is demanding, especially when considering research challenges faced when analyzing social structures in GSD.

We proposed a collaboration social network based on data related to popular projects on GitHub. An initial analysis showed that few developers are associated with many connections on the network. These are relevant members in the network, called core nodes. Identifying core nodes in all periods was possible, and they were considered essential for the network evolution due to their high contribution degree.

In order to address the five research challenges, we proposed an architectural framework for expert identification. The framework has seven main components for performing temporal, diversity-based, and semantic network analysis.

First, we investigated the evolution of communities in the network. Our method was to analyze overlapping structures in sequential time periods and, as a result, we were able to explore temporal changes by investigating syntactic and semantic network aspects. We could identify influential individuals with a high contribution degree in the network since they are responsible for contributing in periods of high demand and specific periods that require particular technologies. They are considered essential based on their role in meeting specific project demands. However, our results also showed that some of these individuals are highly requested, which can cause them to be overworked.

With the main goal of finding non-obvious experts for collaboration, we developed a diversity-based approach to identify non-obvious developers with similar skills to experts. We aimed at reducing the work overload over the same group of developers by finding non-obvious individuals to assist with project issues, increasing the diversity in GSD collaboration scenarios. Several classification models were explored and the adopted classification model, LDA, had the highest accuracy. We could also identify individuals that were classified by LDA as false positives (*fp*) indicating that they had skills similar to those of identified experts. We provided evidence that those that fell into the *fp* group could play important roles in the projects.

However, these analyses were insufficient to understand fully these individuals'

roles in the network. In this sense, semantic approaches may be considered in searching for individuals with specific knowledge. Thus, we proposed an ontology as a semantic approach to represent the domain knowledge and enrich the results with semantic information. In developing the ontology we looked to extract topics and specific topics from keywords in the projects. It allowed us to enrich the developer network with more semantic weight and brought knowledge about the projects' evolution. Therefore, we were able to identify and classify individuals with particular expertise considering the variation of their interests over the years.

Furthermore, some community detection methods, such as NetSCAN, face processing time infeasibility in analyzing large networks as the structural complexity of the networks increases. In this sense, graph partitioning methods can reduce the effort required by the network processing approaches, acting as a starting point for many parallel applications. Therefore, we proposed a new parallel partitioning algorithm for graph structures. Our approach has a balanced subtree-splitting strategy, designed to support density-based clustering algorithms, considering nodes between clusters, and ensuring partition connectivity. We investigated the algorithm behavior on a well-known large data set, aiming to understand the partitioning results and analyze the algorithm performance according to time and memory consumption. Initial findings point out that the algorithm is performative and can split a dataset into n balanced partitions, keeping the proposed particulars.

All these approaches were integrated into our final architectural framework that combines syntactic and semantic analysis techniques, including machine learning algorithms, complex network analysis methods, and ontology development, in addition to taking into consideration performance issues for high-volume networks. The framework allowed us to explore distinct aspects of the network in order to identify those with specific knowledge. Our results are promising for recommending experts as it was possible to identify central developers with a high weight of expertise and positive interest trends. Therefore, **RQ1** is finally answered since we could develop an expert recommendation approach integrating the proposed temporal, diversity-based, semantic, and partitioning strategies.

In future work, we intend to advance each research challenge addressed in this paper by enhancing our approaches for temporal, diversity, and semantic aspects in analyses. We also look to improve the ontology, with the aim of recommending specialists and teams of specialists with complementary skills. This work focused on analyzing a GSD context with various individuals, projects, and artifacts. In the future, we also plan to extend our taxonomy by exploring other software development projects on Github.

Moreover, we intend to investigate the next steps of the parallel process illustrated in Figure 2. In addition to the partitioning step, we will apply a density-based clustering algorithm to the obtained partitions and merge the partitions in order to analyze the

parallelism process as a whole.

REFERÊNCIAS

1 M. Janssen, H. van der Voort, and A. Wahyudi, "Factors influencing big data decision-making quality," *Journal of Business Research*, vol. 70, pp. 338–345, jan 2017. [Online]. Available: https://doi.org/10.1016%2Fj.jbusres.2016.08.007

2 A. Sapountzi and K. E. Psannis, "Social networking data analysis tools & challenges," *Future Generation Computer Systems*, vol. 86, pp. 893–913, 2018.

