From Ontology to Knowledge Graph: An AI-Powered Approach for Extracting Software Context from System Documentation

Athul Raj Nambiar
athul.raj.nambiar@student.uva.nl

May 24, 2023, 77 pages

Academic supervisor:	Dr. Ana Maria Oprescu, a.m.oprescu@uva.nl
Host supervisor:	Xander Schrijen MSc., x.schrijen@sig.eu
Daily supervisor:	Damian Frölich MSc., d.frolich@uva.nl
Host organisation:	Software Improvement Group, https://www.softwareimprovementgroup.com/

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Comprehending the context of a software system is an integral part of its maintenance, evaluation, and evolution. Hence, businesses and organizations are witnessing a growing demand for the efficient extraction of contextual information of software systems from their documentation. However, there is currently limited research being done in this subdomain of software engineering. Our study aims to bridge this research gap and serve as a foundation for future research.

Knowledge graphs (KGs) have the capacity to establish context for software systems, much like their role in representing real-world contextual information. Consequently, our research is dedicated to presenting software context from documentation through KGs. This research proposes a formal definition of the concept "software context" using our Software Context Ontology (SCO). Our definition draws upon scientific literature and insights obtained from consultation with experts who engage with software systems daily. To explore the efficacy of extracting software context from documentation through Natural Language Processing (NLP), we introduce a novel approach for few-shot Software Context Named Entity Recognition (SCNER). Given the absence of suitable datasets for training, validating and evaluating NER models in the domain of software context, we present our own compact Software Context Dataset (SCD). Finally, leveraging the SCO and our trained SCNER model, we present our exploration of a Software Context Knowledge Graph (SCKG) development strategy with the aim of offering a comprehensive overview of software context within system documentation.

Thorough assessments, conducted using the HermiT reasoner, the OOPS! scanner, and the FOCA methodology, validate the consistency and syntactic accuracy of the SCO while also evaluating its overall quality. Our findings contribute a formalized and versatile definition of software context through the SCO, opening up possibilities for its application across the software engineering domain. Nonetheless, it's crucial to emphasize the importance of further validation through expert consultation. Our specialized NER dataset (SCD), encompassing 8 software context entities, served as the foundation for fine-tuning the SCNER model. Remarkably, this model achieved an impressive average F1 score of 0.78, demonstrating its efficacy in software context entity extraction. While these results emphasize the model's success, future endeavours will involve broader and more diverse datasets to mitigate potential biases and improve generalization capabilities. Lastly, our exploration of the SCKG was to offer a holistic view of software context within system documentation. Although the initial approach faced validity challenges due to rule-based relationship matching, insights gained from expert evaluations highlighted its potential. Future work will pivot towards refining the relation extraction process to ensure the validity and high quality of the SCKG.

Contents

Chapter 1

Introduction

In the rapidly evolving landscape of software engineering, it becomes ever more important to consider the context of a software system, because system context describes the interaction of a system with its environment [1]. Understanding how a system interacts with its environment holds substantial importance in the field of software evaluation, due to its ability to provide a holistic perspective on how a software system operates within its surroundings. Moreover, comprehending software context also allows developers to define a system's boundaries, relevant factors, and irrelevant elements. This comprehension not only shapes the initial development process but also offers valuable insights to software engineers for the ongoing modification, maintenance, and evolution of the system. Therefore, researching how to obtain and comprehend the context of a software system is valuable, especially to businesses and organizations that run and maintain large software systems with many employees.

Knowledge graphs (KG) have been used to put information into context by linking it to other bits of information. In a KG, entities in the real world and/or a business domain (e.g., people, places, or events) are represented as nodes, which are connected by edges representing the relations between those entities [2]. The KGs provide structural context to these real-world/domain-specific scenarios. Many companies such as Google, Microsoft, and Facebook have their own, non-public KGs, there is also a larger body of publicly available KGs, such as DBpedia or Wikidata [2].

As of now, there is limited research on retrieving context from software systems. KGs which establish context for real-world contextual information by linking it to other related data, hold the potential to extend this concept to software systems. For software engineers, the documentation of software systems plays a pivotal role as it documents the contextual information of said systems. Software documentation refers to reports, white papers, documentation, README files, and more that provide valuable information regarding a software system's development, users, components, technologies, and other aspects. To retrieve context for software, we would study the available documentation for software systems and appropriately construct KGs for the systems. This would require the extraction of entities and relations for the systems presented in the documentation. While the field of natural language processing (NLP) techniques, specifically Named Entity Recognition (NER) and relation extraction (RE), is not extensively explored in this domain, there exist valuable studies that can serve as a foundation for further research. Once this information is extracted, we can integrate it into a simplified visual representation in the form of a KG.

1.1 Problem statement

Providing system context in a concise format in a digestible format such as a KG can greatly improve the time and resources required for software engineers to understand, modify and evaluate a system. Furthermore, this streamlined KG representation of system context can also facilitate non-technical experts' understanding of the software system. However, to our current knowledge, there is limited to no research conducted on both the extraction of software context from system documents and the representation of system context as a KG. Additionally, there is no clear and universally accepted definition of software system context. We reference that one of the few online available general definitions for the context of software systems by an organization in the software development industry [1], but there is no scientific definition to describe this term. Therefore, constructing a valid and well-argued definition is important for our research.

Ontologies are a means of formally conceptualizing and defining what we wish to represent in a specific domain [3]. However, as of current research, there has been no software context-related ontology developed. On the other hand, there are well-documented ontologies within the software engineering domain such as SEON [4], SWEBOK [5], and a few others. They provide a good foundation for building new ontologies in the software domain.

KGs are a graph-structured data model or topology that integrate data from a knowledge base [6]. They typically store knowledge based on interlinked descriptions of entities (objects, events, situations, or abstract concepts; related elements to so-called real-world objects and actions) while also encoding the semantics underlying the used terminology [6, 7]. KGs hold the capability to depict a software system's context in a manner that can be highly beneficial for software evaluators, regardless of their technical background. The entity-relationship model has the capability of visually representing the relationships within a software system. The current scope of KG research is heavily influenced by the use of NLP techniques [8, 9]. These techniques are able to extract knowledge from text with great accuracy, and therefore increasingly being researched.

Creating a KG based on the context of a software system requires investigating techniques to extract entity-based information from the system's documentation. NER is a field of research that uses NLP to create entities based on input text. After looking into NER for software systems specifically, we found that there is a lack of fundamental NLP techniques for identifying software-related named entities that appear within a natural language sentence [10]. To date, we found that the little entity extraction research done in the software domain generally does not relate to documentation, but instead, discussions about software and code on online forums or social content [11–13]. Other works investigate NER for software bug-related entities [14, 15]. These papers use deep learning models based off Bidirectional Long Short-Term Memory—Conditional Random Fields (BiLSTM-CRF) [13, 14, 16], in older research CRF based models [12, 15], or in recent years the well researched Bidirectional Encoder Representations from Transformers (BERT) model [12]. These models are well-researched for other NLP tasks and therefore can be adapted to other domain-specific NLP tasks, provided there is sufficient data to train and/or fine-tune the models.

1.1.1 Research questions

As there is a lack of research within the software domain that investigates software context, our investigation analyses the current level of related fields and adapts them to our specific interests. To tackle these issues, we investigate these research questions:

RQ1 How can the concept of "software context" be defined and represented through the creation of a formal ontology?

With this research question, we want to establish a formal definition of the concept "software context" to provide clarity to the term. Our aim is to formalize this concept using an ontology to create a precise and structured knowledge representation that can be used for future research in the domain.

RQ2 How effectively can relevant context be extracted from software system documentation, using AI techniques, particularly Named Entity Recognition (NER)?

The aim of this research question is to investigate the degree or extent to which relevant context can be successfully and accurately extracted from software system documentation using AI techniques. How we measure this "extent" (meaning the effectiveness) is by evaluating the performance of the technique we use.

RQ3 To what extent can a Knowledge Graph development methodology be employed to provide a holistic representation of software context from documentation?

Knowledge Graphs, like organized maps of information, can be used to understand and represent different parts of software context in system documents. We want to know if this method can effectively, and if so to what extent, capture and show all the important details about how the software works in documents. For a holistic representation, we will consider the ability of our approach to: cover all relevant software context information in a document (comprehensive coverage), identify and present deeper contextual understanding through the relationships (contextual understanding), and be an overall good method to represent software context (methodological soundness).

1.1.2 Research method

Our research is an exploratory study of an NLP-based approach to extracting software context information to construct a KG. To formulate our approach, we investigate the relevant literature and use our personal experiences/knowledge of ML. In the process, we develop a formal definition of software context in the form of an ontology, and a tool that extracts software context from documentation input into the system and constructs a KG. Parts of the tool (particularly the NLP models) are evaluated using quantitative experiments. This research is performed in collaboration with the Software Improvement Group (SIG). The evaluation of the entire approach is completed using qualitative experiments with the assistance of 5 industry experts from SIG (technical consultants who are software evaluation experts).

1.2 Contributions

In this section, we highlight the contributions and novel insights produced by this thesis, shedding light on the advancements made in the field of software engineering:

1. **Formal Definition of Software Context:**
   In this work, we first contribute a new ontology that is a formal definition of software context for the evaluation of software systems. As part of the contribution, we detail our full ontology development methodology. These contributions collectively address **RQ1**.
2. **Custom NER dataset in the domain of Software Context:**
   Our second contribution is the construction of a new fully labelled software context NER dataset, built using documentation from publicly available sources; it contains 8 software context entities.
3. **NER approach to extract Software Context information from text:**
   Our third contribution is a new, well-performing fine-tuned NER model that is capable of predicting software context entities within software documentation. The performance evaluation of this model significantly contributes to addressing **RQ2**.
4. **Investigation on constructing Software Context related Knowledge Graphs:**
   Our fourth contribution is the investigation of our approach to constructing software context KGs. This investigation plays a crucial role in addressing **RQ3**. Consequently, from our investigation, we contribute a tool that takes software documentation as its input and constructs a KG. This KG provides an overview of the software context knowledge presented in the original system documentation.

1.3 Outline

In Chapter 2 we describe the background of all the foundational knowledge and research relevant to our exploration. Chapter 3 describes the full ontology construction methodology, including the steps taken to evaluate our ontology. The construction of our labelled software context NER dataset is outlined in Chapter 4. Chapter 5 explains the full experimental setup to fine-tune our NER model, including the steps required to validate and evaluate the model. In Chapter 6 we connect all components of our research, illustrating their relevance in the KG development process. These steps also explain what our tool does to construct a KG. Furthermore, within this chapter, we explain how we will validate and evaluate our proposed approach. Our results are shown in Chapter 7 and discussed in Chapter 8. Chapter 9, contains the work related to this thesis. Finally, we present our concluding remarks in Chapter 10 together with future work.

Chapter 2

Background

This Chapter will present the necessary background information for this thesis. We provide academic definitions of Knowledge Graphs (KGs) and ontologies. We then outline a methodology for constructing a KG. Finally, we introduce the concept of few-shot Named Entity Recognition (NER) and state-of-the-art technology used in this field.

2.1 Knowledge Graph Definition

Most attempts to define KGs have focused on describing a generic semantic representation or key features [17]. However, there is not a formal definition that is broadly acknowledged. Four criteria were established for KGs by Paulheim [18]. Ehrlinger and Woß [19] examined multiple definitions and proposed a definition which emphasizes the KGs' reasoning engines. They state that in order to generate new knowledge, a KG gathers and incorporates information into an ontology. Wang et al. [20] state that a KG is a multi-relational graph composed of entities and relations. These entities and relations are regarded as nodes and different types of edges, respectively. For our research, we follow the definition by Wang et al. [20]. Based on Wang et al.'s [20] definition, Ehrlinger and Woß [19] provide the mathematical notation for a KG as $G = \{E, R, F\}$, E, R and F are sets of entities, relations and facts, respectively. A fact is denoted as a triple $(h, r, t) \in F$, where h is the starting node (head), r is the relationship and h is the ending node (tail).

2.2 Ontology Definition

An ontology is defined as a formal and explicit specification of a shared conceptualization [21]. Conceptualization refers to identifying the concepts of some sort of abstract model based on some phenomenon in the world. In an ontology, the types of concepts and the constraints on their use are explicitly defined. The purpose of creating a formal specification is to allow an ontology to be machine-readable. Last but not least, the term shared describes the idea that an ontology captures information that is accepted by a group and is not the property of any one individual. An ontology is a valuable research tool, as it can provide a formal specification of a representational language for a common domain of discourse [22].

There are numerous knowledge representation formalisms to represent ontologies [3, 21]. Each of them provides different components that can be used for their specific tasks. However, there is a set of fundamental components seen in all ontologies [21]:

1. **Classes:** represent concepts, and are the main formalized elements of the domain. These classes are typically organized into taxonomies, through which inheritance mechanisms are applied. An example of a class would be `Locations` (with subclasses such as `Cities`, `Villages`, etc.).
2. **Relations/ Relationships:** are links that represent an association between the concepts in the domain. They are formally defined as a subset of a product of n sets, where: $R \subset C1 \times C2 \times \ldots \times Cn$. A relation has two parameters, the domain indicating the initial concept and a range which is the second concept the relation links to. An example of a relation is `arrivalPlace`, which has a domain concept `Travel` and a range concept `Location`. In an ontology, relations are also used to indicate the taxonomic hierarchies between concepts, typically defined using a `isA` or inversely a `isPartOf` relation.

3. **Instances:** are used to represent individuals or the main objects within the domain in an ontology. For example, considering the class `Cities`, `Amsterdam` would be an instance of the class.
4. **Axioms:** are used to model sentences that are always true [22]. They are typically used to represent knowledge that can not be defined by the other components. Axioms consist of restrictions, rules, and logic correspondence definitions [23] that need to be met by the relations between the ontology components. Hence, they are used to verify the consistency of the knowledge and ontology itself.

2.3 Knowledge Graph Construction

Although many research papers provide the essential steps to construct a KG, these approaches differ from paper to paper and the field lacks a more generalized and global view [24]. Knowledge graph development is typically classified into two primary approaches: top-down and bottom-up [24]. In the top-down approach, an ontology or data schema is initially defined, and knowledge is subsequently extracted based on this established structure [25]. While the bottom-up approach involves extracting knowledge from data, and the ontology of the knowledge graph is defined based on the inherent characteristics of the data itself [25]. In our exploration, we follow the top-down approach. Tamašauskaitė and Groth [24] propose a methodology for KG development that is formulated using a systematic review of existing research. They describe the methodology as "A synthesis of common steps in KG development described in the academic literature". The steps of the methodology are as follows [24]:

1. **Identify data:** Choosing the domain of interest, a data source, and a data collection method are the goals of this step. Following the selection of the domain, it is critical to pinpoint the data sources because this will affect how the KG is developed overall and the methods used to extract information. The final step is to select the data acquisition techniques based on the kind and source of the data. For example, web crawlers can be used to find resources on the web.
2. **Construct the KG ontology:** Building the KG ontology, which gives the KG its top-level structure, is the goal of this step. When using the top-down strategy, this step is necessary. When building a KG ontology, the top-down technique is often employed when there is either (i) an existing, well-defined domain ontology that may serve as a foundation or (ii) structured data that acts as a framework to construct the ontology. By building the ontology for the KG, the entities, and the relationships between them, will already be defined.
3. **Extract knowledge:** Once the data is obtained, the next step is to extract knowledge from it. This is done by extracting entities, the relations between them and attributes.
   (a) *Extract entities:* The goal of entity extraction is to find and identify entities in a variety of data. The goals of this step are to identify entities for a given entity type as well as to identify more informative types for a certain entity. NER is one of the techniques most commonly used and is discussed in further detail in Section 2.4.
   (b) *Extract relations:* Once the entities have been extracted, they need to be linked. Therefore, the next step is to identify the relationship between them. This step depends on the type of data. For structured data, relations are explicit and can easily be identified. For semi-structured data, pattern-based and rule-based approaches are used; machine learning-based approaches can also be implemented [26]. Additionally, relation extraction from unstructured text data requires understanding semantic data, for which natural language processing (NLP) techniques are commonly used.
   (c) *Extract attributes:* Attribute extraction describes the gathering and aggregation of data about a certain item. It is sometimes viewed as the identification of unique kinds of relationships between things. However, the primary goal of this step is to provide a more detailed description of the entity.
4. **Process knowledge:** Knowledge processing is the next step in the methodology. This step's goal is to make sure the knowledge that has been retrieved is of high quality. The retrieved entities, relations, and characteristics that have not been processed may be unclear, redundant, or lacking. Furthermore, it is necessary to coordinate knowledge from many sources.
5. **Construct the KG:** Ensuring the KG is usable and accessible is the goal of this step. This involves permitting its usage, visualizing the KG for exploration, and storing it in an appropriate database.
6. **Maintain the KG:** KGs are never complete, since knowledge is continually developing. As a result, it's important to continuously monitor the KG, how it's used, the data sources that are

relevant to the domain, and to update the KGs appropriately.

(a) *Evaluate the KG:* Part of this maintaining is to evaluate the quality of the KG. Outside evaluating the completeness and quality of the KG, the KG can also be tested by users in order to obtain feedback. Feedback analysis can indicate missing information and suggestions for improvement.

(b) *Updating the KG:* Generally speaking, it may be necessary to update the KG when (i) a new data source that is relevant to the knowledge domain becomes available or (ii) an existing data source has new data.

2.4 Few-Shot Named Entity Recognition

NER is a task in the field of information extraction. It entails identifying and classifying a small subset of information elements known as Named Entities (NEs) into pre-defined categories [27]. As a result, it forms the foundation for a number of other important aspects of information management, including semantic annotation, question-answering, ontology population, and opinion mining. A NE has been defined in a variety of ways by experts in NER. Based on the study performed by Marrero et al. [27] they group these definitions according to the following four criteria: grammatical category, rigorous designation, distinctive identification, and domain of application. Mohit et al. [28] provide a more general definition where they state NEs are words or phrases which are named or categorized in a certain topic. These NEs contain key information in sentences and are subjects of interest for many language processing systems. A simple example of NER would be to recognise the entities PER and ORG, which are entities that show a person and organization respectively, from the sentence "Dr. Smith is informed by the Federal Bureau of Investigation that they must be placed into witness protection". A NER model would take this text input and label "Dr. Smith" as PER and "Federal Bureau of Investigation" as ORG.

Many language processing models demand a substantial amount of data for effective performance. Training these models from scratch requires exceptionally large datasets to yield satisfactory results. For instance, consider BERT (Bidirectional Encoder Representations from Transformers) [29], one of the most renowned and high-performing transformers in NLP. (For a comprehensive explanation of transformers, refer to Section 2.4.1). BERT is trained on a staggering 3.3 billion words in total, including 2.5 billion from Wikipedia and 0.8 billion from BooksCorpus. The hunger for data and the significant computational resources required are common challenges in many deep learning methods, stemming from their data-intensive nature. Furthermore, data scarcity is often encountered due to the complexity of the issue/problem, privacy issues, as well as the costs associated with data preparation. Gathering, preprocessing, and labelling data can be labour-intensive tasks, making large training datasets a scarce resource for many researchers. To address the challenge of limited training data, there exists a research field in machine learning (ML) known as "few-shot learning". Here, researchers focus on discovering underlying patterns in data using only a small sample of training data [30]. Few-shot learning seems like a low-cost approach that might significantly cut the turnaround time of developing ML applications. Sections 2.4.4 and 2.4.5 discuss two few shot learning techniques used in combination with NER training.