3 S. Haymond and S. R. Master, "How can we ensure reproducibility and clinical translation of machine learning applications in laboratory medicine?" pp. 392–395, 2022.

4 J. D. Herbsleb and D. Moitra, "Global software development," *IEEE software*, vol. 18, no. 2, pp. 16–20, 2001.

5 S. Jalali and C. Wohlin, "Agile practices in global software engineering-a systematic map," in *2010 5th IEEE International Conference on Global Software Engineering*. IEEE, 2010, pp. 45–54.

6 D. Schall, "Who to follow recommendation in large-scale online development communities," *Information and Software Technology*, vol. 56, no. 12, pp. 1543–1555, 2014.

7 J. Bryan, "Excuse me, do you have a moment to talk about version control?" *The American Statistician*, vol. 72, no. 1, pp. 20–27, 2018.

8 J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," *Journal of Systems and Software*, vol. 83, no. 1, pp. 67–76, 2010.

9 V. Horta, V. Ströele, V. Schettino, J. Oliveira, J. M. David, and M. A. P. Araújo, "Collaboration analysis in global software development," in *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2019, pp. 464–469.

10 T. Lopes, V. Ströele, R. Braga, and M. Bauer, "Unraveling the semantic evolution of core nodes in a global contribution network," in *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2020, pp. 594–601.

11 J. Rubin and M. Rinard, "The challenges of staying together while moving fast: An exploratory study," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 982–993.

12 S. Mahmood, S. Anwer, M. Niazi, M. Alshayeb, and I. Richardson, "Key factors that influence task allocation in global software development," *Information and Software Technology*, vol. 91, pp. 102–122, 2017.

13 H. Ko, S. Lee, Y. Park, and A. Choi, "A survey of recommendation systems: recommendation models, techniques, and application fields," *Electronics*, vol. 11, no. 1, p. 141, 2022.

14 T. Lopes, V. Ströele, R. Braga, J. M. N. David, and F. Campos, "A recommendation approach to diversify the collaboration scenario in global software development contexts," in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2021, pp. 1191–1196.

15 Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1. IEEE, 2014, pp. 335–342.

16 M. B. Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 530–543, jun 2016. [Online]. Available: https://doi.org/10.1109%2Ftse.2015.2500238

17 P. G. Campos, F. Díez, and I. Cantador, "Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols," *User Modeling and User-Adapted Interaction*, vol. 24, no. 1, pp. 67–119, 2014.

18 P. K. Singh, M. Sinha, S. Das, and P. Choudhury, "Enhancing recommendation accuracy of item-based collaborative filtering using bhattacharyya coefficient and most similar item," *Applied Intelligence*, vol. 50, no. 12, pp. 4708–4731, 2020.

19 X. Xie, X. Yang, B. Wang, and Q. He, "Devrec: Multi-relationship embedded software developer recommendation," *IEEE Transactions on Software Engineering*, 2021.

20 K. Das and R. N. Behera, "A survey on machine learning: concept, algorithms and applications," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no. 2, pp. 1301–1309, 2017.

21 E. Mirsaeedi and P. C. Rigby, "Mitigating turnover with code review recommendation: balancing expertise, workload, and knowledge distribution," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1183–1195.

22 S. Asthana, R. Kumar, R. Bhagwan, C. Bird, C. Bansal, C. Maddila, S. Mehta, and B. Ashok, "Whodo: Automating reviewer suggestions at scale," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 937–945.

23 M. Ali, N. Yousuf, M. Rahman, J. Chaki, N. Dey, K. Santosh *et al.*, "Machine translation using deep learning for universal networking language based on their structure," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 8, pp. 2365–2376, 2021.

24 S. Scheider, F. O. Ostermann, and B. Adams, "Why good data analysts need to be critical synthesists. determining the role of semantics in data analysis," *Future generation computer systems*, vol. 72, pp. 11–22, 2017.

25 M. Zanin, D. Papo, P. A. Sousa, E. Menasalvas, A. Nicchi, E. Kubik, and S. Boccaletti, "Combining complex networks and data mining: why and how," *Physics Reports*, vol. 635, pp. 1–44, 2016.

26 D. Dou, H. Wang, and H. Liu, "Semantic data mining: A survey of ontology-based approaches," in *Proceedings of the 2015 IEEE 9th international conference on semantic computing (IEEE ICSC 2015)*. IEEE, 2015, pp. 244–251.

27 L. aDepartament de Llenguatges i Sistemes Informàtics, "Taking advantage of semantics in recommendation systems," in *Artificial Intelligence Research and Development: Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence*, vol. 220. IOS Press, 2010, p. 163.

28 E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 92–101.

29 R. Diaz-Bone, K. Horvath, and V. Cappel, "Social research in times of big data. the challenges of new data worlds and the need for a sociology of social research," *Historical Social Research/Historische Sozialforschung*, vol. 45, no. 3, pp. 314–341, 2020.

30 M. Eirinaki, J. Gao, I. Varlamis, and K. Tserpes, "Recommender systems for large-scale social networks: A review of challenges and solutions," pp. 413–418, 2018.

31 A. P. B. Veyseh, T. N. Nguyen, and T. H. Nguyen, "Graph transformer networks with syntactic and semantic structures for event argument extraction," *arXiv preprint arXiv:2010.13391*, 2020.

32 V. Horta, V. Ströele, R. Braga, J. M. N. David, and F. Campos, "Analyzing scientific context of researchers and communities by using complex network and semantic technologies," *Future Generation Computer Systems*, vol. 89, pp. 584–605, 2018.

33 T. Lopes, V. Ströele, R. Braga, J. M. N. David, and M. Bauer, "A broad approach to expert detection using syntactic and semantic social networks analysis in the context of global software development," *Journal of Computational Science*, vol. 66, p. 101928, 2023.

34 ——, "Identifying and recommending experts using a syntactic-semantic analysis approach," in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2021, pp. 739–744.

35 T. Lopes, V. Ströele, M. Dantas, R. Braga, and J.-F. Meháut, "A parallel graph partitioning approach to enhance community detection in social networks," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–6.

36 L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, Jan. 1978. [Online]. Available: https://doi.org/10.1016/0378-8733(78)90021-7

37 D. Camacho, Á. Panizo-LLedot, G. Bello-Orgaz, A. Gonzalez-Pardo, and E. Cambria, "The four dimensions of social network analysis: An overview of research methods, applications, and software tools," *Information Fusion*, vol. 63, pp. 88–120, Nov. 2020. [Online]. Available: https://doi.org/10.1016/j.inffus.2020.05.009

38 J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection," in *Proceedings of the 19th international conference on World wide web - WWW '10*. ACM Press, 2010. [Online]. Available: https://doi.org/10.1145/1772690.1772755

39 W. Yuan, D. Guan, Y.-K. Lee, S. Lee, and S. J. Hur, "Improved trust-aware recommender system using small-worldness of trust networks," *Knowledge-Based Systems*, vol. 23, no. 3, pp. 232–238, Apr. 2010. [Online]. Available: https://doi.org/10.1016/j.knosys.2009.12.004

40 F. Wu, B. A. Huberman, L. A. Adamic, and J. R. Tyler, "Information flow in social groups," *Physica A: Statistical Mechanics and its Applications*, vol. 337, no. 1-2, pp. 327–335, Jun. 2004. [Online]. Available: https://doi.org/10.1016/j.physa.2004.01.030

41 P. Liu, B. Raahemi, and M. Benyoucef, "Knowledge sharing in dynamic virtual enterprises: A socio-technological perspective," *Knowledge-Based Systems*, vol. 24, no. 3, pp. 427–443, Apr. 2011. [Online]. Available: https://doi.org/10.1016/j.knosys.2010.12.004

42 A. Lieberman, "The hidden power of social networks: Understanding how work really gets done in organizations," *Teachers College Record*, vol. 107, no. 11, pp. 2507–2510, Nov. 2005. [Online]. Available: https://doi.org/10.1111/j.1467-9620.2005.00619.x

43 M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, Feb. 2004. [Online]. Available: https://doi.org/10.1103/physreve.69.026113

44 S. Y. Bhat and M. Abulaish, "A density-based approach for mining overlapping communities from social network interactions," in *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics - WIMS '12*. ACM Press, 2012. [Online]. Available: https://doi.org/10.1145/2254129.2254143

45 S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, Feb. 2010. [Online]. Available: https://doi.org/10.1016/j.physrep.2009.11.002