Another way to overcome the hurdle created by a lack of training data is a practice in ML known as "transfer learning". Transfer learning and few-shot learning are related but two distinct concepts in ML. Transfer learning (or inductive transfer) is the idea of improving the performance on a current task by applying information or a concept learnt on a prior task [31]. One of the techniques commonly used in transfer learning is "fine-tuning". Fine-tuning is the process of taking a pre-trained model and further training it on a new dataset specific to the target task. The idea is that during training, the weights within a pre-trained model are updated based on the new data, while still retaining a lot of the knowledge gained from the original model. In the context of creating NLP models for various tasks, fine-tuning pre-trained language models has become a standard practice [32]. This is because pre-trained language models can capture extensive semantic and syntactic information in text.

2.4.1 Transformers

In 2017, Vaswani et al. [33] introduced the transformer — revolutionizing the state of natural language processing (NLP) for the foreseeable future. Transformers rely solely on attention mechanisms, eliminating the need for recurrence and convolutions. Figure 2.1 outlines the architecture of the transformer. The following Subsections will provide an overview of the architecture components.

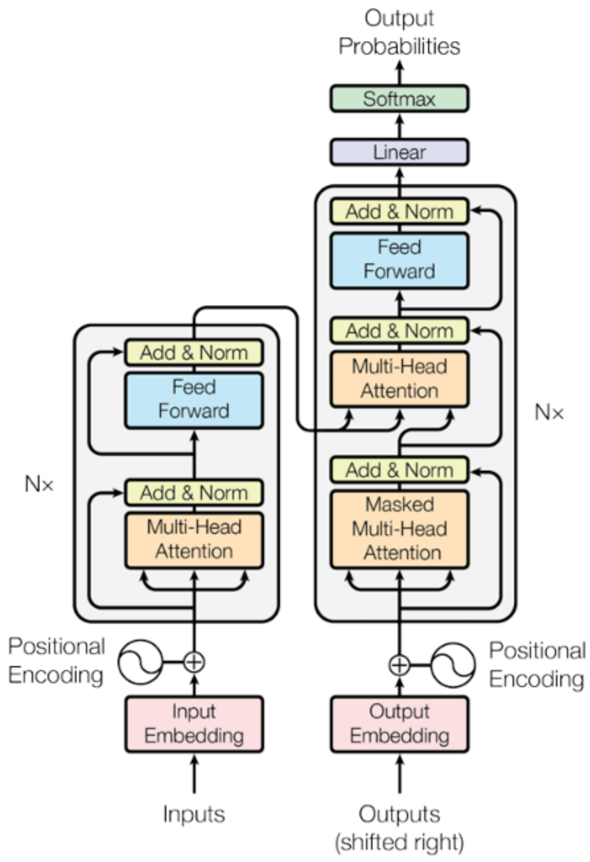

Figure 2.1: The transformer model architecture [33].

Encoder and Decoder Stacks

The encoder is composed of a stack of $N = 6$ identical layers [33]. There are two sublayers in each layer, as seen on the left portion of Figure 2.1. The first is a multi-head self-attention mechanism, while the second is a fully connected feed-forward network. Around each of the two sub-layers, they use a residual connection, followed by layer normalization. In other words, each sub-layer's output is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function that the sub-layer itself implements. All model sub-layers as well as the embedding layers generate outputs with dimension $d_{model} = 512$ to allow these residual connections.

Additionally, the decoder is made up of a stack of $N = 6$ identical layers [33]. The decoder uses the same stack as the encoder with an additional third sub-layer, which performs multi-head attention over the encoder stack's output. This can be seen on the right side of Figure 2.1. Residual connections are used around each of the sub-layers, much like the encoder, and then layer normalization. To stop positions from paying attention to succeeding positions, the self-attention sub-layer in the decoder stack is also changed. This masking guarantees that the predictions for location i, together with the fact that the output embeddings are offset by one position, can only depend on the known outputs at places less than i.

2.4.2 Embeddings and Softmax

The transformer uses learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}, similar to that of other sequence transduction models [33]. It uses learned linear transformations and the softmax function to convert the decoder output to predicted next-token probabilities.

The weight matrix between the two embedding layers and the pre-softmax linear transformation is the same. In the embedding layers, the weights are multiplied by $\sqrt{d_{model}}$.

Positional Encoding

The model contains neither recurrences nor convolutions. Because of this, in order for the model to make use of the order of the sequence, some information about the relative or absolute position of the token in the sequence must be input [33]. The positional encodings are added to the input embeddings for this purpose. The positional encodings have the same dimension d_{model} as the embeddings to calculate a sum. This model uses sine and cosines functions of different frequencies for the positional encoding:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.1}$$

pos is the position and i is the dimension. Each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. Using 10000 to calculate the wavelengths is a design choice that allows the positional encoding to cover a wide range of positions. The authors selected this function with the hypothesis that it would facilitate the Transformer's ability to learn relative positional attention effortlessly. This is because, for a constant offset k, PE_{pos+k} can be expressed as a linear function of PE_{pos}.

Attention Layers

An attention function maps a query and a set of key-value pairs to an output [33]. The query, keys, values, and output are all vectors. A weight is assigned to each value, which is calculated using a compatibility function of the query and the corresponding key. The output is calculated as a weighted sum of the values.

Figure 2.2 displays the architecture of "Scaled Dot-Product Attention". The input contains values of dimension d_v, keys of dimension d_k and the queries. The formula for the computation is as follows [33]:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.2}$$

QK^T is the dot product of a matrix of queries with a matrix of all the keys, softmax() is the softmax function, and V is the values' matrix.

Multi-head attention enables the model to jointly attend to data from several representations located in various positions. The layer in Figure 2.3 linearly projects the queries, keys, and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. This is done instead of performing a single attention function with d_{model}-dimensional keys, values, and queries because a single performance proved less beneficial. On each one of the projected versions of queries, keys, and values, the attention function is performed in parallel, yielding d_v-dimensional output values. These output values are then concatenated and once again project as displayed in Figure 2.3. This process can be expressed in this formula [33]:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(head_1, ..., head_h)W^O \\ \text{where } head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.3}$$

Where the projections are parameter matrices $W_i^Q \varepsilon\ \mathbb{R}^{d_{model}}$, $W_i^K \varepsilon\ \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \varepsilon\ \mathbb{R}^{d_{model} \times d_v}$. This architecture uses $h = 8$ parallel attention layers, or heads.

Feed-Forward Networks

As seen in Figure 2.1, both the encoder and decoder contain a fully connected feed-forward network. The layer consists of two linear transformations with a ReLU activation in between and can be summarised using this expression [33]:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.4}$$

Across the different positions, the linear transformations are the same, and the parameters used from layer to layer are different. Another way to describe this is having two convolutions with kernel size 1. The input and output dimensionality is $d_{model} = 512$, and the inner-layer dimensionality is $d_{ff} = 2048$.

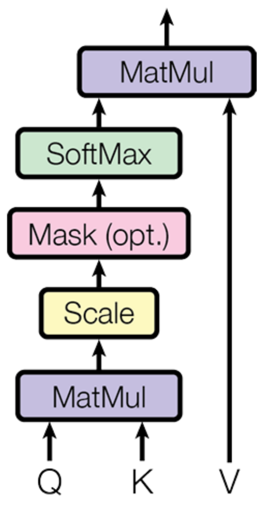

Figure 2.2: Scaled Dot-Product Attention [33].

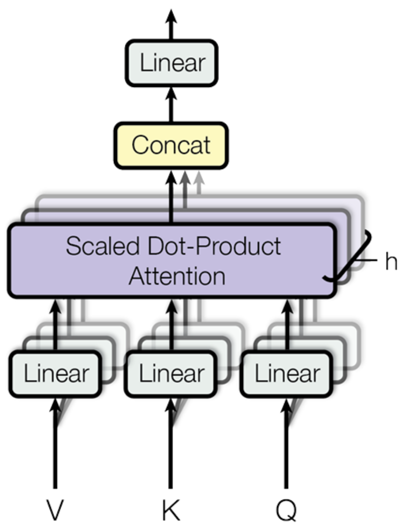

Figure 2.3: Multi-Head Attention consists of several attention layers running in parallel [33].

2.4.3 RoBERTa

In 2018, Devlin et al. [29] introduced a new language representation model called BERT. BERT is designed to pre-train deep bidirectional representation from unlabelled text, which is done by jointly conditioning both the left and right context in all layers. Consequently, a pre-trained BERT model can easily be fine-tuned with an additional output layer, which creates state-of-the-art models for a wide range of tasks, including a downstream task like NER. Furthermore, at the time of its release, BERT produced new state-of-art results for eleven NLP tasks. This is because BERT is able to capture the context and meaning of words within a sentence very well.

The model architecture of BERT is a multi-layer bidirectional Transformer encoder [29]. The architecture of the transformer is based on the architecture of the original transformer that is described in Section 2.4.1. The implementation of the transformer encoder in BERT is almost identical to this version. However, it is important to note that BERT is trained differently and used for different tasks to that of the transformer discussed in Section 2.4.1.

In 2019, Liu et al. [34] took the research introduced with BERT and worked to further improve the field of language model pretraining. They introduced their replication study of BERT, which carefully measured the impact of many key hyperparameters and training data size [34]. In their paper, they explain how BERT is significantly under-trained and as a result, propose an improved guideline for training BERT models, called RoBERTa (A Robustly Optimized BERT Pretraining Approach). The proposed training approach obtains state-of-the-art results that outperform BERT on a wide range of NLP benchmarks (i.e. GLUE, RACE, and SQuAD). It is important to note that their modifications do not include any architectural changes to BERT. All modifications relate to the training process [34]:

1. Training the model for much longer, with more data all in bigger batches.
2. The removal of the next sentence prediction objective used within the original BERT.
3. Training the model on longer sequences.
4. Making dynamic changes to the masking pattern applied to the training data.
5. Training on a large dataset (CC-NEWS) to better control the set size effects.

2.4.4 Linear Classifier Fine-tuning

Linear classifiers are a type of machine learning model that is used in classification tasks. Huang et al. [35] propose fine-tuning a linear classifier as part of their few-shot NER pipeline to obtain the best results with limited data. They use RoBERTa as the transformer-based backbone network to extract the contextualized representation of each token $z = f\theta_0(x)$. At the bottom of the backbone, the authors propose adding a linear classifier. Figure 2.4 displays the NER system where the string inputs are the output by the linear classifier with a prediction of each token to one of the entities. Since softmax is used, the output will be presented as a probability that the input predicted entity.

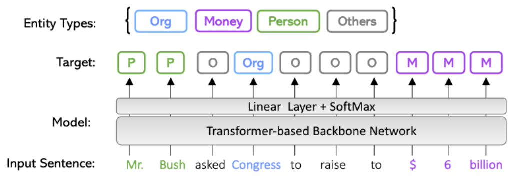

Figure 2.4: NER with a linear classifier [35].

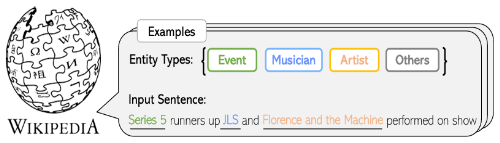
WIKIPEDIA
Examples
Entity Types:
Event
Musician
Artist
Others
Input Sentence:
Series 5 runners up JLS and Florence and the Machine performed on show

Figure 2.5: Noisy supervised pretraining [35].

The linear classifier is a linear layer with the parameter $\theta_1 = \{W, b\}$ followed by a softmax layer. In mathematical terms, the projection of the contextualized representation z into the label space is represented as $f_\theta(z) = \text{Softmax}(Wz + b)$. Therefore, the full output described in Figure 2.4 is output using the composition function: $y = f\theta_1 \circ f\theta_0(x)$, with the trainable parameters $\theta = \{\theta_0, \theta_1\}$ The model is optimized by minimizing cross-entropy loss:

$$\mathcal{L}(x, y) = \sum_{(\mathbf{X},\mathbf{Y}) \in \mathcal{D}^{\mathrm{L}}} \sum_{i=1}^{T} \mathrm{KL}\left(y_i \| q\left(y_i \mid x_i\right)\right) \tag{2.5}$$

where the KL divergence between the two distributions is $KL(p||q) = \mathbb{E}_p \log(p/q)$, and the prediction probability vector for each token is:

$$q(y \mid x) = \text{Softmax}\left(W \cdot f_{\theta_0}(x) + b\right) \tag{2.6}$$

2.4.5 Noisy Supervised Pretraining

The numerous deep-learning-based methods that obtain state-of-the-art results of NLP tasks owe their success to the availability of large data sources with reliable labels [36]. However, quality annotations at scale are expensive. To combat this, learning with noisy labels is an approach that suggests acquiring cheap annotation at scale, even if it means tolerating some mislabelled data.

Huang et al. [35] propose noisy supervised pretraining (NSP) in combination with a linear classifier, described in Section 2.4.4, and obtain much higher results for few-shot NER tasks. They apply NSP on the RoBERTa model with the large-scale noisy web data WiNER [37]. Over 50 million phrases and 113 entity kinds are included in the 6.8GB WiNER dataset. It is constructed by automatically annotating the 2013 English Wikipedia dump to Freebase using queries of the anchored string and the co-reference mentions in each wiki page. Their automatic annotation technique is very scalable and reasonably priced, however, it introduces inherent noise. For instance, a random subset of 1000 mentions is manually assessed and the accuracy of automatic annotations is around 77%, due to the inaccuracy of finding co-references. WiNER covers a wide variety of entity types, many of which are related but distinct from entity types in the downstream datasets. In Figure 2.5, the entities `Artist` and `Musician` are more fine-grained compared to, for example, `Person`, which is a common entity type in most other common NER datasets [35]. The advantage of NSP here is that it learns representations to distinguish each entity from other entities and non-entities. Additionally, in a few-shot environment, NSP helps prevent over-fitting because the models have knowledge previously learned for extracting entities from various contexts during training [35].

Chapter 3

Ontology Construction Methodology

In this chapter, we will discuss the methodology we use to construct our ontology. Thereafter, we introduce our evaluation steps to describe how we evaluate the quality of our constructed ontology.

3.1 Methodology Overview

In the last two decades, there has been extensive scholarly research being conducted on domain ontologies. For creating their own stand-alone ontologies, numerous research groups have proposed several methodologies, however, since ontology construction relates to specific fields/domains, there is no set standard [38]. Currently, the main ontology construction methods include Dev. 101 method, TOVE method, Methontology method, skeleton method, KACTUS method, SENSUS method, and IDEF5 method [39]. With these methods, each provides a general procedure with step-by-step operational guidelines to construct a domain-specific ontology. According to Sun et al. [40] a method for creating an ontology should include the following fundamental steps: first, gathering knowledge; second, abstracting and refining knowledge; third, expressing it in a way that is machine-readable and a computer can understand; and finally, assessing the ontology's quality and maintaining updates.

Sun et al. [40] constructed an ontology in the domain of software testing (EPOST) using a comprehensive construction method that follows the structure of relatively complete and mature methods: Dev 101. method, Methontology, and IDEF5 method. By merging T. R. Gruber's design ontology requirements with the software development life cycle standard IEEE 1074-2006, Yun et al. provide a knowledge engineering method for developing domain ontologies [38]. Both techniques have very similar approaches, whereas the knowledge engineering approach has some additional steps that are not explicitly outlined in the construction of EPOST. For our ontology construction process, we adapt a method that uses the EPOST construction process as a foundation. Then we slightly refactor the order of the steps to better validate our ontology and the identified concepts. Finally, we introduce additional steps from the knowledge engineering approach that are not explicitly included within the EPOST process.

Figure 3.1 outlines the six steps of our ontology construction process: ontology requirement analysis, core concept establishment, concept classification level establishment, ontology implementation, reusable ontology investigation, and ontology evaluation and evolution. In our adapted version of the ontology development process, we move the "reusable ontology investigation" from step 2 in the EPOST process to step 5. In the EPOST process, the authors use requirement analysis to determine the scope of the domain and then look for existing researched ontologies that can be reused as a foundation for their EPOST ontology. This foundation provides a list of concepts and other ontology components and avoids researchers investigating a domain and recreating already existing ontologies. However, this method is best suited for the construction of ontologies that can generalize a domain for all contexts. With our research, our domain of interest is SCESS, particularly in the context of the industry. Additionally, our project is in collaboration with industry experts that have both a distinctly practical and theoretical understanding of the domain in this context compared to software engineering researchers with a more general theoretical understanding. In our research, it can introduce errors and bias if we are to use reusable ontologies as the foundation. This is because including concepts and ontology components of similar ontologies that are not created for the context of our formal definition can potentially lead to the inclusion of inaccurate, irrelevant and/or biased information. Furthermore, after investigating ontology

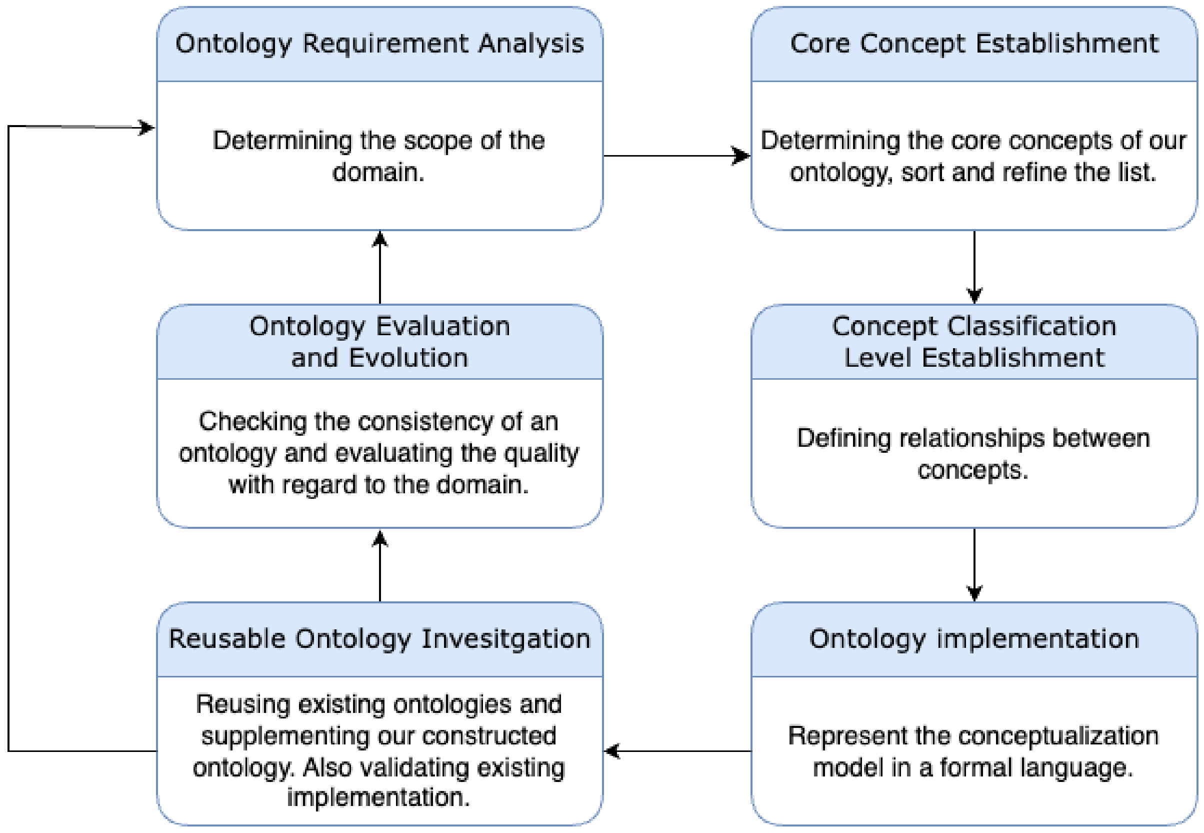
Ontology Requirement Analysis
Determining the scope of the domain.
Core Concept Establishment
Determining the core concepts of our ontology, sort and refine the list.
Ontology Evaluation and Evolution
Checking the consistency of an ontology and evaluating the quality with regard to the domain.
Concept Classification Level Establishment
Defining relationships between concepts.
Reusable Ontology Invesitgation
Reusing existing ontologies and supplementing our constructed ontology. Also validating existing implementation.
Ontology implementation
Represent the conceptualization model in a formal language.