46 C. Wang, W. Tang, B. Sun, J. Fang, and Y. Wang, "Review on community detection algorithms in social networks," in *2015 IEEE International Conference on Progress in Informatics and Computing (PIC)*. IEEE, Dec. 2015. [Online]. Available: https://doi.org/10.1109/pic.2015.7489908

47 S. Kianian, M. R. Khayyambashi, and N. Movahhedinia, "FuSeO: Fuzzy semantic overlapping community detection," *Journal of Intelligent &amp Fuzzy Systems*, vol. 32, no. 6, pp. 3987–3998, May 2017. [Online]. Available: https://doi.org/10.3233/jifs-151276

48 M. Cordeiro, R. P. Sarmento, and J. Gama, "Dynamic community detection in evolving networks using locality modularity optimization," *Social Network Analysis and Mining*, vol. 6, no. 1, pp. 1–20, 2016.

49 A. Reihanian, M.-R. Feizi-Derakhshi, and H. S. Aghdasi, "Overlapping community detection in rating-based social networks through analyzing topics, ratings and links," *Pattern Recognition*, vol. 81, pp. 370–387, Sep. 2018. [Online]. Available: https://doi.org/10.1016/j.patcog.2018.04.013

50 Z. Zhao, S. Feng, Q. Wang, J. Z. Huang, G. J. Williams, and J. Fan, "Topic oriented community detection through social objects and link analysis in social networks," *Knowledge-Based Systems*, vol. 26, pp. 164–173, Feb. 2012. [Online]. Available: https://doi.org/10.1016/j.knosys.2011.07.017

51 S. Akter, "Recommending expert developers using usage and implementation expertise," Ph.D. dissertation, University of Lethbridge (Canada), 2021.

52 S. Lefterova, "Expert finding system: Profile components and design," Ph.D. dissertation, Saxion, 2019.

53 C. Moreira and A. Wichert, "Finding academic experts on a multisensor approach using shannon's entropy," *Expert Systems with Applications*, vol. 40, no. 14, pp. 5740–5754, 2013.

54 G. Robinson, "Capturing a moving target: interviewing fintech experts via linkedin," *Area*, vol. 53, no. 4, pp. 671–678, 2021.

55 T. V. Rampisela, D. Elisabeth, and D. I. Sensuse, "Characteristics of expertise locator system in academia: A systematic literature review," in *2020 International Seminar on Intelligent Technology and Its Applications (ISITIA)*. IEEE, 2020, pp. 186–193.

56 X. Wang, C. Huang, L. Yao, B. Benatallah, and M. Dong, "A survey on expert recommendation in community question answering," *Journal of Computer Science and Technology*, vol. 33, no. 4, pp. 625–653, 2018.

57 Y. Zou, L. Zhao, Y. Kang, J. Lin, M. Peng, Z. Jiang, C. Sun, Q. Zhang, X. Huang, and X. Liu, "Topic-oriented spoken dialogue summarization for customer service with saliency-aware topic modeling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 16, 2021, pp. 14 665–14 673.

58 N. Nikzad-Khasmakhi, M. Balafar, and M. R. Feizi-Derakhshi, "The state-of-the-art in expert recommendation systems," *Engineering Applications of Artificial Intelligence*, vol. 82, pp. 126–147, 2019.

59 E. Rich, "User modeling via stereotypes," *Cognitive science*, vol. 3, no. 4, pp. 329–354, 1979.

60 M. Sanderson and W. B. Croft, "The history of information retrieval research," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1444–1451, 2012.

61 G. Shani and A. Gunawardana, "Evaluating recommendation systems," in *Recommender systems handbook*. Springer, 2011, pp. 257–297.

62 G. Suresh, A. S. Kumar, S. Lekashri, and R. Manikandan, "Efficient crop yield recommendation system using machine learning for digital farming," *International Journal of Modern Agriculture*, vol. 10, no. 1, pp. 906–914, 2021.

63 M. Kherad and A. J. Bidgoly, "Recommendation system using a deep learning and graph analysis approach," *Computational Intelligence*, vol. 38, no. 5, pp. 1859–1883, 2022.