Figure 3.1: Ontology development process.

construction in the domain of SCESS, to the best of our knowledge, no ontologies are proposed for this specific domain. However, after careful consideration of the sub-concepts of our domain, we found research supporting ontologies that provide a formal definition of these sub-concepts, typically in a different context than our domain of interest. This research is discussed in Section 3.6. With the concern of investigating reusable ontologies introducing undesirable information to our ontology and the lack of reusable ontologies for our specific domain, we decide to move this to step 5 in our development process. We intentionally still included this step in the development process, as it can still be a valuable technique to validate the constructed ontology before the actual evaluation step (step 6). Furthermore, this step can point out theoretical changes that may have not previously been considered in our implemented ontology.

Step 4 of our ontology construction process, ontology implementation, is included based on the knowledge engineering approach of ontology construction [38]. This step is to represent the conceptualized model in a formal language that can be interpreted by a machine. Although this is implicitly mentioned in the EPOST process, we thought it is important to include it as an explicit step in our ontology, as it greatly impacts the evaluation methods and tools used to evaluate the ontology.

As part of our investigation, we will construct two versions of our ontology. One version that follows steps 1-5 of the ontology development process. This version will only contain information and knowledge elicited and extracted from our industry experts at SIG. Hereafter, we will refer to this specific version of the ontology as the “industry ontology” (IO). The second version will be the product of our ontology after following step 6, where we supplement our knowledge from SIG by reusing existing ontologies. We will refer to this specific version of the ontology as the “supplementary research ontology” (SRO). The goal of keeping two versions is to track the difference in quality when supplementing existing ontologies.

In Figure 3.1, we can see that the ontology development process acts as a cycle. After going through the initial steps and implementing an ontology, the previous steps may have to be repeated. Specifically, steps 5 and 6 may lead to changes and improvements in our ontology, which requires the repetition of the earlier steps to include modification and update our definition.

3.2 Ontology Requirement Analysis

The first step of our ontology development process is to determine the goal and scope of our ontology and the domain of interest. In our research, the field of interest is software context for the evaluation of software systems (SCESS). We aim to build a domain ontology with the ability to cover the overview of background information needed to accurately and quickly aid in the process of evaluating a software system. This is so that the software context for evaluating systems becomes more structured, to facilitate the acquisition and management of knowledge using the constructed ontology.

As part of this step in our ontology construction process, we follow the detailed methodological guideline developed for ontology engineers to produce their own ontology requirements specification document (OSRD) [41]. Here are the tasks to be completed to create our OSRD:

1. **Identifying the purpose, goal and implementation language:** The goal of this ontology research is to create a formal definition of SCESS. Once we have a formal definition of software context, we will have a comprehensive guide as to what entities/concepts belong to the software context. (Using this information, we can progress to the next step of our exploration, which is to construct a custom NER dataset, explained in Chapter 4). As part of the specification process for the ontology requirements, the ontology development team conducts a series of interviews with potential users and subject-matter experts, taking into account a set of ontological demands, or the requirements that the information be represented in the form of an ontology [41]. To determine the scope of the ontology, users and domain experts are essential. To get an initial understanding of the scope of the ontology, an informal interview is conducted with a technical consultant. The scope is then narrowed down through formal interviews discussed in the following steps. The decision of implementation language is decided by the ontology developers. Our research is implemented in OWL using Protégé [1]. In Section 3.5, we discuss the implementation tools in further detail.
2. **Identifying the intended end users:** The intended users of our ontology are industry experts who provide consultations of software system evaluations. To complete this task, the ontology development team conducts interviews with users and subject-matter experts, based on potential use cases of our intended research.
3. **Identifying the intended uses:** Our research has two clear intended use cases. Our first use case is to establish a standardized and shared understanding of software context through a formal definition. The second use case of our ontology is to serve as the foundation to construct an NLP tool that will use the concepts from the ontology to extract software context from system documentation.
4. **Identifying requirements:** Obtaining the list of conditions that the ontology must meet is the aim of this task. Similar to software requirements, there are two categories of ontology requirements:
   - *Non-functional ontology requirements:* refer to the elements that the ontology should reflect, such as traits, attributes, or overall characteristics that are unrelated to the content. Examples of non-functional requirements are (a) whether the ontology must be multilingual, (b) whether the ontology must utilize a standards-based language, and (c) if the ontology should be created using a particular naming convention [41].
   - *Functional ontology requirements:* relate to the specific knowledge to be represented by the ontology and the specific terminology to be used in the ontology, which may be considered as content-specific requirements.

   For tasks 1-4, we conduct formal interviews with 3 technical consultants and 1 advisory consultant, all with expert-level experience working on software evaluation projects and have gone through the process numerous times. Based on these interviews, we extract the desired information. Specifically for this task with the first interview, we obtain the initial set of ontology requirements (both functional and non-functional), and each subsequent interview is adapted to elicit the additional/missing requirements. To identify the functional requirements, competency questions (CQs) are a common technique used in the requirement analysis phase [40, 41].

   To construct our CQs we use a middle-out strategy: start by writing down important questions that will subsequently be combined and divided into more complex and simpler questions, respectively. We use the criteria provided by Dev 101. method for ontology construction to design our

[1] https://protege.stanford.edu/

CQs [42]. Malone et al. construct a software ontology (SWO) for biomedical data analysis and provided a good baseline of example CQs [43]. In order to get the greatest results from our interviews, we utilize example CQs to think about the structure and kinds of questions we should create. To formulate our CQs based on the business process aspect of software context, we study success factors as part of business process management [44–46].

We implement additional requirements engineering techniques as part of our elicitation process. Based on the paper written by Malviya et al, we use the survey results and queries to outline the topics and design multiple CQs based on them [47]. In requirements engineering, closed questions are often used in the form of questionnaires [48], however, they can also be used within interviews. Having closed interview questions as part of our CQs bring benefits such as the ability to generate well-formed answers and allow for easy analysis [49]. Furthermore, we are adamant about the inclusion of closed questions, as it allowed for the opportunity to direct the scope of the research. Simple yes and no questions allow for the possibility to narrow down the scope of the software context that industry experts are interested in. In addition, we ask consultants to elaborate on their answers to the closed questions to understand where their answers are coming from. This method also leads to the consultants providing us with many additional sources and documentation that specifically addresses the topics our competency questions cover but in much larger detail. Processing these sources and documents allows us to efficiently identify the remaining requirements of our ontology. Appendix A lists the CQs asked during our interviews.

5. **Grouping functional requirements:** This task consists of categorizing the list of functional ontology requirements that are acquired in task 4 in the form of CQs and the answers that go with them. Grouping the CQs is beneficial since it makes it easier to identify the crucial components that the ontology must cover.
6. **Validating the set of requirements:** We want to find any potential inconsistencies, missing ontology requirements, and conflicts between ontology requirements. Our method of validation is to discuss the elicited requirements with another expert and ask for comments on the elicited results. This is the purpose of having multiple interviews, as a way to validate the answers of each expert and see which results correlate and which do not.
7. **Prioritising requirements**: The objective of this activity is to assign various levels of priority to the identified functional and non-functional ontology needs. For our research purposes, this is especially important to identify which ontology components will be most important to investigate for our NLP model, since our resources will not allow for us to investigate all concepts.
8. **Extracting terminology**: From the list of CQs and their replies found in task 4's list of CQs, a pre-glossary of words with their frequencies is to be derived in this task. For our research purposes, we use this step as a means of processing the interview results with all the elicited information, so that they can be used in the next steps of otology construction. We manually process the interview results and supplementary documents provided by the consultants to extract the terminology, relations and hierarchies.

3.3 Core Concepts Establishment

In this step, the concepts involved in the information domain ontology of the SCESS are identified. All important terms need to be listed, and domain concepts, semantics, characteristics, examples, etc. are gathered, sorted, and refined before being compiled into a concept summary table. The conceptual model's core concept dictionary must satisfy the criteria for being clear and comprehensive in its coverage of the software context domain knowledge. This step is made quite simple as in the previous Section(3.2) we have already processed the elicited requirements from interviews, documentation, etc. and extracted the terminology. Based on this organized terminology, we can adopt them as the concepts of the core concept dictionary. Table 3.1 displays a small portion of the core concept dictionary of our software context ontology, which includes information mainly under business process and software quality factors. (The concepts correspond to the classes of our ontology. To view all the concepts, inspect the classes of our final ontology that is available on our project repository).

Concepts		
Software Quality Metrics	Accessibility	API Documentation
	API Maturity	Branches
	Development Practices	Commit Changes
	Maintainability	Communication Protocols
	Performance Efficiency	Current Challenges
	Reliability	Data Formats
	Scalability	Data Scalability Metrics
	Security	Dependencies
	Technology	Endpoints
	Test Environments	Error Messages
	Test Types	File Uploads
	Tools	Frameworks
	Performance Metrics	Hardware
	Policy	Hotfixes
Business Process	Business Model	Project Management Practices
	Competitors	Stakeholders
	Definition of Done	Strategic Roadmap
	KPI	Userbase Information
	Organisation	

Table 3.1: Portion of the core concept dictionary extracted purely from experts

3.4 Concept Classification Level Establishment

The purpose of this step is to analyse and define the potential links between the domain concepts; both implicit and explicit links need to be considered. We initially define the hierarchical relationships between the concepts. This is to categorize and link the concepts to other concepts that share the same characteristics and will simplify the next step of connecting non-hierarchical relationships. To determine the hierarchical relationship of idea classification, there are three methods: top-down, bottom-up, and middle-out. Since in previous steps, the core concept dictionary has been developed, for our research we employ the middle-out approach to create the concept classification hierarchy. This means that we start from a middle level of abstraction based on our concepts and expand both upwards and downwards to include more general and specific concepts. This approach also allows for more flexibility and adaptability as new information can be obtained with further elicitation sessions in between the construction of the hierarchy. Additional concepts and related terms that could not be properly defined are obtained using the Software Engineering Body of Knowledge (SWEBOK) [5] and the ISO standard for Systems and Software Quality Requirements and Evaluation (ISO 25010:2017) [50].

The framework of ontology is concept hierarchy, which has to be expanded and enhanced by attributes and relationships [40]. The concept hierarchy is a structure with a semantic capacity that can better organize and explain information. We may alter, add, and remove ideas and the connections between them in the concept dictionary as ontology development progresses. Figure 3.2 displays the hierarchical classification of our ontology concepts. Note that all these concepts are based on the requirement elicitation interviews and processed documents provided by industry experts; no reusable ontologies have been investigated or included as of this step. Not all concepts from Table 3.1 are present in Figure 3.2, as many are not constructed with a hierarchical relationship with the concepts *Software Quality Metrics* and *Business Process*, but are connected using other relationships. Once the concept hierarchy is created, we can use an `isA` or `isPartOf` relationship to express the taxonomic relationship between concepts in our ontology.

The next step is to define the non-taxonomic relationships between the concepts in our full dictionary. This is important to make sure our ontology has strong semantic capabilities. The concepts behind SCESS are defined under the two main concepts *Business Process* and *Software Quality Factors*. Under-

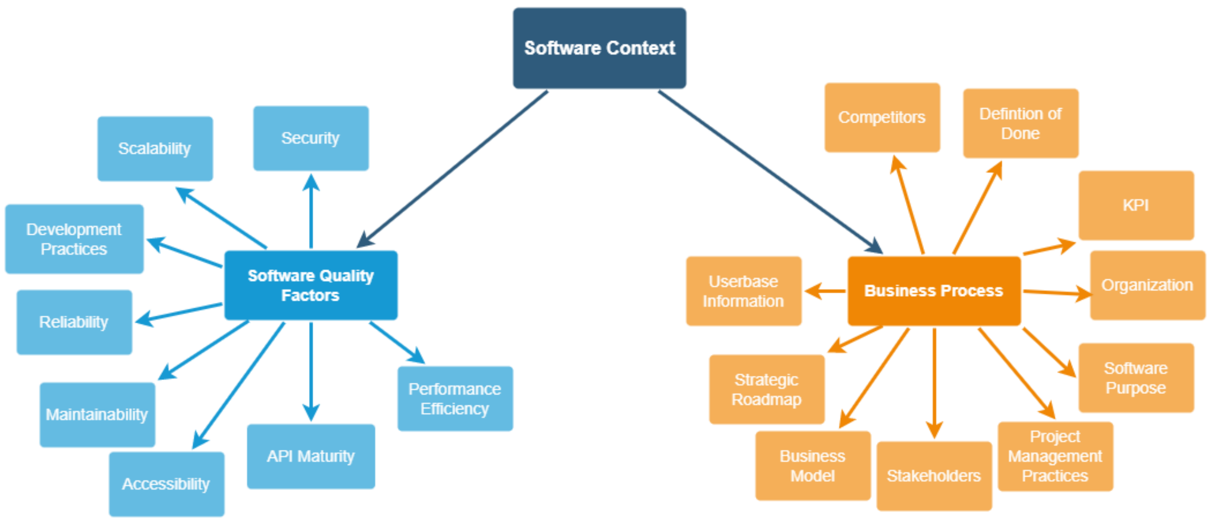

Figure 3.2: 3-level hierarchy of concept classes for our ontology. Concepts based purely on elicitation sessions and documentation from experts.

standing the business process is essential to comprehend how a software system is expected to operate within the context of the business as a whole. Researchers do not consider this aspect of a software system, as it is primarily a concern for industry-related software evaluations. In order to effectively formulate relationships between business process concepts, we study research on critical success factors of a business process [44–46]. These success factors provide a good guideline as to what factors of the business process we need to consider when evaluating a software system. Hence, this field of research provides a good alternative, as we did not come across a set standard for business process evaluation and knowledge. In research, but especially in our investigation of the industry, software quality factors are the main technique used when evaluating software systems. This is because they provide a good foundation of characteristics to investigate when assuring and evaluating the quality of a software system. To provide a comprehensive overview of the non-taxonomic relationships between the software quality factor sub-concepts, we use the SWEBOK [5] and ISO 25010:2017 [50]. The ISO standard outlines the majority of the factors that the collaborator's (SIG) model is based on. While the SWEBOK provides a well-rounded overview connecting the concepts and theory that is not directly illustrated in the ISO standard.

3.5 Ontology Implementation

This step aims to explicitly represent our conceptualized models in a formal language for us to make the ontology machine readable. This allows our ontology to be utilized by a computer for processing and achieving interoperability among various systems, as explained by Yun et al. [38]. There are numerous formal ontology languages such as DL (Description Logics), RDF (Resource Description Framework), RDFS (RDF Schema), Ontolingua, Cycl, OWL, etc. [38].

The World Wide Web Consortium (W3C) establish a more expressive Web Ontology Language (OWL) to increase the RDF Schema's restricted expressiveness [51]. Kalibatiene and Vasilecas conduct a literature survey to create a comparison of the four most popular ontology languages (KIF, OWL, RDF + RDF(S) and DAML+OIL) [52]. Their results indicate that, as seen by the research, OWL claims to be the most popular language, having the biggest community and the most spreading and applicable semantic web language. The drawback with this language is that at the time this paper [52] is written, OWL lacks supporting tools. However, it has been many years since this paper [52] is published and many additional and useful tools have been developed for OWL and the popularity has only further increased. One of the tools that assist in the development of ontologies is an editor. There are a few publicly available editors we can use, such as Protégé. Protégé is a free open-source ontology editor framework used to construct OWL ontologies and has plugins available for many of these tools [53]. Nowadays, there are also publicly available tools that can be used to validate OWL ontologies by checking for consistency and defects by evaluating the OWL code. We discuss some of these evaluation tools in Section 3.7.

Additionally, numerous ontology development papers for various domains have also used OWL in coordination with Protégé as an editor to construct their own domain-specific ontologies [40, 54, 55] Recent Ontology learning research also uses OWL with Protégé as tools to construct ontologies [56, 57]. Based on the popularity and, consequently, the availability of documentation/resources, we select OWL as the language to construct our ontology. We use Protégé as the ontology editor, as it functions well with OWL code and provides many additional plugins and tools to edit our ontology.

Once we select our Ontology language and editor, we familiarize ourselves with the components used by OWL to define the different ontology components. In OWL, a class is a group of individuals that share certain characteristics [40]. Every knowledge point typically has its own class. According to the knowledge level, the upper knowledge point is the parent class, and the lower knowledge point is the subclass. The most abstract entity concept is represented by the highest level class. Each subclass is a more focused and condensed entity notion than its parent class, and it inherits the abstract properties of its parent class. The next level subclass is gradually specified in this process in accordance with the upper-level abstract parent class that has been predefined. Classes directly correspond to the concepts we define in Section 3.3 and follow the hierarchy we define in 3.4.

In ontology, a class can have individuals. An individual refers to a specific instance that belongs to a particular class within a given domain. It represents a unique entity or element within the ontology. An individual is an instantiation of a class and can possess various attributes and relationships defined for that class. A class also has two types of attributes: numerical attributes that provide its structure information and object attributes that describe the relationships between other classes. In OWL, the numerical attribute is known as a datatype property, and the object attribute is known as an object property. Numerical values are used to express the structural information of a class using a numerical attribute (datatype property in OWL). It indicates a particular quality or attribute of the class that may be assessed quantitatively or represented by a number. Examples of numerical attributes are `Weight`, `Length`, `Temperature` and `Age`. These attributes offer numerical data about the class and can be given values in the form of integers, decimals, or other numerical formats.

The connections between classes are defined by object attributes, referred to as object properties in OWL. It indicates a link or association between various ontology class instances. The connections, dependencies, or interactions between the instances of the classes are captured by object attributes. Some examples of object attributes include `hasParent`, `hasPart`, `hasColor` or `hasOwner` These characteristics explain the links or relationships between examples of various classes. For instance, `hasParent` may create a connection between an instance of the `Child` class and an instance of the `Parent` class. Each attribute has a unique attribute name that designates the class it refers to. Representing an ontology can be summarized as the process of defining classes and creating their attributes.