64 S. Sharma, V. Rana, and V. Kumar, "Deep learning based semantic personalized recommendation system," *International Journal of Information Management Data Insights*, vol. 1, no. 2, p. 100028, 2021.

65 J. Pérez-Marcos, L. Martín-Gómez, D. M. Jiménez-Bravo, V. F. López, and M. N. Moreno-García, "Hybrid system for video game recommendation based on implicit ratings and social networks," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 11, pp. 4525–4535, 2020.

66 G. T. Reddy, M. P. K. Reddy, K. Lakshmanna, R. Kaluri, D. S. Rajput, G. Srivastava, and T. Baker, "Analysis of dimensionality reduction techniques on big data," *IEEE Access*, vol. 8, pp. 54 776–54 788, 2020.

67 D. Camacho, Á. Panizo-LLedot, G. Bello-Orgaz, A. Gonzalez-Pardo, and E. Cambria, "The four dimensions of social network analysis: An overview of research methods, applications, and software tools," *Information Fusion*, vol. 63, pp. 88–120, 2020.

68 M. Ianni, E. Masciari, and G. Sperlí, "A survey of big data dimensions vs social networks analysis," *Journal of Intelligent Information Systems*, vol. 57, no. 1, pp. 73–100, 2021.

69 R. Kitchin and G. McArdle, "What makes big data, big data? exploring the ontological characteristics of 26 datasets," *Big Data & Society*, vol. 3, no. 1, p. 2053951716631130, 2016.

70 X. Ye and X. Wei, "A multi-dimensional analysis of el niño on twitter: Spatial, social, temporal, and semantic perspectives," *ISPRS International Journal of Geo-Information*, vol. 8, no. 10, p. 436, 2019.

71 H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

72 S. Cavallari, E. Cambria, H. Cai, K. C.-C. Chang, and V. W. Zheng, "Embedding both finite and infinite communities on graphs [application notes]," *IEEE computational intelligence magazine*, vol. 14, no. 3, pp. 39–50, 2019.

73 M. Dehghan, M. Biabani, and A. A. Abin, "Temporal expert profiling: With an application to t-shaped expert finding," *Information Processing & Management*, vol. 56, no. 3, pp. 1067–1079, 2019.

74 Z. Meng, F. Gandon, and C. F. Zucker, "Joint model of topics, expertises, activities and trends for question answering web applications," in *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 2016, pp. 296–303.

75 N. Dakiche, F. B.-S. Tayeb, Y. Slimani, and K. Benatchba, "Tracking community evolution in social networks: A survey," *Information Processing & Management*, vol. 56, no. 3, pp. 1084–1102, 2019.

76 G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: a survey," *ACM computing surveys (CSUR)*, vol. 51, no. 2, pp. 1–37, 2018.

77 J. Sun, W. Xu, J. Ma, and J. Sun, "Leverage raf to find domain experts on research social network services: A big data analytics methodology with mapreduce framework," *International Journal of Production Economics*, vol. 165, pp. 185–193, 2015.

78 J. Yang, S. Peng, L. Wang, and B. Wu, "Finding experts in community question answering based on topic-sensitive link analysis," in *2016 IEEE first international conference on data science in cyberspace (DSC)*. IEEE, 2016, pp. 54–60.

79 R. Medeiros and O. Díaz, "Assisting mentors in selecting newcomers' next task in software product lines: A recommender system approach," in *International Conference on Advanced Information Systems Engineering*. Springer, 2022, pp. 460–476.

80 L. Guo, E. Tan, S. Chen, X. Zhang, and Y. Zhao, "Analyzing patterns of user content generation in online social networks," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 369–378.

81 D. Hong and V. Y. Shen, "Online user activities discovery based on time dependent data," in *2009 International Conference on Computational Science and Engineering*, vol. 4. IEEE, 2009, pp. 106–113.

82 N. H. Bidoki, M. Schiappa, G. Sukthankar, and I. Garibay, "Predicting social network evolution from community data partitions," in *2019 International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation*, 2019.

83 A. Daud, J. Li, L. Zhou, and F. Muhammad, "Temporal expert finding through generalized time topic modeling," *Knowledge-Based Systems*, vol. 23, no. 6, pp. 615–625, 2010.

84 A. Pal, S. Chang, and J. Konstan, "Evolution of experts in question answering communities," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 6, no. 1, 2012, pp. 274–281.

85 R. Yeniterzi and J. Callan, "Moving from static to dynamic modeling of expertise for question routing in cqa sites," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 9, no. 1, 2015, pp. 702–705.

86 R. Brochier, A. Guille, J. Velcin, B. Rothan, and D. Cioccio, "Peerus review: a tool for scientific experts finding," *arXiv preprint arXiv:1807.03719*, 2018.

87 B. W. Bader, R. A. Harshman, and T. G. Kolda, "Temporal analysis of semantic graphs using asalsan," in *Seventh IEEE international conference on data mining (ICDM 2007)*. IEEE, 2007, pp. 33–42.

88 P.-I. Lee and P.-R. Hsueh, "Emerging threats from zoonotic coronaviruses-from sars and mers to 2019-ncov," *Journal of microbiology, immunology, and infection*, vol. 53, no. 3, p. 365, 2020.

89 Y. Liu, W. Tang, Z. Liu, L. Ding, and A. Tang, "High-quality domain expert finding method in cqa based on multi-granularity semantic analysis and interest drift," *Information Sciences*, vol. 596, pp. 395–413, 2022.

90 M. Dehghan, H. A. Rahmani, A. A. Abin, and V.-V. Vu, "Mining shape of expertise: A novel approach based on convolutional neural network," *Information Processing & Management*, vol. 57, no. 4, p. 102239, 2020.

91 K. Bradley and B. Smyth, "Improving recommendation diversity," in *Proceedings of the twelfth Irish conference on artificial intelligence and cognitive science, Maynooth, Ireland*, vol. 85. Citeseer, 2001, pp. 141–152.

92 L. McGinty and B. Smyth, "On the role of diversity in conversational recommender systems," in *International Conference on Case-Based Reasoning*. Springer, 2003, pp. 276–290.

93 D. T. Hoang, N. T. Nguyen, and D. Hwang, "Recommendation of expert group to question and answer sites based on user behaviors and diversity," *Journal of Intelligent & Fuzzy Systems*, vol. 37, no. 6, pp. 7117–7129, 2019.

94 H. Yin, B. Cui, and Y. Huang, "Finding a wise group of experts in social networks," in *International Conference on Advanced Data Mining and Applications*. Springer, 2011, pp. 381–394.

95 D. Easley and J. Kleinberg, *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge university press, 2010.

96 D. R. Karger and C. Stein, "A new approach to the minimum cut problem," *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 601–640, 1996.

97 K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, 2004, pp. 120–124.

98 T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser, "Graph bisection algorithms with good average case behavior," *Combinatorica*, vol. 7, no. 2, pp. 171–191, 1987.

99 C. G. Fernandes, T. J. Schmidt, and A. Taraz, "On minimum bisection and related cut problems in trees and tree-like graphs," *Journal of Graph Theory*, vol. 89, no. 2, pp. 214–245, 2018.

100 M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *Proceedings of the sixth annual ACM symposium on Theory of computing*, 1974, pp. 47–63.

101 H. Xu, D. Luo, and L. Carin, "Scalable gromov-wasserstein learning for graph partitioning and matching," *Advances in neural information processing systems*, vol. 32, 2019.

102 M. Vojnovic, C. E. Tsourakakis, C. Gkantsidis, and B. Radunovic, "Graph partitioning for massive scale graphs," Jun. 14 2022, uS Patent 11,361,483.

103 L. Dhulipala, G. E. Blelloch, and J. Shun, "Theoretically efficient parallel graph algorithms can be fast and scalable," *ACM Transactions on Parallel Computing (TOPC)*, vol. 8, no. 1, pp. 1–70, 2021.

104 J. Zeng and H. Yu, "A study of graph partitioning schemes for parallel graph community detection," *Parallel Computing*, vol. 58, pp. 131–139, 2016.

105 C. Y. Cheong, H. P. Huynh, D. Lo, and R. S. M. Goh, "Hierarchical parallel algorithm for modularity-based community detection using gpus," in *European conference on parallel processing*. Springer, 2013, pp. 775–787.

106 H. Sun, W. Jie, J. Loo, L. Wang, S. Ma, G. Han, Z. Wang, and W. Xing, "A parallel self-organizing overlapping community detection algorithm based on swarm intelligence for large scale complex networks," *Future Generation Computer Systems*, vol. 89, pp. 265–285, 2018.