Additional to the ontology components previously mentioned, ontologies can also include cardinality and logical axioms. Cardinality restrictions are used to limit the number of instances or values that can take part in a relationship or possess a certain property. Meaning, based on the chosen attribute, they set limits on the number of instances that belong to a class. Logical axioms establish logical relationships and constraints within the ontology. These axioms include logical operations like equivalence to express complex logical relationships between concepts. There are multiple purposes for the inclusion of these components. Such as providing additional clarity to the ontology, validating the ontology against data instances, and using reasoning engines to make deductions. However, after a thorough discussion, we decide to exclude the use of these components. Cardinality would not add too much to our research because cardinality focuses on limiting and specifying much lower-level relationships and attributes, however, the purpose of our ontology is to provide a high-level overview of our domain and an ontology that can be applied to aid in any research related to the domain. Cardinality would have added a layer of complexity to our ontology as we work with hundreds of classes and the resources required to define the cardinality outweigh the potential knowledge gained. We also decide to exclude the use of logical axioms, equivalence, and disjointedness as we decide to design our classes such that there are no clear equivalences and disjointedness. This is to reduce the complexity of the overall ontology, as we are working with a large number of concepts.

3.6 Reusable Ontology Investigation

One of the fundamental purposes of building an ontology is that it has the ability to be reused and shared. Checking if a current ontology is scalable and reusable is a crucial step in making sure that this ontology feature can be utilized. Reusing existing ontologies is a step that can be used to efficiently supplement an ontology and verify the quality of our ontology. We can supplement our ontology by adapting the knowledge presented in existing ontologies. While we can verify the quality of our ontology by inspecting existing ontologies from published research with similar subdomains as our ontology, and compare if the knowledge presented in our ontology is similar to that of the published research. Furthermore, it avoids the production of redundant research in the field.

When investigating software engineering ontologies, the topic of software context ontologies is an area of research without much to currently offer. Therefore, since there is no "software context ontology" available, we decide to investigate concepts that are lower level in our IO. Our investigation is split based on the two main sub-concepts displayed in Figure 3.2: Business process and software quality factors.

The business process is crucial to a company's competitiveness, thus there is a constant need for a company's business processes to be carried out efficiently. Kim and Suhh aim to create a semantic business process space, which stores semantic information such as various generic and specific ontologies, and to demonstrate how such semantic information can be used [58]. Within their research, they introduce various sub-ontologies related to the business process domain. For our research, we use knowledge introduced in their ontologies with specific interest to their Business Process Ontology (BPO), Organization Unit Ontology (OUO), Service Ontology (SO). These contain relevant concepts and relationships that provide valuable context to a software system. Domain Knowledge Ontology (DKO) and Resource Ontology (RO) include knowledge that is out of scope with software context. They introduce domain knowledge that is helpful for performing business processes, such as payment conditions, shipment conditions, etc. and the input and output of the business's resources respectively. Outside this paper, no relevant ontologies related to the business process are found.

Investigating software quality ontologies required a few more steps. Our first step is to investigate complete ontologies that are focused on the software quality domain. Kayed et al. focuses on concepts and terminologies related to Software Product Quality Attributes [59]. To identify the key ideas for SWQAs, they do various tests and their findings indicate that several terms are often used to describe these features. Their research provides an additional overview of attributes and concepts that should be focused on when investigating existing ontologies. Motogna et al. present an ontology created for the ISO25010 standard that serves as the foundation for the methodology used to assess the quality of software [60]. Blas et al. present another ontology for documenting a quality scheme based on the product quality model of ISO/IEC 25010 [55]. The model tries to make clear the characteristics that are often found in a product, as well as its significance. The framework of an ontological model is presented in a publication by Bajnaid et al. that describes and defines the operational and domain knowledge of Software Quality Assurance (SQA) [61]. The major sources of the terminology and semantic relationships used in the suggested SQA conceptual model are international standards (SWEBOK, IEEE, and ISO). Based on these existing ontologies, we extract concepts, relationships, attributes, and entire sub-ontologies, which are then incorporated into the SRO.

Additionally, we investigate ontologies for the sub-concepts of maintainability, performance efficiency, security, and reliability. The ontology that Ruiz et al. describe, works under software maintenance management. Although not directly within the scope of software maintenance, we apply some concepts for our own ontology. Our research for performance efficiency ontologies only leads to models that are mostly out of scope, so we focus on extracting relevant concepts from the papers [62–64]. The remaining ontologies found through our investigation are in scope with the topics of software maintainability [65], security [66, 67], and reliability [68]. From those papers, just as previously mentioned with software quality ontologies, we adapt concepts, relationships, attributes, and full parts of the ontologies to our SRO. It is important to note that the other sub-concepts of software quality (i.e. compatibility, usability, etc.) either did not provide relevant research or are given less importance after consultation with industry experts. Therefore, we decide not to include them and focus on our selected sub-concepts.

3.7 Ontology Evaluation

Once the ontology is constructed, we need to evaluate the quality of the ontology and find potential areas of improvement. Ontology evaluation is a procedure that assesses all of the variables influencing an ontology's quality using a variety of evaluation indicators and scientific evaluation techniques [69]. Ontology evaluation entails determining whether the goals established during the requirement analysis stage have been achieved, as well as the accuracy of the ontology that has been created. Evaluation techniques, evaluation indicators, and evaluation tools are the three components that makeup ontology evaluation activities [40]. Gruber's [22] proposes five criteria for assessing ontologies: clarity, consistency, extensibility, minimum coding preference, and minimal ontology commitment. Today, these criteria are widely accepted in the field of ontology evaluation. Ontology assessment techniques that are often used include these "Golden-Standard" evaluation techniques and evaluation techniques based on statistical analysis, logic or rules, and metric systems. Currently, Ontoclean, Core, OOPS!, TEXCOMON and other tools for ontology assessment are often used [69].

3.7.1 Internal Consistency Check

Tools for semantic-based ontology reasoning, such as consistency checking, concept satisfiability, classification, and realization, infer the consistency of the content's logical sequence using the original set of axioms [70]. Concept satisfiability checks whether it is logically possible for a concept to have instances; classification computes hierarchical relationships between classes; and realization identifies concepts to which specific individuals belong [70]. Consistency checking ensures that the ontology does not contain contradictory facts. In other words, logical conclusions may be deduced from a starting set of axioms by semantic reasoners. To check for consistency and other reasoning results, we evaluate our model using HermiT 1.4.3.456 reasoner in Protégé 5.6.1.

3.7.2 Context Pitfall Detection

Having a checklist of frequent mistakes that previous developers have done in the past is one of the most popular methods for reviewing ontologies [71]. As a result, the developer evaluates the ontology under construction against such a list, detects the pitfalls, and corrects them. Poveda-Villalón et al. have constructed a pitfall detection tool called OOPS! [71]. Their method does not claim to be another checklist; rather, it extends studies where modelling issues have already been highlighted by adding additional dangers discovered through an empirical analysis of existing ontologies. OOPS! is publicly available as an automatic detection tool for ontology defects, and lists 29 ontology pitfalls. To evaluate our ontology from six different perspectives—human comprehension, logical consistency, modelling issues, ontology language specification, real-world representation, and semantic applications—we use the tool. The results are divided into three categories, depending on their severity: critical, important, and minor.

3.7.3 Ontology Verification

The challenge with a good deal of ontology evaluation methods is that there is a lot of potential to create biased evaluation results due to human reasoning. Therefore, in our research, we investigate criteria-based evaluation methods to outline the quality of our ontology objectively. FOCA is a mature and still novel methodology for ontology evaluation [72]. Three key elements are considered while presenting their novel technique for ontology assessment: i) it is based on the Goal Question Metric (GQM) approach for empirical evaluation; ii) Each ontology is evaluated based on the type of ontology; iii) the objectives of the methodology are based on the roles of knowledge representations paired with particular evaluation criteria. The methodology is assessed using several ontologists and ontologies from the same domain. FOCA consist of 3 verification steps displayed in Figure 3.3.

Here are further details about each step:

1. **Ontology Type Verification:** In this part, we differentiate our ontology between two types [72]: Type 1 (a domain or task ontology) and Type 2 (application ontology). Type 1 describes concepts from more generic domains (i.e. medicine or vehicles) and generics tasks. While Type 2 describes concepts from a very specific domain (i.e. student registration system at a particular university for the computer science programme). The purpose of differentiating between the two types is that certain evaluation questions in step 2 are irrelevant depending on the type of ontology, and therefore will not be considered.

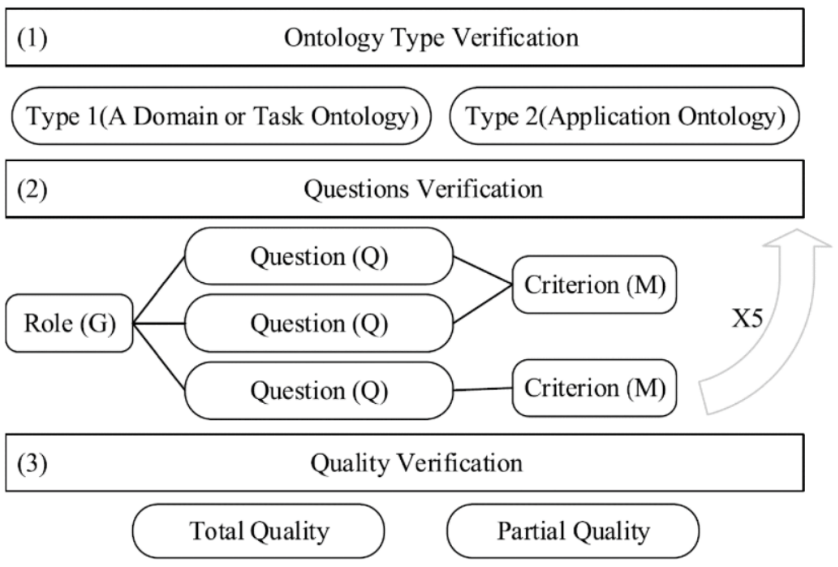

Figure 3.3: The FOCA methodology. Diagram for this paper: [40] however methodology based of this paper: [72]

2. **Questions Verification:** In this step, our evaluator will answer questions following a goal/question/metric (GCM) approach. 12/13 questions are answered based on the ontology type, where the paper [72] describes how to verify and provide a grade between 0 and 100. The questions are described in Table 7.3.
3. **Quality Verification:** Once the questions are answered and verified by our evaluator, we took the corresponding grades to calculate the quality of the ontology. The following formula is used:

$$\widehat{\mu}_i = \frac{\begin{array}{l}\exp\{-0.44 + 0.03\left(\text{Cov}_S \times Sb\right)_i + 0.02\left(\text{Cov}_C \times Co\right)_i + 0.01\left(\text{Cov}_R \times Re\right)_i\} \\ +0.02\left(\text{Cov}_{Cp} \times Cp\right)_i - 0.66LExp_i - 25(0.1 \times Nl)_i\end{array}}{\begin{array}{l}1 + \exp\{-0.44 + 0.03\left(\text{Cov}_S \times Sb\right)_i + 0.02\left(\text{Cov}_C \times Co\right)_i + 0.01\left(\text{Cov}_R \times Re\right)_i\} \\ +0.02\left(\text{Cov}_{Cp} \times Cp\right)_i - 0.66LExp_i - 25(0.1 \times Nl)_i\end{array}} \tag{3.1}$$

i is in reference to the evaluator. Cov_S is the mean of the grades from Goal 1. Goal 1 contains 3 sub-questions, there we calculate the mean between the three sub-questions. Hence, Cov_S is the mean between Question 1, Question 2 and Question 3. Cov_C is the mean of the grades from Goal 2. As our ontology is a Type 1 domain ontology, Cov_C is the mean between Question 5, Question 6, and Question 7. Cov_R is the mean of the grades from Goal 3. Cov_Cp is the mean of the grades from Goal 4. $LExp$ is the variable for evaluator experience, with 1 being very experienced and 0 being not experienced at all. Nl is 1 only if some Goal is impossible for the evaluator to answer all the questions. The FOCA methodology also allows for the evaluation of each of the roles of knowledge representation: Substitute, ontological commitments, intelligent reasoning, efficient computation, and human expression [72]. $Sb = 1, Co = 1, Re = 1, Cp = 1$ are parameters that change the roles, and they are all equal to 1 as we want to evaluate total ontological quality. The value output by the formula produces a score between 0 and 1 where 1 is the highest level of quality.

Chapter 4

Software Context Dataset

In this Chapter, we explain how we construct our Software Context Dataset (SCD). We specifically cover how the data is collected, and then explain the steps taken to label the data for Named Entity Recognition (NER). Finally, we present our data management plan (DMP).

4.1 Data Collection

In Chapter 3, we discuss the entire process of building ontologies. During the process, we directly contact industry experts, and during interviews, we ask them to rank the software context concepts they discussed. Specifically, we ask them, when reading through software documentation, which are the top 4 high-level concepts they would like information about. This question is presented because certain concepts of software context are better extracted from source code than from documentation. Therefore, the experts also consider what concepts are less investigated for the task of information extraction from software documentation. From these interviews, the two highest-ranking concepts are business process and scalability. Once the ontology is constructed (the final results are discussed in Chapter 7.1), we can select the specific sub-concepts of the business process and scalability concepts that will act as the labelled entities in our dataset. We once again discuss with industry experts which sub-concepts provide valuable software context knowledge from system documentation. Based on the available resources, we decide to focus on 3 business process entities (`Userbase_Information`, `Software_Purpose`, and `Internal_Organization`) and 3 scalability entities (`Transaction_Scalability`, `Development_Scalability`, and `Data_Scalability`). Additionally, we conclude that this entity information would be a lot less useful if we did not know which software or company the information corresponds to in the system documentation. Hence, we include two additional entities `Software_Name` and `Company_Name`. Table 4.1 displays all the entity labels and the descriptions that are contained within our SCD.

Once the entities are selected, we start the data collection process. The first approach we took is to construct a Web Crawler to start with a list of initial links and crawl through the relevant links. Since we only have the resources to construct a small data set, this approach proves ineffective because the crawler extracts many irrelevant links that did not contain valuable data related to our entities. Therefore, we rework our approach. Here are the steps we take:

1. The first step is to construct all the search queries that are input into a search engine. For this step, the queries follow the structure: `[Company]` + `[Document Type]` + "for" + `[Entity Query]`. `Company` consisted of the top 75 Software as a Service (SaaS) companies as of February 2023[1]. `Document Type` consists of white papers, documentation, case studies, and reports. Finally, `Entity Query` is a list of topics and their synonyms related to the entity description itself. In Appendix B, we list the Entity Query for the scalability entities, as these entities are less clear from the name.
2. Once the search queries are constructed, run the queries into some search engine or search engine API. For our data, we use the Google Search API and extract the first 10 URLs for each query.
3. For each extracted URL, we manually read through the document types and extracted sentences that related to the appropriate topic. This allows for consistency and accurate data collection, however, the tradeoff is the quantity of data.
4. The extracted sentences are processed so that each data point has at most 4 sentences (with a few

[1]https://www.datamation.com/cloud/saas-companies/

Table 4.1: This table provides descriptions for our 8 chosen entities of our annotated dataset. The queries in Appendix B will provide a more clear idea as to what type of data the scalability entities contain.

Entity Label	Description
Software_Name	States the name of a software.
Company_Name	States the name of a company.
Userbase_Information	Provides information about the userbase of a company or software (e.g. amount of users, demographics, explicit mentions of users, etc.).
Internal_Organization	Provides insight into the internal organization of a company (e.g. specific teams, departments, job/team roles, etc.).
Software_Purpose	Indicates what the purpose of a specific software is.
Transaction_Scalability	Highlights characteristics related to transaction scalability (e.g. scaling strategy, latency and elasticity information, etc.).
Development_Scalability	Highlights characteristics related to development scalability (e.g. development practices, coding standards, etc.).
Data_Scalability	Highlights characteristics related to data scalability (e.g. data infrastructure, storage capacity, volume monitoring, etc.).

exceptions). This is so that each data unit provides sufficient context to the entities present.

4.2 Annotation Schema

Our dataset contains the 8 entities listed in Table 4.1, 3 business process-related entities, 3 software development-related entities, and two name entities. We use the following annotation guideline that is cross-checked with the help of an industry expert:

- `Software_Name`: Any mentions of software names are annotated. Cases, where the text is both a company name and also a software name, will require annotating the text based on the context of the sentence. For example, "slack" can be referred to both as the software or company, therefore we look at the word in the context of the sentence and see if the sentence refers to the software or the company. Additionally, if the software is mentioned as something like "slack's user management software", which contains both a company name followed by the software, we annotate the full text as a software entity, including the company name. Software names are structured as nouns in the sentence, so we only label nouns for this entity.
- `Company_Name`: Any mentions of company names are annotated. If a company is mentioned along with software, then the full text is labelled as the name of software, as previously explained. We annotate a company as a distinct entity in combination with the other entity types other than `Software_Name`. For example, text like "google's American userbase" contains information from `Company_Name` and `Userbase_Information`, and will be annotated as "google's" and "American userbase" respectively. This is the case for all entities other than `Software_Name`. Company names are structured as nouns in the sentence, so we only label nouns for this entity.
- `Userbase_Information`: Any mentions of the userbase and the information about their users. This includes mentions of the words "user" and "customer" and the information surrounding these words, as well as mentions of the userbase demographics, statistics, and user roles/privileges. Note that this will typically be in the context of a software or company. Information related to the userbase is typically a combination of numbers, adjectives, and nouns in a sentence.
- `Internal_Organization`: Any mentions that provide insight into the internal organization and members of a company/organization. This is typically the names of specific teams, specific job roles/titles, departments, or even general terms used to describe parts of an organization. This entity is mostly structured as nouns in the sentence, so we only label nouns for this entity.
- `Software_Purpose`: Any mentions of the purpose of software/ what a software is used for. However, for this entity, we only label the verb in the sentence used to describe the purpose of the software. For example, if we have the sentence "SimScale's cloud-based simulation software gives engineers across all industries the ability to test their design prototypes without having to build them." the

mention of the software's purpose is in "to test their design prototypes without having to build them". We only annotate the verb "test". Typically, when a sentence is structured like this, the annotated verb comes right after the word "to". Therefore, for this entity, we only annotate verbs.

- `Transaction_Scalability`: Any mention of information in the text that relates to transaction scalability. The queries listed in Appendix B provide a large overview of what types of topics/keywords to look for when annotating this entity. This entity is mostly structured as adjectives and nouns, but occasionally as verbs as well, in a sentence. For instance, "scale" is a verb commonly associated with this entity.
- `Development_Scalability`: Any mention of information in the text that relates to development scalability. The queries listed in Appendix B provide a large overview of what types of topics/keywords to look for when annotating this entity. This entity is also mostly structured as adjectives and nouns, but occasionally as verbs as well, in a sentence.
- `Data_Scalability`: Any mention of information in the text that relates to data scalability, data infrastructure and the technologies used. The queries listed in Appendix B provide a large overview of what types of topics/keywords to look for when annotating this entity. This entity is mostly structured as nouns in a sentence.
- General rules: There are no overlapping entity annotations. This means if the text is annotated as one entity, a substring or the full text cannot be annotated as a second entity. We prioritize which entity is annotated based on the context of the full sentence. The annotator should ask themselves, what is the context of the sentence, and what is the main information the sentence is trying to convey?

4.2.1 Annotation Tool and BIO Tagging

For the construction of our dataset, we collect data using the process described in Section 4.1, and label the data using Doccano [73]. Doccano is an open-source text annotation tool, and it can be used to create labelled datasets for various natural language processing (NLP) tasks, including NER. Doccano exports the dataset into JSONL format[2], which is not suitable input for most modern pretrained language models. Therefore, we convert the JSONL data into the BIO tagging scheme. This can be done by using the doccano-transformer [3]. In the BIO tagging scheme, we indicate tags using "B" when we want to show the first word of an entity, "I" when we want to indicate the words inside an entity, and "O" to indicate non-entity words. Table 4.2 shows an example of the BIO tagging scheme being applied to a sentence in our dataset.

Table 4.2: This table shows an example of how the BIO tagging scheme looks on the tokens of the sentence, "SimScale's cloud-based simulation software gives engineers across all industries the ability to test their design prototypes without having to build them.".

Token	BIO Tag	Token	BIO Tag
SimScale's	B-Software_Name	ability	O
cloud-based	I-Software_Name	to	O
simulation	I-Software_Name	test	B-Software_Purpose
software	I-Software_Name	their	O
gives	O	design	O
engineers	B-Internal_Ogranization	prototypes	O
across	O	without	O
all	O	having	O
industries	O	to	O
the	O	build	O
		them.	O

[2]https://jsonlines.org/
[3]https://github.com/doccano/doccano-transformer

4.2.2 Using ALBERT to Check and Modify Annotation Guidelines

Our small dataset is annotated by one annotator (the author of this thesis) who is a Software Engineering master student. To check the quality of annotations, the guidelines are discussed with an industry expert in the domain of software evaluation. Small samples of collected data are also presented to check if the scope of the data corresponds to the entities that are being labelled (particularly for the more ambiguous scalability entities).

In order to check whether a model can learn from our annotations, we can fine-tune a lightweight model and inspect the behaviour of the predictions. ALBERT is the lite variant of BERT [74]. It simplifies the architecture of BERT by reducing the number of parameters to optimize memory efficiency. Garima et al. [75] provide a comparison study on software requirement NER, where they indicate that the average training time for ALBERT is 2.9 minutes, whereas BERT and RoBERTa variants took over 9 minutes. Therefore, ALBERT requires a small training time and obtains state-of-the-art results, making it the ideal model to quickly train using our labelled data and check if the predictions are being formulated correctly. Note that we did not use ALBERT for our main NER model to extract entities for our knowledge graph. We only used it to check if the model is learning properly based on the way our data is annotated. The reason we do not use ALBERT as our main NER model is that on its own it would require a much larger dataset to obtain good results. Therefore, we use another approach and model for our main NER model, which is discussed in Chapter 5.

To check the annotation guidelines, we fine-tuned the Albert-base-v2 from Hugging Face[4] with 80% of the data and used the other 20% of the data to test the NER prediction capabilities of the model. By observing the incorrect predictions from the fine-tuned Albert-base-v2, we notice that in certain entities there are inconsistencies in the way we are labelling. We noticed this because the model is making mistakes due to these inconsistencies. These inconsistencies are then corrected and specified in our annotation guideline. An example would be the way we originally annotated `Software_Purpose` entities. Looking at the example sentence "SimScale's cloud-based simulation software gives engineers across all industries the ability to test their design prototypes without having to build them.", the purpose of the software is mentioned in the substring "to test their design prototypes without having to build them". The older version of the guidelines meant that "test their design prototypes" and "without having to build them" or the entire substring itself would have been labelled. Firstly, this is a very large string and a NER model will not be able to recognize such an entity, as annotating the purpose of software in this manner will provide too much inconsistency between all the data units. Meaning, because we will not provide enough units of data for the model to learn how to label such large text. This becomes evident when the model is not able to predict any `Software_Purpose` entities correctly, and the predictions that are made are quite off. Therefore, studying the inconsistencies and predictions of the model, we modify the annotation guideline to label the verbs in the sentence; in this case "test" is the verb used to describe the purpose of the SimScale software. Hence, we use ALBERT to check if our annotations had issues, we then update our guidelines and lastly correct the annotations.

4.3 Data Management Plan

For our research, we define our Data Management Plan (DMP) using the DMP template from Horizon 2020 FAIR [76]. Our goal is to provide our data as findable, accessible, interoperable, and reusable (FAIR) by using the DMP and adhering to this framework.

1. **Data Summary:** For our research, there is no available dataset that contains entities related to software context, software company business process, software quality factor or related topics. Therefore, we construct the SCD through the processes described in the previous Sections of this chapter. Our dataset contains 745 total data samples, where most data samples range in between 1–2 sentences, and nearly no sample exceeds 4 sentences. When calculating the average of the distribution, there are roughly 518 labels per entity. The distribution of the labelled entities within these 745 data samples is displayed in Figure 4.1. We attempt to keep the number of entities similarly distributed throughout. `Company_Name` has fewer entities but is a much simpler entity to learn based on in structure in text and many `Comapny_Name` text is combined with `Software_Name` as explained in the annotation guideline in Section 4.2.

[4]https://huggingface.co/albert-base-v2

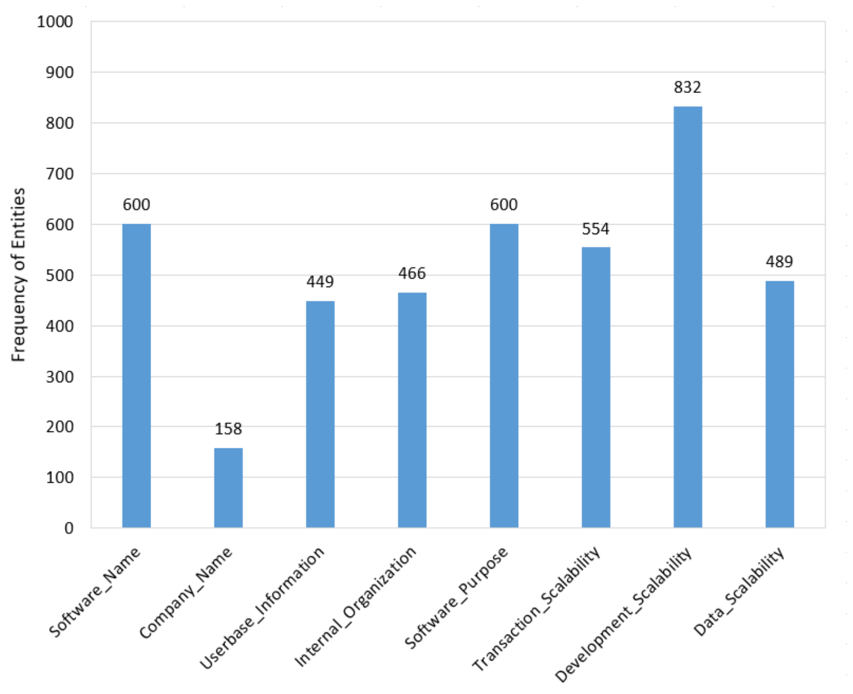

Figure 4.1: Distribution of entities in the Software Context Dataset.

2. **FAIR Data:** Our data is stored in 5 JSON files. All 5 files contain the same data, however, we applied 5-fold cross-validation when our models are trained, therefore each file contains the same data in different groupings. In 5-fold cross-validation, the data must be split into training data and testing data in 5 variations. Each JSON file contains a “train” and “test” key. The values within these keys are a list of strings, which contains each data sample in BIO tagged scheme. The dataset is made accessible in the thesis repository[5], including in the original JSONL format output by the Doccano Annotation Tool. Along with this, the URLs of all the data samples will also be made available.
3. **Allocation of Resources:** There is a very low cost of making the data FAIR for the SCD. As each file only contains 383 KB of data, very little network storage space is required and can be published on a GitHub repository with no additional costs. It will be our responsibility to organize and describe the data for future use. While SIG will be responsible for providing storage space and making decisions on how long the data will be stored within their company network.
4. **Data Security:** Multiple copies of the data have been kept throughout the research process on various hard drives and a backup cloud server. Additionally, they have been stored in Git commits in our research repository. The text within the dataset is taken from publicly available sources, and therefore our dataset has no privacy concerns.
5. **Ethical Aspects:** Although the data is provided by publicly available sources, many contain copyright restrictions. However, since the data is being used to train machine learning models and is not being used for commercial use, but instead for academic purposes, the data will be ethically used. EU copyright rules permit the use of copyrighted data for our non-profit research purposes. Furthermore, this thesis describes how the data is used and provides full transparency of where the data is extracted from.
6. **Other:** Not applicable.

[5]https://github.com/athuln-99

Chapter 5

Experimental Setup for Named Entity Recognition

In Chapter 4 we explain how our Software Context Dataset (SCD) is constructed. This chapter discusses our experimental setup to fine-tune a pretrained language model (PLM) using this dataset. Through the experimental setup, we specifically train our PLM for a few-shot Named Entity Recognition (NER) task. Hence, the goal of our experimental setup is to fine-tune our PLM so that it can learn to recognize the SCD entities and extract these entities from any system documentation.

5.1 Tools and Technologies

The SCD is a dataset that only contains 745 data samples, with roughly an average of 518 labels per entity. Large PLMs are trained on millions of samples of data to obtain their state-of-the-art results. Furthermore, our data set is still quite small to obtain good results by just fine-tuning a PLM. With this consideration in mind, our research focuses on using few-shot entity recognition technologies. Specifically, we construct the RoBERTa model, which is pretained using Noisy Supervised Pretraining (NSP), and in combination, we add a linear classifier to the last layer of the architecture. In Section 2.4, these technologies are explained in detail. We selected these technologies specifically based on the research paper written by Huang et al. [35]. They provide a comprehensive study for few-shot NER, where they use the RoBERTa model as a baseline and apply additional machine learning (ML) techniques and see how the F1 score of the model changes when it is fine-tuned with very small amounts of data (including just 5 data samples). Based on the results, the combination of NSP and a linear classifier obtain the best results when using 100% of a dataset, therefore we chose to use the same implementation. The authors have published their code on GitHub [1].

In order to prepare the dataset into suitable inputs, we use Python and the standard libraries. We use the deep learning framework, PyTorch, to set up our models, load the dataset and train/fine-tune the model. The RoBERTa transformer that we use is available on Hugging Face library [2] along with the necessary tokenizers. For our training, we use an NVIDIA GeForce GTX 1650 Max-Q GPU. This proves sufficient for the training parameters that are used (specified in Section 5.4).

5.2 5-Fold Cross Validation

Cross-validation is a statistical technique for comparing and evaluating learning algorithms that involve splitting the data into two sections: one for learning or training a model and one for testing the model [77]. The two sets must overlap in subsequent rounds during typical cross-validation so that every data point gets a chance to be validated against. In our research, we apply 5-fold cross-validation. In 5-fold cross-validation, the data is initially divided into 5 segments or folds of equal (or nearly equal) size. Then, 5 new iterations of training and testing are carried out, with each iteration holding back a different fold of the data for testing while using the other 4 folds for learning. For each training iteration, we use 80%

[1]https://github.com/few-shot-NER-benchmark/BaselineCode
[2]https://huggingface.co/docs/transformers/model_doc/roberta

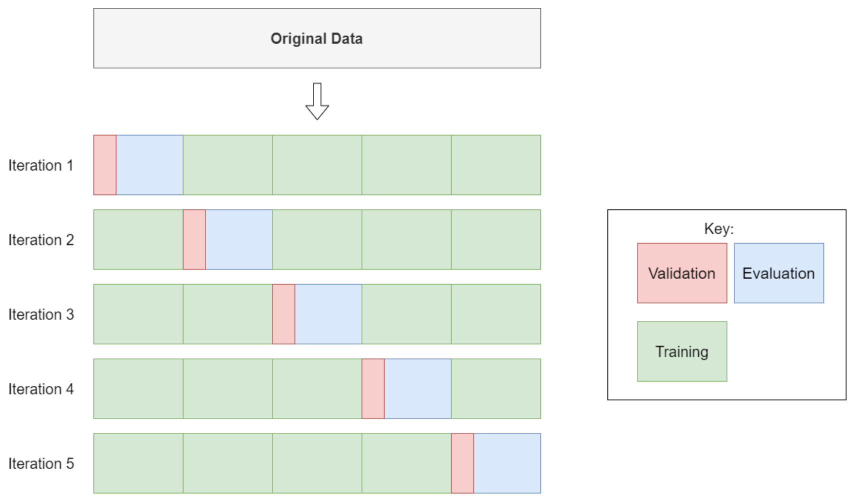

Figure 5.1: Diagram of 5-fold cross-validation. The red boxes are validation data partitions, the blue boxes are evaluation data partitions, and the green boxes are the training data partitions.

of the SCD for training data (used by the model to learn) and 20% for testing data (data that the model does not learn from). In our case, 20% of testing data is used for both model validation and evaluation. These topics are discussed in Section 5.5 and 5.6. Therefore, from the testing data, we also take 20% for validation and 80% for evaluation. This means that in reality, the data split of the SCD for each iteration looks more like this: 80% training data, 4% validation data, and 16% evaluation data. Looking at Figure 5.1, it is clear how we partition/split our original full dataset for 5-fold cross-validation. It is important to understand that the iterations are independent of each other, and for each iteration we train, validate and evaluate a different version of the NER model.

We implement 5-fold cross-validation because we are working with a very small dataset. When training a model with such a small dataset, we run the risk of our model overfitting. Overfitting is when our model learns to perform very well on the training data but fails to generalize well to new, unseen data. By applying this cross-validation, we reduce (but not completely mitigate) this risk of overfitting, and are more confidently able to say whether our model is capable of providing reliable predictions on unseen data [77].

5.3 Data Preprocessing

In the Data Management Plan (DMP) it is explained that the dataset is contained in 5 JSON files that correspond to the iteration of 5-fold cross-validation. From these files, the data must be preprocessed before they can be input into our model. The data must be preprocessed so that it can be input into the RoBERTa architecture. RoBERTa uses Byte-Pair-Encoding (BPE) [34]. BPE is a straightforward data compression algorithm that repeatedly substitutes a single, unused byte for the most common pair of bytes in a sequence [78]. In the context of RoBERTa the algorithm is used for word segmentation and is a hybrid between character- and word-level representations. Instead of using the full words, BPE works with subword units that are extracted using statistical analysis on the training corpus. Here are the steps taken to preprocess our data and applying BPE:

1. **Normalization:** The pretrained RoBERTa tokenizer from hugginface normalizes the text that we input. The normalization process entails some basic clean-up, including removing unnecessary whitespace, lowercasing, and/or accents. For example, the string `"Héllò hôw are yoü?"` would be normalized to `"hello how are you?"`.
2. **Pre-tokenization:** This step involves splitting the full text into small entities, like words. The RoBERTa tokenizer pre-tokenizes the text that is already normalized. For instance, the string `"Hello, how are  you?"` would pre-tokenized to `[('Hello', (0, 5)), (',', (5, 6)), ('Ġhow', (6, 10)), ('Ġare' (10, 14)), ('Ġ', (14, 15)), ('Ġyou', (15, 19)), ('?', (19, 20))]` with the RoBERTa tokenizer. The `"Ġ"` value indicates that there is a whitespace

at this position. Note that the double space is not ignored like in most other tokenizers.

3. **Splitting into characters:** This step is quite simple, we extract the characters from the words obtained from the pre-tokenization step. This would mean we have `["a", "e", "h", "l", "o", "u", "y"]`
4. **Applying merge previously learned merge rules:** The RoBERTa tokenizer is pretrained with many merge rules for very large amounts of data, therefore there are many rules available to the tokenizer. We apply these merge rules to the words from the pre-tokenization step. For example, assume we have the rule `("o", "u") -> "ou"`, then the word `"you"` would be split into the tokens `["y","ou"]`.

To understand where the merge rules come from, it is beneficial to understand how a tokenizer is trained using BPE. Consider the example of a corpus that contains these 5 words: `"hug"`, `"pug"`, `"pun"`, `"bun"`, `"hugs"`. The base vocabulary then becomes `["b", "g", "h", "n", "p", "s", "u"]`. Now, during each training step, the BPE algorithm searches for the most frequent pair of existing tokens ("pair" means two consecutive tokens in a word). Then the most frequent pair is then merged. For instance, let's assume the example corpus has this frequency: `("hug", 10)`, `("pug", 5)`, `("pun", 12)`, `("bun", 4)`, `("hugs", 5)`, where the number corresponds to the frequency of the word. The words are then split: `("h" "u" "g", 10)`, `("p" "u" "g", 5)`, `("p" "u" "n", 12)`, `("b" "u" "n", 4)`, `("h" "u" "g" "s", 5)`. The algorithm looks at pairs, so the pair `("h", "u")` is present in the words `"hug"` and `"hugs"`, so 15 times total in the corpus. However, the pair `("u", "g")` is present 20 times in the vocabulary. Meaning the merge rule `("u", "g") -> "ug"` is learned by the tokenizer. Then `"ug"` will be added to the vocabulary, and the pair should be merged in all words of the corpus:

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4),
("h" "ug" "s", 5)
```

There are now several combinations that produce tokens longer than two characters, such as the pair `("h", "ug")` (15 occurrences in the corpus). The second merging rule discovered is `("u", "n") -> "un"` despite the fact that the most frequent pair at this point is `("u", "n")`, which is present 16 times in the corpus. By including that in the vocabulary and combining all instances, our corpus, and vocabulary are updated once again:

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4),
("h" "ug" "s", 5)
```

This process is repeated till all possible rules are identified. The RoBERTa tokenizer is trained to contain a much larger set of rules and vocabulary, therefore for our preprocessing, we use the pretrained tokenizer from Hugging Face. When words from input text are not in the vocabulary of the RoBERTa tokenizer, then they are broken down into subword pieces that are in the vocabulary.

5.4 Training

As discussed in Section 5.2, for each cross-validation iteration 80% of the SCD is used for training. The standard RoBERTa model is constructed using PyTorch and Hugging Face, after which we add the additional linear classifier layers are added and the NSP weights are imported into our model. We train 5 versions of the same model for each training split. The linear classifier takes an input embedding of 768 (same as the output for RoBERTa) and outputs a 17-dimensional embedding. This is because the SCD has 8 entities, and following BIO tagging there is a "B" type and "I" type for each entity, meaning technically 16 entities. Finally, we also have to consider the prediction of outside text "O" from the BIO tagging scheme, making it 17. At this stage, the model is ready to be fine-tuned to our specific dataset.

5.4.1 Hyper-parameters

For training, during each iteration, we set a fixed set of hyperparameters. The hyperparameters are selected based on the research provided by a comprehensive study on few-shot NER [35] and the software requirement NER paper [75] that compares BERT transformer variations. Most importantly, we also use the training validation results described in Section 5.5 to observe and select the hyperparameters that ensure our models are learning properly and show no indications of overfitting. We train the model by setting the batch size to 8. The selected learning rate is $1e^{-4}$, the max sequence length is 128 and for each iteration we train the model for 5 epochs.