107 Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, "Streaming graph partitioning: an experimental study," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1590–1603, 2018.

108 A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," *Algorithm engineering*, pp. 117–158, 2016.

109 P. Vassiliadis, "A survey of extract–transform–load technology," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 5, no. 3, pp. 1–27, 2009.

110 X. Zou, Y. Hu, Z. Tian, and K. Shen, "Logistic regression model optimization and case analysis," in *2019 IEEE 7th international conference on computer science and network technology (ICCSNT)*. IEEE, 2019, pp. 135–139.

111 A. Tharwat, T. Gaber, A. Ibrahim, and A. E. Hassanien, "Linear discriminant analysis: A detailed tutorial," *AI communications*, vol. 30, no. 2, pp. 169–190, 2017.

112 I. Wickramasinghe and H. Kalutarage, "Naive bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation," *Soft Computing*, vol. 25, no. 3, pp. 2277–2293, 2021.

113 P. Cunningham and S. J. Delany, "K-nearest neighbour classifiers-a tutorial," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.

114 M. Tanveer, T. Rajani, R. Rastogi, Y.-H. Shao, and M. Ganaie, "Comprehensive review on twin support vector machines," *Annals of Operations Research*, pp. 1–46, 2022.

115 K. Das, S. Samanta, and M. Pal, "Study on centrality measures in social networks: a survey," *Social network analysis and mining*, vol. 8, no. 1, pp. 1–11, 2018.

116 M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "Methontology: from ontological art towards ontological engineering," 1997.

117 N. Zöller, J. H. Morgan, and T. Schröder, "A topology of groups: What github can tell us about online collaboration," *Technological Forecasting and Social Change*, vol. 161, p. 120291, 2020.

118 V. K. Eluri, T. A. Mazzuchi, and S. Sarkani, "Predicting long-time contributors for github projects using machine learning," *Information and Software Technology*, vol. 138, p. 106616, 2021.

119 D. Banks, C. Leonard, S. Narayan, N. Thompson, B. Kramer, and G. Korkmaz, "Measuring the impact of open source software innovation using network analysis on github hosted python packages," in *2022 Systems and Information Engineering Design Symposium (SIEDS)*. IEEE, 2022, pp. 110–115.

120 G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 345–355.

121 M. Tanaka, K. Taura, T. Hanawa, and K. Torisawa, "Automatic graph partitioning for very large-scale deep learning," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 1004–1013.

122 M. Ley, "Dblp: some lessons learned," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1493–1500, 2009.

123 X. Cai, J. Zhu, B. Shen, and Y. Chen, "Greta: Graph-based tag assignment for github repositories," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 63–72.

124 P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation." *Encyclopedia of database systems*, vol. 5, pp. 532–538, 2009.

125 V. Ströele, G. Zimbrão, and J. M. Souza, "Group and link analysis of multi-relational scientific social networks," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1819–1830, 2013.

126 M. F. López, A. Gómez-Pérez, J. P. Sierra, and A. P. Sierra, "Building a chemical ontology using methontology and the ontology design environment," *IEEE Intelligent Systems and their applications*, vol. 14, no. 1, pp. 37–46, 1999.

127 M. C. Suárez-Figueroa, A. Gómez-Pérez, and B. Villazón-Terrazas, "How to write and use the ontology requirements specification document," in *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*. Springer, 2009, pp. 966–982.

128 A. Khatoon, Y. Hafeez, S. Asghar, and T. Ali, "An ontological framework for requirement change management in distributed environment," *The Nucleus*, vol. 51, no. 2, pp. 291–301, 2014.

129 C. Bezerra, F. Freitas, and F. Santana, "Evaluating ontologies with competency questions," in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, vol. 3. IEEE, 2013, pp. 284–285.

130 Y. Lv, T. Ma, M. Tang, J. Cao, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "An efficient and scalable density-based clustering algorithm for datasets with complex structures," *Neurocomputing*, vol. 171, pp. 9–22, 2016.

131 W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

132 J. Meena and V. S. Devi, "Overlapping community detection in social network using disjoint community detection," in *2015 IEEE Symposium Series on Computational Intelligence*. IEEE, 2015, pp. 764–771.