Additionally, we use the AdamW optimizer [79] with a linearly decaying schedule with warmup at 0.1. The optimizer is an algorithm that is used during training to adjust the parameters of the model (update the weights) and guide the optimization process [80]. Meaning, at each epoch our optimizer modifies the model's weights and minimizes the loss function, and consequently improves the model's accuracy. We use the cross-entropy function, introduced in Section 2.4.4, in our training to monitor the training and validation loss. Additionally, during the training process, the AdamW optimizer also adjusts the learning rate and uses a regularization technique to prevent overfitting by penalizing large weights. The learning rate scheduler's purpose is to also adjust the learning rate during training to ensure stable initial updates and gradually increase the learning rate to reach the level where training loss converges (our optimal point in training).

5.5 Validation

The purpose of validating our model is to assess if the model is fulfilling its intended purpose/function, which is to predict Software Context entities in text. What this means is that validating our model requires us to check if the model is learning to predict the Software Context entities (displayed in Figure 4.1) during and after training. To validate the model, we will measure performance metrics at the end of each training epoch on both the training data and validation data. Calculating the performance metrics at the end of the 5th epoch is an indication of the results after the model is fully trained. To measure if the model is learning properly on the dataset, we will measure the cross-entropy loss (Equation 2.5) at the end of each training epoch. Additionally, to measure the performance of the model, we will measure the F1 score as well.

The F1 score is a very common metric that is used in NER [11, 35, 75, 81] to measure the performance of the model. It provides a balanced measure between the model's precision and recall. Precision is the ratio of true positive predictions to the total predicted positives (true positives+false positives). Meaning that it measures how many of the positive predictions that are made by the model are actually correct. Here is the formula for precision:

$$Precision = \frac{True\_Positives}{True\_Positives + False\_Positives} \tag{5.1}$$

Recall is the ratio of true positive predictions to the total actual positives (true positives + false negatives). It measures how many of the actual positive values are correctly predicted by the model.

$$Recall = \frac{True\_Positives}{True\_Positives + False\_Negatives} \tag{5.2}$$

The F1 score is the harmonic mean of precision and recall, and is calculated as follows:

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \tag{5.3}$$

The F1 score is an ideal metric to see how our model performs because there are limitations to how we can use precision and reveal it individually. This is because without looking at precision, it can go unnoticed that our model creates a lot of false positives and without recall, we can miss the false negatives. Here is how we define our predictions when calculating our F1 scores on the training and validation data:

- **True Positive:** This refers to a case where the NER model correctly identifies and labels an entity in the text.
- **True Negative:** This refers to when the NER model correctly identifies a portion of the text as non-entity.
- **False Positive:** This refers to when the NER model incorrectly identifies a portion of the text as an entity when it should be a non-entity. In other words, it labels something as an entity when it's not. Additionally, if the model labels an entity with the wrong entity, we recognize it as a false positive as well.

- **False Negative:** This refers to when the NER model fails to identify and label an entity that should have been recognized and labelled. It occurs when the model misses or overlooks an entity.

Note that when identifying true positives for the model validation metrics, we use exact matches of text. For example, let us take the labelled text "[Google maps] is the latest technology of the year.", where [Google maps] is a labelled entity. If our NER model predicts that [Google] is the labelled entity, then we consider that to be a false positive because the prediction is not an exact match. This rule only applies during model validation, model evaluation is done differently as explained in Section 5.6.

Accuracy is another metric that is used for many ML tasks, however, we intentionally avoided using this metric for NER validation and evaluation. The formula for accuracy is:

$$\frac{True\_Positives + True\_Negatives}{True\_Positives + False\_Positives + True\_Negatives + False\_Negatives}$$

Using accuracy can give the very wrong impression about the predictive capabilities of a trained NER model. This is because it considers true negatives, which means a model is good a predicting if a model is a non-entity. But the issue is that most texts include numerous non-entities and a lot fewer entity types. Therefore, a model might be very good a predicting non-entity types as most text contains much more non-entity types, and be very bad a predicting entity types. But in this scenario, the overall value of accuracy will be very high and give the impression that the model performs very well even if it does poorly to correctly label text with entities. Hence, we use the F1 score as a measure of our model's performance, as it provides a much better overview of a model's predictive capabilities.

5.6 Evaluation

The purpose of evaluating our model is to assess the overall quality of the model and how it performs for its intended task. The goal is to see whether it performs well in the real-life context that the model is constructed for. When evaluating the model in machine learning, it is absolutely crucial that the performance of the model is only measured using data that the model has never seen. Using training data is unacceptable because it easily generates overoptimistic predictions that are a poor indication of real-world application. Therefore, in our 5 iterations of training, each version of our trained model is evaluated using the evaluation data of their respective fold.

The metric that we will use to calculate the performance of the models is the F1 score. The reason that we use the F1 score to measure the performance is explained in Section 5.5. The F1 score provides the harmonic mean between precision and recall and will provide a well-rounded overview of how the model performs for its intended task. True positives, true negatives, false positives and false negatives for NER are defined in Section 5.5. There is one difference between how we calculated the F1 score for the model validation and model evaluation; the true positives are identified using overlapping matches for evaluation, not exact matches. For instance, if we look at the labelled text "[Google maps] is the latest technology of the year.", where [Google maps] is the labelled entity, "[Google]", "[Google maps]" and "[Google maps is the]" would all be considered positive matches. This is because the overlapping entities still provide valuable information and are very close to the information that we want to extract from the text. As of now, we do not set any limitations on the allowed overlap. The reason we use exact matches for validation is that we want to provide stricter requirements to see whether the model is properly learning. Checking only overlaps during this process runs the risk of getting lucky from randomness, especially in the early training epochs. However, once we validate the model and know that it is doing, what it is supposed to, we can evaluate using true positives identified using overlaps. Furthermore, in relation to the construction of the knowledge graph, the extracted entities can be cleaned up using additional processing that is discussed in Section 6.4.1.

Accuracy is a metric used in ML, however, we decide not to include it in our model evaluation. Accuracy would not be a good metric to use to evaluate a NER model's performance because most texts contain a majority of non-entity types and this can give the wrong impression with the accuracy results (as explained in Section 5.5).

Chapter 6

Knowledge Graph Construction Process

This chapter explains the steps taken to construct our knowledge graph (KG). The chapter is structured based on the KG construction methodology explained in Section 2.3. The KG construction methodology provides steps to construct a large-scale knowledge graph that can be used in a variety of tasks for its domain of interest, such as DBpedia or Wikidata [82]. However, we are using the methodology to create a tool/software that will construct a knowledge graph based on the software documentation that we input into the system, to extract the software context information. We reference the previous Chapters throughout the KG construction process, as this Chapter aims to combine everything investigated throughout our research.

6.1 Identify data

The domain of interest for our research is software context for the evaluation of software systems. Our domain of interest is the same as our Ontology's, which is discussed in Section 3.2. The sources of data for the KGs constructed by our tool will be the software documentation. This documentation is publicly available online, in particular, it would be appropriate to look up documentation for the systems of Software as a Service (SaaS) companies as it is most related to the use case of our tool.

6.2 Construct the KG Ontology

By constructing a high-level ontology, we obtain a formal definition of software context. Additionally, it provides us with a clear outline as to what type of information and knowledge needs to be extracted in order to construct a KG in our domain. Chapter 3, provides an extensive explanation of how our Software Context Ontology (SCO) is constructed. Once the ontology is constructed, we have a proper overview as to what entities and knowledge needed to be extracted to construct our KGs.

6.3 Extracting Knowledge

For our tool, we only focus on extracting entities and relations. The extraction of attributes will be left for future work.

6.3.1 Extracting Entities

Since we did not have sufficient resources to extract entities for all the concepts defined in our Software Context Ontology, we chose a select number of entities based on the guidance of our industry experts. These entities are displayed in Table 4.1, and the explanation as to why they are selected can be found in Section 4.1. In Chapter 4, we explain the whole process of constructing a dataset that contains labelled data on these entities. Our small Software Context Dataset (SCD) is then used to train, validate and evaluate a few-shot Named Entity Recognition (NER) model. This is thoroughly explained in Chapter 5. Additionally, in Chapter 5 we also explain the tools, the full process of constructing the model and the steps required to preprocess the dataset. Once the experimental setup is completed, the model became ready for entity extraction.

6.3.2 Relation Matching

There are multiple approaches to extracting relations in order to construct a KG as discussed in Section 2.3. However, many of these techniques require a large amount of resources, additional time and sometimes labelled data (for example, the Natural Language Processing based approaches). Due to these limitations, we implement a rule-based relating matching approach that is more suitable given our available resources. Our implementation of relation matching is different from relation extraction because relation extraction involves extracting the relation directly from text, while our approach maps predefined relations based on a set of rules. We discuss relation extraction techniques in areas of related work (9.3.1).

The process of relation matching is done with semi-structured data, where the text has only labelled entities. The labels are based on the predictions that are output by our fine-tuned few-shot NER model. For relation matching, we define a rule-based approach that is designed with our specific software context entities in mind. We design a set of rules on how to define the relationship between the labelled entities within our semi-structured data. These rules are designed by studying the behaviour/patterns of the entities from the training data in the SCD and studying the relations between the concepts in the SCO. Each labelled entity has a relation with another labelled entity. These relation labels are defined in Figure 6.1. The relation labels are assigned based on these rules:

1. Every `Company_Name` entity is linked to the `Company` node in the KG. Similarly, every `Software_Name` entity is linked to the `Software` node in the KG.
2. A `Software_Name` entity is only connected to a `Company_Name` if it is present in the text. Step 4 explains how the `Company_Name` is selected when multiple `Company_Name` entities are in the document.
3. Every business process entity (`Internal_Organization`, `Software_Purpose`, and `Userbase_Information`) is connected to a `Software_Name` or `Company_Name` entity, and every scalability entity (`Transaction_Scalability`, `Data_Scalability`, and `Development_Scalability`) is connected to a `Software_Name` entity. Step 4 explains how the entities are selected when multiple appears in a document.
4. In many cases, there will be multiple entities of `Company_Name` or `Software_Name` in the document that is input into the system. Therefore, we follow these rules to define the triples. Triples are defined as $(h, r, t) \varepsilon F$, where h is the starting node/entity, r is the relationship and t is the ending node/entity (as explained in Section 2.1). We will refer to the entity that connects to the `Company_Name` or `Software_Name` entity as the secondary entity:
   (a) If there are multiple `Company_Name` or `Software_Name` entities in a document, then only one of them needs to be connected to the other entities. If the `Company_Name` or `Software_Name` is present in the same sentence as the secondary entity, we make a connection to the secondary entity. For example, if "Amazon" and "Google" are mentioned in the document multiple times, and we have a sentence "Amazon is great. But Google works with many engineers", where "engineers" is an `Internal_Organization` entity, we connect "Google" to "engineers" as they are in the same sentence.
   (b) When these multiple `Company_Name` or `Software_Name`entities are in the same sentence, then the secondary entity is connected to the entity that is closer to the secondary entity based on the index position within the string. Using a similar example but in one sentence we have "Amazon is great, but Google works with many engineers", where the index of "Amazon" = 0, "Google" = 19, and "engineers" = 42. If we calculate the absolute distance: $|42\text{-}19| = 23 < |42\text{-}0| = 42$. Therefore, "Google" would be connected to "engineers".
   (c) When there are no `Company_Name` or `Software_Name` entities in a sentence, we check the frequency of each `Company_Name` or `Software_Name` mentioned throughout the entire document and select the entity with the highest frequency. For instance, throughout the entire document if "Amazon" is mentioned 5 times and "Google" is mentioned 3 times, "Amazon" is connected to the secondary entity. The frequency table is constructed before the relation matching process, by just going through the document beforehand and creating a frequency table of all the `Company_Name` and `Software_Name` entities. If there are multiple entities with

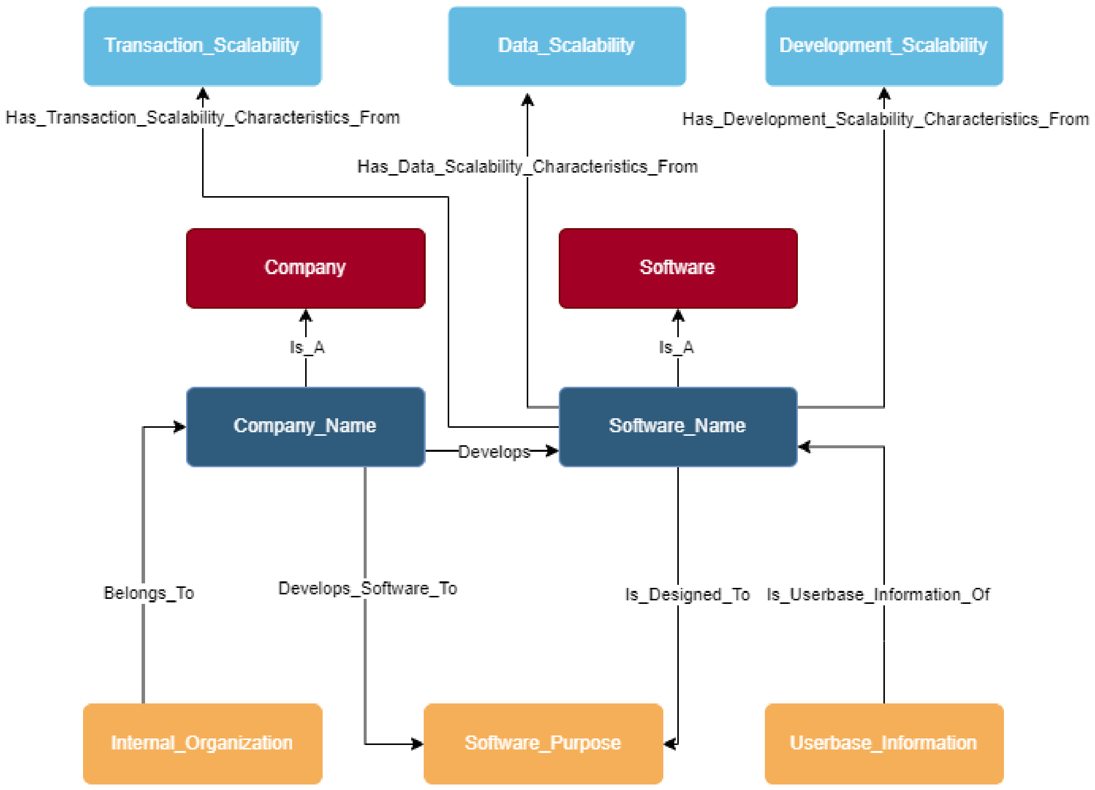

Figure 6.1: Relation labels between entities.

the highest frequency (i.e. the highest frequency is 5 and multiple entities come up 5 times in the document), follow the next step.

(d) When the frequencies are all the same, we use the last `Company_Name` or `Software_Name` entity mentioned in the document. This means that when going through each sentence/sample in the document, we record the last mention of a `Company_Name` and `Software_Name` entity. We use this recorded entity and connect it to the secondary entity when the frequencies are all the same. We make sure to update these values at every new mention of a `Company_Name` or `Software_Name` entity.

6.4 Process Knowledge

We apply additional processing to the extracted entities. The relations are defined based on a set of rules with set labels, and therefore we do not apply any additional processing to them.

6.4.1 Extracted Entity Processing

NLP models are never 100% accurate when performing language processing tasks, therefore our NER model automatically has a risk of producing incorrect predictions. Additionally, we use the overlap method when checking for positive predictions from the model as explained in Section 5.6, which means there is a potential for additional text outside the entity to also be extracted. Therefore, we apply additional processing steps to the entities in an attempt to make our tool produce only semantically correct knowledge graphs. We apply these two steps to process the extracted entities:

1. For each entity, we look at the full sentence and remove certain parts-of-speech (POS) that are irrelevant when understanding the entity.
2. We lemmatize the tokens of the entity.

In the first step, we use the POS tagger from the spaCy library [1], to figure out what POS an extracted entity contains. From observing the training data, we found that only certain entities should only contain

[1] https://spacy.io/usage/linguistic-features#pos-tagging

certain POS. Through studying this behaviour and observing what POS should be removed for certain entities, we came to the conclusion that for the entities `'Company_Name'`, `'Internal_Organization'`, and `'Software_Name'` we should remove all other POS that are not contained within this list: `["NOUN", "PROPN", "ADJ", "NUM", "SYM", "X", "ADP"]`, which correspond to the Universal POS tags [2]. Furthermore, for the remaining entity types, we remove all other POS not contained within this list: `["NOUN", "PROPN", "ADJ", "NUM", "SYM", "X", "ADP", "VERB"]`. For the second step we use lemmatization to reduce the words of each extracted entity to their root word. This is to provide correct semantic meaning when producing the nodes for our knowledge graph. For example, if we have a "Discord" `Software_Name` entity connected to a "communicating" `Software_Purpose` with an "Is_Designed_To" relation the triple does not semantically make sense: ("Discord", "Is_Designed_To", "communicating"). Therefore, we use lemmatization to process our entities and produce this triple: ("Discord", "Is_Designed_To", "communicate"), which is semantically correct. We use the lemmatization tool provided by the library spaCy [3].

6.5 Construct the Knowledge Graph

The code for our tool is available in the project repository. We visualize our KGs using the Networkx library [4]. Additionally, when a document is input into our tool, the nodes and the triples of the produced KG are stored in a JSON file. This JSON file can be imported to visualize the KG using the Networkx library and to save the image of the visualization.

6.6 Maintain the Knowledge Graph

As part of this step of our KG construction process, we will specifically focus on the validation and evaluation of the knowledge graphs to validate and evaluate our proposed methodology for software context extraction from system documentation.

6.6.1 Validate the Knowledge Graph

To validate our approach, we provide a qualitative assessment. The qualitative assessment is to check if our tool accurately achieves our goal of extracting software context from system documentation. We conducted validation sessions with 5 industry experts (all technical consultants that have experience in extracting software context from system documentation; all their projects require this). The validation sessions are structured as follows:

1. Provide the consultant with an example of system documentation. Ask them to familiarize themselves with the documentation and apply the software context extraction techniques they use. That way, they know what information needs to be extracted from the documentation.
2. Present them with the KG constructed by our tool. Ask them to study the KG.
3. Ask them to explain if they see the correct/expected software context information on the KG after analysing the original system documentation. Ask them if the software context information is represented accurately and whether there is missing information.

6.6.2 Evaluate the Knowledge Graph

To evaluate our approach, we will also use qualitative assessment and calculate a quality metric. We conduct the evaluation session with the same 5 industry experts from the validation session. Huaman [83] proposes a general-purpose quality assessment framework for assessing the quality of KGs, and it is customizable to specific domains or tasks. As part of the evaluation of our approach, we follow this quality assessment framework and adapt it to our specific task. The framework is based on a literature review of the current state-of-the-art KG quality assessment. It considers various quality dimensions (QDs) and quality metrics (QMs) that are specific to KGs. The framework follows a Goal Question Metric (GQM) approach [83], where we define (i) a goal, (ii) a set of questions to achieve the goals (iii) and a set of metrics to answer the questions. Note that the metrics for the majority of goals have to be based on qualitative assessment (only 2 questions are exceptions), as calculating the quantitative metrics

[2]https://universaldependencies.org/u/pos/
[3]https://spacy.io/usage/linguistic-features#lemmatization
[4]https://networkx.org/

Table 6.1: Goals and questions based on the studied KG quality assessment frameworks [83, 84].

Goal	Questions
Accuracy	QM1: Is the KG syntactically reliable/correct? Are the triples defined and structured? Are the triple concise/ non-redundant? QM2: Is the KG semantically reliable/correct? Do you think the information from the original document is correctly represented in the knowledge graph?
Appropriate amount	QM3: Does the KG contain an appropriate amount of information extracted from the documentation? Do you get a good picture of what the documentation is about?
Ease of understanding	QM4: Is the KG represented using descriptive entities and relations? Are they clear?
Interoperability	QM5: Does the KG use standard vocabulary for the domain?
Relevancy	QM6: Is the KG relevant for the task of presenting/visualizing extracted information from software document? QM7: How well does the KG perform for its use case?
Believability	QM8: Does the information in the KG seem trustworthy? QM9: Is there any missing relationships or other information?
Completeness	QM10: Entity Coverage: Are all relevant entities from the target domain represented in the knowledge graph? Are there any missing entities that should be included? QM11: Relationship Coverage: Does the graph capture all the significant relationships between entities? Are there any missing connections or edges that should be present?

requires additional data and our experts did not have time to participate in longer sessions to collect this data. Table 6.1 displays all the goals and questions for the KG assessment, customized for our particular task. The goals and questions are all selected from the paper by Huaman [83], and the questions are slightly modified to include some KG requirements discussed in the framework for evaluating the quality of KGs proposed by Chen et al. [84]. Syntactic reliability in the goal "accuracy" will be measured by calculating the percentage of syntactically incorrect triples. Entity coverage in the goal "completeness" will be calculated using the F1 score for the document. For the rest of the questions, we asked the evaluator to provide a score from 0-100 based on the score criteria in Table C.1, which is present in Appendix C.

The evaluators are asked to provide a score for each of the questions in our customized quality assessment framework. For each question, we obtain a QM. The QM scores recorded during the evaluation sessions are divided by 100 to normalize the scores. Note that for our quality assessment, for each goal, we give equal importance to each QM. Mathematically, we define the QM as $m_{i,j}$, where we have the j^{th} QM for the i^{th} QD. The QD is the same as the goal in Table 6.1, meaning we have 7 QDs. We calculate the weight of the QM as follows:

$$\alpha_i = \frac{1}{c_i}, \text{where } i = 1, ..., n \tag{6.1}$$

where c_i corresponds to the total number of quality metrics for the i^{th} QD, n is the total number of QDs, and α_i is the weight for every QM for the i^{th} QD.

Near the end of the evaluation session, once all the questions are asked and scores are collected, the evaluators are asked to rank the goals/QDs in Table 6.1. We asked the evaluators to rank the 7 QDs from a rank between 1-7 from most important to least important. It is made clear to them that they need to make their ranking based on the context of the task/use case; software context extraction from system documentation. The rankings are collected because we need to calculate an aggregate score for each QD, but each QD will have a different weight of importance, which impacts the overall quality assessment score. Based on the rankings, we decide that the Rank Order Centroid (ROC) formula would be a good formula to use when deciding what weights to assign to each QD. We chose this formula because it provides a good indication of relative importance based on ranking and provides an objective and consistent approach to calculating the QS weights. Therefore, we adapt the ROC formula to Huanman's original formula [83] to calculate the weight distribution of each QD. The weight for each

QD is represented by β_i for the i^{th} QD:

$$\beta_i = \frac{\frac{1}{n}\sum_{s=r_i}^{n}(\frac{1}{s})}{\sum_{t=1}^{n}(\frac{1}{n}\sum_{s=r_t}^{n}(\frac{1}{s}))}, \text{where } i = 1, ..., n \text{ and } \sum_{i=1}^{n}\beta_i = 1 \tag{6.2}$$

where r_i is the ranking for the i^{th} QD, that is provided by the evaluator. Note that the denominator for 6.2 will equate to 1 if the evaluator uses all numbers between 1-7 for their rankings. Some evaluators provide the same rankings for multiple QDs, therefore the denominator is required to normalize the ranking weights.

The calculation for the aggregate score for each QD is represented by $d_i(g)$ for the KG g:

$$d_i(g) = \sum_{j=1}^{k_i} m_{i,j} \cdot \alpha_i \tag{6.3}$$

where k_i is the number of metrics for the i^{th} QD, $m_{i,j}$ is the score of the j^{th} QM for the i^{th} QD, α_i defines the impact of each QM score on the i^{th} QD score, and $d_i(g),\ m_{i,j},\ \alpha_i\ \varepsilon\ [0,1]$. Finally, the full formula for our KG quality assessment is represented by $T(g)$:

$$T(g) = \sum_{i=1}^{n} d_i(g) \cdot \beta_i \tag{6.4}$$

where n is the number of QDs, $d_i(g)$ is the score of the *ith* QD of g, β_i represents the weight of the *ith* QD, and $T(g),\ d_i(g),\ \beta_i\ \varepsilon\ [0,1]$.

Once we have asked the industry experts all the questions related to the KG quality assessment framework, we will ask them one last question about the KG. As part of the evaluation, we would like the industry experts to explain how using this knowledge graph compares to the current method they use to extract software context from the documentation. The purpose of this question is to evaluate our approach and its performance compared to the current standard they use. Note that this will only provide additional qualitative data.

Chapter 7

Results

In Chapter 3 we introduce the methodology to construct our Ontology which is a formal definition for software context. Next, in Chapter 5 we introduce the few-shot Entity recognition model that we train using the dataset built in Chapter 4. Finally, in Chapter 6 we explain the process used by our tool to construct software context Knowledge Graphs (KG) from system documentation.

In this chapter, we collectively present the results from all the components of our research (from the above-mentioned chapters) to provide a comprehensive understanding of the results of the full research process. This is important because the results of each part of our research directly influence the next part. The results of our ontology impact whether we have high-quality software context entities, which are the basis of our Software Context Dataset (SCD). The choice of entities for the SCD directly impacts the quality of the software context information that we train our NER model to extract. Lastly, the performance of the NER model and the quality of the entity extracted (whether the entity type is information that is useful to an expert in terms of software context or not) directly correlate to the quality of the KGs we construct. Hence, we present the results of all components in this chapter to highlight the interdependencies between our research components.

7.1 Software Context Ontology (SCO)

We construct our Software Context Ontology (SCO) to provide a formal definition of software context for evaluating software systems. Based on the methodology introduced in Chapter3, we construct two variations of our SCO, our Industry Ontology (IO) that contain the knowledge obtained only through industry experts, and our Supplementary Research Ontology (SRO) which uses the IO as the foundation and adds supplementary research from existing reusable ontologies.

In Section 3.7, we discuss the techniques used to evaluate our IO and SRO. We first check the ontologies for consistency and correct any inconsistencies/ errors in the ontologies. Once completed, we use the OOPS! scanner[71] to detect pitfalls in our ontologies. Table 7.1 displays the evaluation results output by the scanner for our IO and SRO. The table shows the pitfalls detected for our ontologies along with the number of cases for each corresponding pitfall. Furthermore, to get a clear understanding of what category each pitfall fell under, we reference the dimension of each pitfall as described in the paper by Poveda-Villalón et al. [71]. Finally, we address the severity of each pitfall, where it can fall under 3 categories [71]:

1. **Minor**: It does not pose a problem. However, fixing it improves the ontology's organization and user-friendliness.
2. **Important:** Although not critical for the ontology to function, it is crucial to avoid this kind of pitfall.
3. **Critical:** It is essential to avoid these pitfalls. If not, it could have an impact on the ontology's logic, application, and consistency, among other things.

The results indicate that there are numerous cases of minor errors, where the majority are due to missing annotations and inferences. We leave the corrections of these pitfalls for future work. This is because the tradeoff is to spend additional time for significantly less research value because many of the classes to the average domain user are self-explanatory directly from the labels, and do not necessarily require an annotation. By research value, we mean new knowledge that is being obtained through our research; correcting the pitfalls would have taken away time to explore our other research questions. In terms of important pitfalls, there are 7 cases of equivalent classes not being explicitly declared for the SRO and 1 case for the IO. The labels of these classes are modified for clarification, as these classes are not in fact equivalent. There are also indications of 1 critical pitfall of defining wrong symmetric relationships in both the IO and SRO. This is corrected for both ontologies.

Table 7.1: Detected pitfalls of our IO and SRO when input into the OOPS! scanner [71].

Pitfall	**Dimension**	**Pitfall severity**	**Industry Ontology (IO) cases**	**Supplementary Research Ontology (SRO) cases**
Creating unconnected ontology elements.	Requirement completeness	Minor	1	1
Missing annotations.	Ontology understanding, Ontology Clarity	Minor	235	433
Inverse relationships not explicitly declared.	No inference, Ontology understanding	Minor	36	62
Defining multiple domains or ranges in properties.	Wrong inference	Critical	7	10
Using a miscellaneous class.	Modelling decisions	Minor	1	1
Using different naming conventions in the ontology.	Ontology clarity	Minor	Ontology*	Ontology*
Defining wrong symmetric relationships.	Wrong inference	Critical	1	1
Equivalent classes not explicitly declared.	No inference	Important	1	7

Figures 7.1, 7.2 and 7.3 display parts of the final versions of the IO and SRO, after the discussed pitfall corrections are made. In Figure 7.1, only the first 3 levels of IO from the top of the hierarchy (`Software_Context`) are displayed (the first 3 levels meaning out ontology up to a depth of 3). Figure 3.6 displays the same but for the SRO. (The full ontology is available in our project repository [1]) Figure 7.2 is the same as Figure 7.1 but contains additional classes and relationships. For instance, we can see the SRO contains three additional subclasses to `Software_Quality_Factors`, which are `Compatibility`, `Portability` and `Usability`. In Figure 7.3, we see the additional classes and relationships obtained from supplementary research. We extract these additional classes and relationships from reusable ontologies in the domain of maintainability, reliability, security, and performance efficiency. To provide a statistical difference between the two constructed ontologies, we inspect Table 7.2. The values show that there is an 86% increase ($\frac{369-198}{198} \cdot 100$) in the number of classes once we add reusable ontologies to our IO. Additionally, there is a 53% ($\frac{83-54}{54} \cdot 100$) and 338% ($\frac{70-16}{16} \cdot 100$) increase in the object property count and individual count, respectively. Note that both ontologies do not contain any data properties. This is a conscious decision, as data properties are used to describe more detailed characteristics or features of individuals (in the form of a data type). The purpose of our ontology is to provide a high-level ontology that provides an overview of the domain. Therefore, we intentionally did not include data properties, as they are low-level information that could potentially be too specific and inapplicable to all contexts of the ontology domain.

It is clear that there is a clear difference between the IO and SRO when looking at the statistics in Table 7.2. However, this value is only an indication of the change in the size of the ontology, it does not provide insight into whether the SRO provides higher-quality knowledge to the domain of software context for evaluating software compared to the IO. Therefore, we use the FOCA methodology discussed in Section 3.7.3 to calculate a quality score for both ontologies. Table 7.3 indicates all the grades for all the FOCA evaluation questions and the mean for each goal. Based on these results, we calculate the evaluation score for the IO:

[1] https://github.com/athuln-99

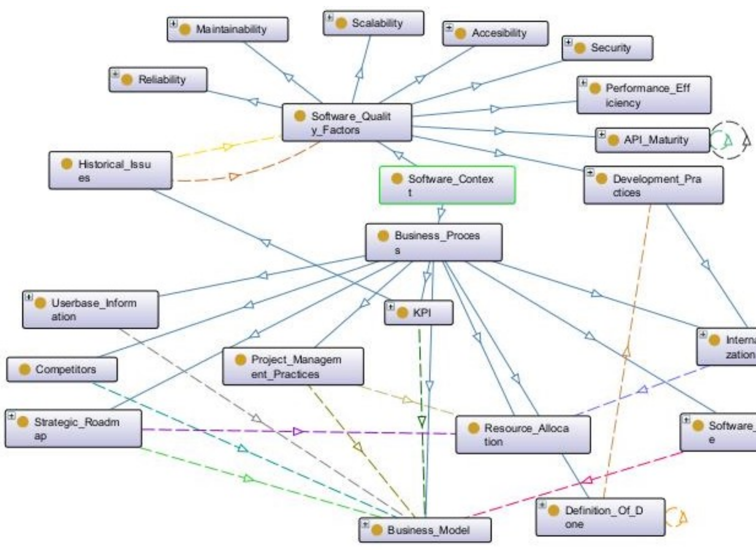

Figure 7.1: Final version of the Industry Ontology (IO). Only indicates the concepts and relationships up to a depth of 3.

Table 7.2: This table provides an overview of some metrics calculated for the IO and SRO.

Metrics	**Industry Ontology (IO)**	**Supplementary Research Ontology (SRO)**
Axiom count	601	1,158
Class count	198	369
Object property count	54	83
Data property count	0	0
Individual count	16	70

$$\widehat{\mu}_i = \frac{\begin{array}{l}\exp\{-0.44 + 0.03\,(58.33 \times 1) + 0.02\,(87.5 \times 1) + 0.01\,(62.5 \times 1) \\ +0.02\,(100 \times 1) - 0.66 \times 0 - 25(0.1 \times 1)\}\end{array}}{\begin{array}{l}1 + \exp\{-0.44 + 0.03\,(58.33 \times 1) + 0.02\,(87.5 \times 1) + 0.01\,(62.5 \times 1) \\ +0.02\,(100 \times 1) - 0.66 \times 0 - 25(0.1 \times 1)\}\end{array}} = \mathbf{0.960} \tag{7.1}$$

Additionally, we calculate the evaluation scores for the SRO:

$$\widehat{\mu}_i = \frac{\begin{array}{l}\exp\{-0.44 + 0.03\,(58.33 \times 1) + 0.02\,(87.5 \times 1) + 0.01\,(62.5 \times 1) \\ +0.02\,(100 \times 1) - 0.66 \times 0 - 25(0.1 \times 1)\}\end{array}}{\begin{array}{l}1 + \exp\{-0.44 + 0.03\,(91.67 \times 1) + 0.02\,(87.5 \times 1) + 0.01\,(62.5 \times 1) \\ +0.02\,(100 \times 1) - 0.66 \times 0 - 25(0.1 \times 1)\}\end{array}} = \mathbf{0.985} \tag{7.2}$$

The results indicate that the quality of the SRO is only 2.5% better than the IO. These results indicate that there is a very insignificant difference in quality for both ontologies.

7.2 Software Context Named Entity Recognition (SCNER)

In our experiments, we construct a few-shot NER model that we then fine-tune to recognize software context-related data. Since we apply 5-fold cross-validation, we train 5 models with the same architecture in the same exact way, just that the training, validation and evaluation data are different for all versions. In this Section, we display the results during the validation and evaluation for all 5 versions of our model.

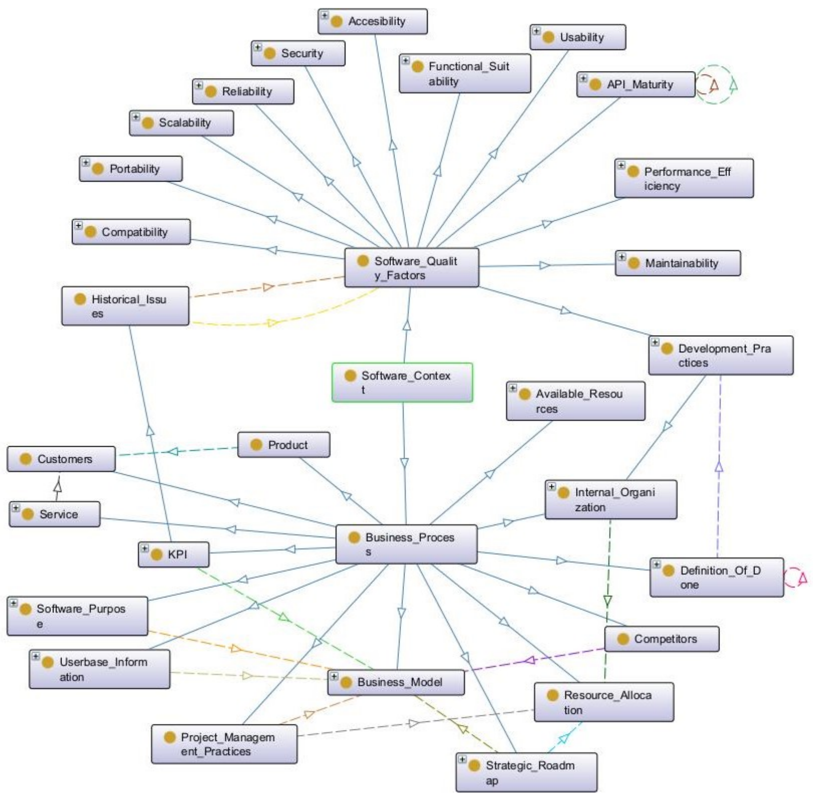

Figure 7.2: Final version of the Supplementary Research Ontology (SRO). Only indicates the concepts and relationships up to a depth of 3.

7.2.1 Validation

In Section 5.5, we detail the steps required to validate our model. Figure 7.4 illustrates the average training and validation loss derived from our 5 fine-tuned SCNER models. To elaborate, we apply 5-fold cross-validation, and we have 5 sets of training and validation data. Following each training epoch, cross-entropy losses are computed for both training and validation data across all 5 model versions. Therefore, the results shown in Figure 7.4 present the combined average training loss and average validation loss (y) across all 5 models at each epoch x, reflecting the model's progress up to epoch x. Similarly, Figure 7.5 illustrates the model's progress up to epoch x. However, instead of cross-entropy loss, this figure represents the average training F1 Score and average validation F1 score (y).

7.2.2 Evaluation

In Section 5.6, we explain our model evaluation process. Table 7.4 showcases the precision, recall and F1 scores for our models. Once the models are fully trained, we calculate these metrics for each cross-validation iteration. The results are calculated using the evaluation data from each cross-validation split. In the last row, we present an average performance across all iterations of the model. Furthermore, using the same 5 fully trained models with the same 5 sets of evaluation data, we record the performance of the models on each distinct entity type. These results are presented in Table 7.5 (The F1 score alone provides a good indication of how the model performs when predicting each entity type). The averages from all five iterations are also included to provide a comprehensive overview of the collective results.

7.3 Software Context Knowledge Graph (SCKG)

For our investigation, we follow a global view approach for KG development. The product of our investigation is a tool, that takes document text as an input, extracts the software context knowledge from the documentation and outputs the information in the form of a KG. To validate and evaluate our approach, we engage in five interactive sessions with industry experts. These experts, all seasoned technical consultants with a primary focus on software evaluation projects, brought their extensive

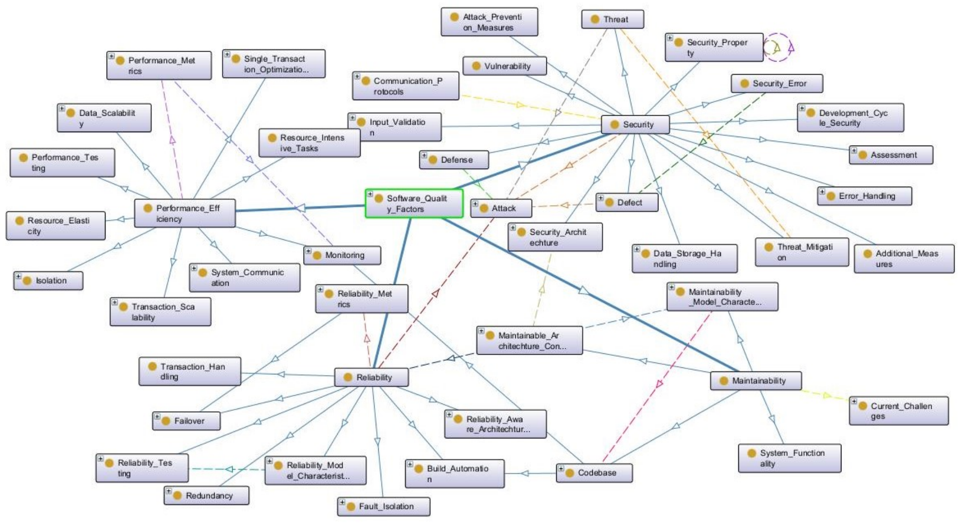

Figure 7.3: Part of the Supplementary Research Ontology (SRO)). The Figure shows the first level of Maintainability, Reliability, Security, and Performance Efficiency classes; the 4 classes for Software Quality Factors that are added onto using existing reusable ontologies.

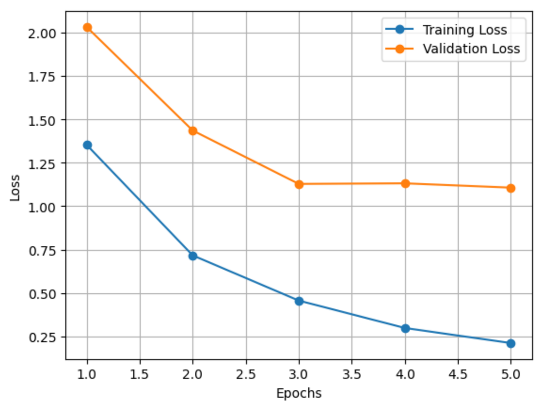

Figure 7.4: The average training and validation loss of the 5 cross-validated models after each training epoch.

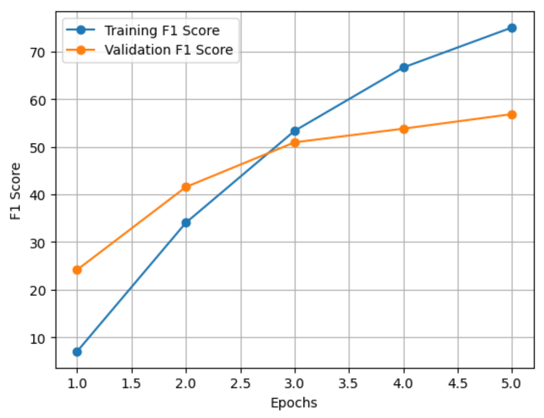

Figure 7.5: The average training and validation F1 score of the 5 cross-validated models after each training epoch.

experience to the evaluation process.

7.3.1 Validation

We collect qualitative data during our sessions to validate our approach. During the sessions, we provide the 5 experts with the original document in Section C.1.2 of the Appendix. Once they are finished analysing the original document, we provide them with the KG output by our tool. Figure 7.6 displays the KG that is provided to the experts. As the KG slowly got too large, making some relationships difficult to read, we also provide the full list of triples written out; they can be seen in Section C.1.3 of the Appendix. As part of the validation, we ask the experts whether they see the expected/ correct software context information in the KG. The following Sections provide a summary of responses from each expert:

Expert 1

- The original documentation itself is a bit confusing in certain aspects to comprehend and requires multiple read-throughs to correctly understand what the documentation is saying.
- All the relationships that follow the structure `Has_X_Scalability_Characteristics_From` are not very readable at an initial glance of the KG. However, once an explanation is provided, then

Table 7.3: This table represents the GQM criteria for the FOCA methodology and our results through the evaluation process.

GQM			Industry Ontology (IO)		Supplementary Research Ontology (SRO)	
Goal	*Question*	*Metric*	*Grade*	*Mean*	*Grade*	*Mean*
1.Check if the ontology complies with Substitute.	Q1. Were the competency questions defined?	Completeness	100	58.33	100	91.67
	Q2. Were the competency questions answered?	Completeness	75		75	
	Q3. Did the ontology reuse other ontologies?	Adaptability	0		100	
2. Check if the ontology complies Ontological Commitments.	Q4. Did the ontology impose a minimal ontological commitment?	Conciseness	-	87.5	-	87.5
	Q5. Did the ontology impose a maximum ontological commitment?	Conciseness	75		75	
	Q6. Are the ontology properties coherent with the domain?	Consistency	100		100	
3. Check if the ontology complies with Intelligent Reasoning.	Q7. Are there contradictory axioms?	Consistency	50	62.5	50	62.5
	Q8. Are there redundant axioms?	Conciseness	75		75	
4. Check if the ontology complies Efficient Computation.	Q9. Did the reasoner bring modelling errors?	Computational efficiency	100	100	100	100
	Q10. Did the reasoner perform quickly?	Computational efficiency	100		100	
5. Check if the ontology complies with Human Expression.	Q11. Is the documentation consistent with modelling?	Clarity	0	33.33	0	41.67
	Q12. Were the concepts well written?	Clarity	100		100	
	Q13. Are there annotations in the ontology that show the definitions of the concepts?	Clarity	0		25	

it becomes more readable.

- Overall, the KG does a good job of covering all the important information from the documentation. This is especially the case with the entities because the entities cover all the most important keywords, and there does not seem to be too much missing in terms of entities. In terms of the relationships, because it does not directly take from the documentation, it does miss out on valuable information.
- There is definitely incorrect information, this is from the relationship that connects entities. For example, "SIG" did not develop "Amazon Aroura".
- The `Is_Designed_To` relationship in the KG does not provide too much value, however, when combined with cross-referencing in the documentation, it can be very valuable.
- It is very obvious and visible what is incorrect, so if an expert were to use this, they would easily be able to differentiate between the correct and incorrect information.

Table 7.4: Precision, Recall and the F1 scores for all the trained models from each cross-validation iteration.

Cross-Validation Iteration	**Precision**	**Recall**	**F1 Score**
Iteration 1	0.75	0.84	0.79
Iteration 2	0.76	0.83	0.80
Iteration 3	0.68	0.85	0.76
Iteration 4	0.74	0.89	0.81
Iteration 5	0.71	0.82	0.76
Average	0.73	0.85	0.78

Table 7.5: F1 scores for all trained models for each cross-validation iteration, based on the performance of each entity type.

Entity Type	F1 Score					
	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Average
Software_Purpose	0.51	0.47	0.50	0.62	0.47	0.51
Data_Scalability	0.82	0.90	0.79	0.82	0.82	0.83
Software_Name	0.81	0.83	0.77	0.85	0.70	0.79
Transaction_Scalability	0.75	0.80	0.79	0.80	0.76	0.78
Company_Name	0.78	0.67	0.71	0.83	0.90	0.78
Internal_Organization	0.87	0.86	0.81	0.86	0.83	0.85
Userbase_Information	0.85	0.73	0.78	0.74	0.74	0.77
Development_Scalability	0.80	0.80	0.76	0.84	0.79	0.80

Expert 2

- 75% of the time the information is correct, but the 25% are still very important.
- All the relationships that follow the structure `Has_X_Scalability_Characteristics_From` are quite confusing as relationships, making the relationships between entities slightly inaccurate.
- The KG incorporates the most important information from the documentation.
- Overall, there are no major issues, but the KG is definitely missing some information. Such as an `Is_Using` relationship to indicate that one software uses another software, tool, framework, etc.

Expert 3

- Based on comparing the information from the document and what is present in the KG, some information that is expected to be seen from the document is missing.
- The relationships are too general, and they need to be more specific. For example, employees should be linked to teams, the current relationships are too broad and needs to be more specific to have value. Another example of a missing relationship is that the "sprint backlog" should link to the "product owner", who then should link to a team.
- Using KGs as a means to represent information from software documentation, is a relevant method in the first place. In theory, it could work because the information is linkable in practice. However, the KG that is presented does not do this well and can definitely not be used for a project in its current state.
- Some relationships are not correct, it is quite clear which ones are not correct.
- The KG requires more conceptualization. In an ideal world, it would require separating entities into larger entities, for example, `Internal_Organization`. Right now it is too general and this affects the overall accuracy and believability. The KG should be conceptualized better to also show transitive relationships.
- Another example of missing relationships is linking "millions of users" to "45000 customers", and then linking "customers" to the specific country and then to "Sigrid". It comes back to the fact that the relationships are too specific.
- The majority of the concepts are covered, but because of the relationships, the KG is very inaccurate. This causes more difficulty as the KG get larger.

Expert 4

- There are definitely inaccuracies. For example, in the document, it says in "Scrum", it just discusses Scrum, however in the KG it implies that "Sigrid" is built with `"Scrum"`, which cannot be inferred based on the documentation. So it is not semantically correct.
- The relation `Is_Userbase_Information` is not very clear to them.
- Entities are very clear.
- In the triple ("SIG", "develops", "Amazon aurora"), the relationship should be with "develops with".
- The KG properly identifies the entities and some relations. There are relations that should not be there and they should be improved.
- It covers mostly. "15 low latency read", is missing "15". So entities are losing some information.

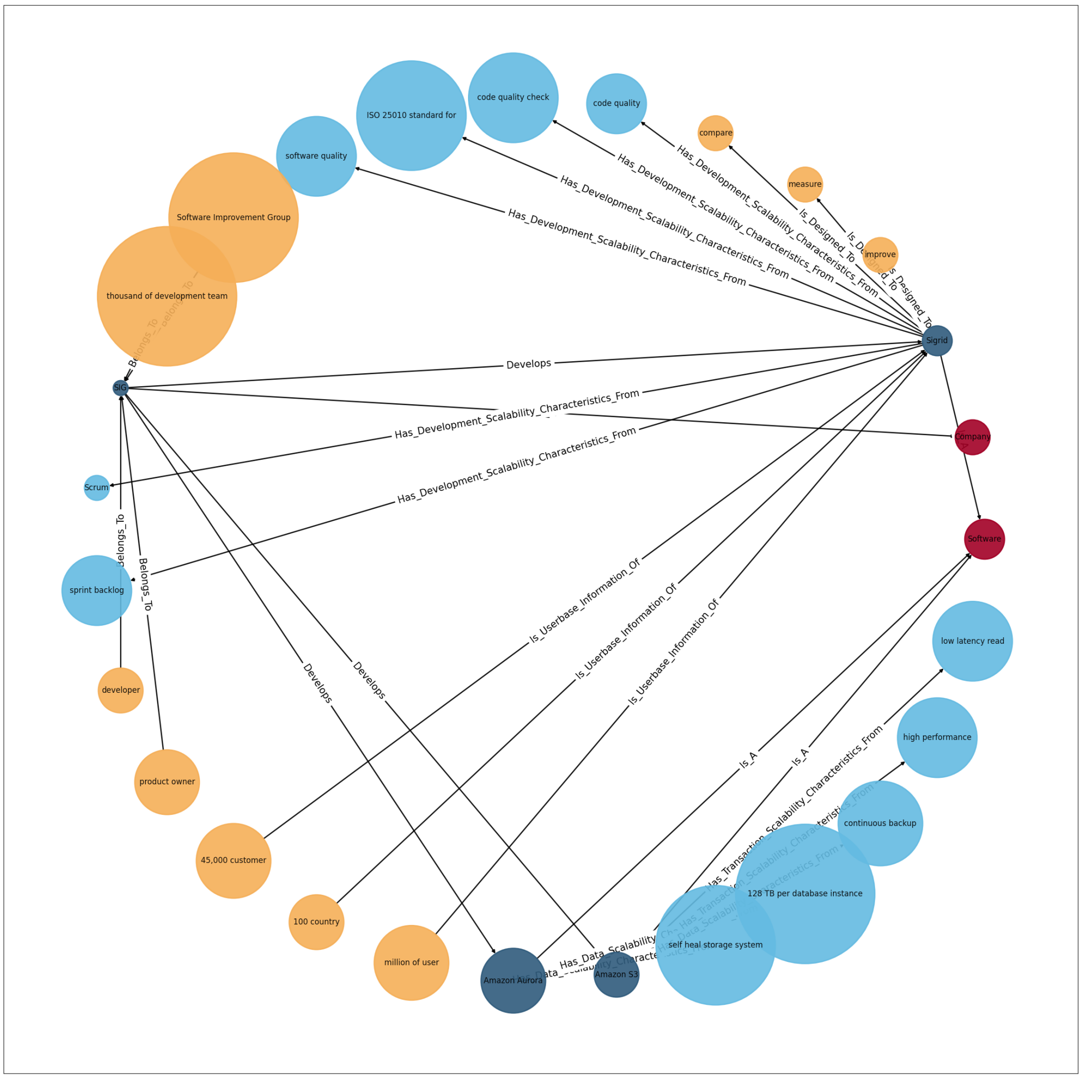

Figure 7.6: KG output by our tool, when we input the documentation text from Appendix C.1.2. The KG is getting large and becomes less readable, therefore the edges are listed in Appendix C.1.3.

- The main worry is the loss of some information. Or even misinterpretation can be an issue. For example, looking at the errors explained above.
- Information from the same sentence is not always linked. There is no relation between “Sigrid” and “Amazon”, while it uses “Amazon”. “Developer” and “product owner”, and it is not linked to “Scrum”, while in the document it is linked to the same sentence. Therefore, there is some missing relationship there.

Expert 5

- The KG provides a lot of unclarity, some things are not accurate, while some information is too generic that even though it is presented in the KG, it will not help much with software evaluation. For example, the relationship `Has_Development_Scalability_Characteristics` provides no substantial meaning, and it is unclear when it is connected to some entities such as “Scrum”. Particularly, this relationship is semantically difficult to understand when connected to entities.
- When the relationships in the sentence are simple, the KG is quite accurate. However, if the sentence becomes more complex either by becoming longer, adding punctuation, etc, the relationships become less accurate.
- Some relationships are difficult to understand, for example `Is_Userbase_Information` is unclear and too generic.

- There are certain relationships that are missing based on reading the documentation for sure, for example, “ISO 25010 standard for” and “software quality” should be linked, but for some reason, they are not.
- In some cases, the additional relationships that added extra information that is not indicated in the documentation are also incorrect. For example, “SIG” does not develop “Amazon S3”.

7.3.2 Evaluation

During the evaluation portion of the session with the experts, we use a goal, questions, and metrics approach that is explained in Section 6.6.2. The experts are asked a series of questions and also asked to provide a score. This score is a Quality Metric (QM) in our KG quality assessment. Additionally, for each Quality Dimension (QD), which is the same as the goals, we ask them to rank them based on how well the KG performs for its use case. Table 7.6 illustrates the results we obtained from the evaluation sessions with each expert. It displays the QM score provided by each expert and the corresponding rank weight for each QD, which is calculated using Equation 6.2 and the original rankings provided in Table C.2 from the Appendix. For example, to calculate the Rank Weight of Expert 1, for the QD accuracy that they gave the rank 5 to, we compute this calculation:

$$\beta_{accuracy} = \frac{\frac{1}{7}\sum_{s=5}^{7}(\frac{1}{s})}{\sum_{t=1}^{7}(\frac{1}{7}\sum_{s=r_t}^{7}(\frac{1}{s}))} = \frac{0.07}{1} = 0.07$$

It is important to note the QM1 and QM10 are the same for all experts. This is because these QMs are automatically calculated using Python scripts. For QM1 we check for the percentage of redundant triples and subtract it from 1. For QM10 we calculate the F1 score of the predicted entities within the documentation.

Table 7.6: The Quality Metric (QM) scores from each expert for each Goal/ Quality Dimension (QD).

Goal	Questions	Expert 1		Expert 2		Expert 3		Expert 4		Expert 5	
		Rank Weight	Score	Rank Weight	Score	Rank Weight	Score	Rank Weight	Score	Rank Weight	Score
Accuracy	QM1	0.07	1.00	0.37	1.00	0.11	1.00	0.20	1.00	0.24	1.00
	QM2		0.30		0.50		0.25		0.50		0.50
Appropriate amount	QM3	0.16	0.75	0.07	0.75	0.17	0.50	0.09	0.60	0.07	0.75
Ease of understanding	QM4	0.23	0.50	0.02	1.00	0.11	0.75	0.09	0.70	0.15	0.25
Interoperability	QM5	0.04	0.75	0.04	0.75	0.27	0.75	0.09	0.75	0.07	0.75
Relevancy	QM6	0.37	0.75	0.16	0.25	0.11	0.00	0.13	0.40	0.24	0.50
	QM7		0.50		0.75		0.25		0.30		0.50
Believability	QM8	0.02	0.75	0.23	0.50	0.11	0.25	0.20	0.30	0.15	0.75
	QM9		0.75		0.50		0.25		0.30		0.50
Completeness	QM10	0.11	0.86	0.11	0.86	0.11	0.86	0.20	0.86	0.10	0.86
	QM11		0.75		0.75		0.25		0.30		0.50

Using all the ranked weights and scores for each QM in Table 7.6, we use Equation 6.4, to calculate the KG quality assessment scores. The findings are presented in Table 7.7. Here is an example of how the quality assessment score is calculated for expert 1:

$$\begin{aligned} T(g) = {} & 0.07 \cdot (0.5 \cdot 1 + 0.5 \cdot 0.3) \\ & + 0.16 \cdot (1 \cdot 0.75) \\ & + 0.23 \cdot (1 \cdot 0.5) \\ & + 0.04 \cdot (1 \cdot 0.75) \\ & + 0.37 \cdot (0.5 \cdot 0.75 + 0.5 \cdot 0.5) \\ & + 0.02 \cdot (0.5 \cdot 0.75 + 0.5 \cdot 0.75) \\ & + 0.11 \cdot (0.5 \cdot 0.86 + 0.5 \cdot 0.75) \end{aligned}$$

Table 7.7: The final quality assessment score for each expert is based on the quality metrics and ranking that each individual provided from Table 7.6.

Evaluator	Quality Assessment Score
Expert 1	0.65
Expert 2	0.67
Expert 3	0.55
Expert 4	0.56
Expert 5	0.59

In addition to calculating the quality assessment score, the experts are asked to make a comparison of using the KG to their current method of software context extraction from the documentation. Here is an overview of our findings:

Expert 1

The KG is definitely a step in the right direction. It is a slight improvement since there is no current method. The current method that is used, is manually going through the documentation. There are definitely concerns, but the KG did a good job of providing an overview of all the important concepts of the document, giving them a good idea of what the document is about. Therefore, in its current state, it has the potential to be used in collaboration with manual reading.

Expert 2

Similar response as expert 1. No substantial comments to add.

Expert 3

The KG is already technically an improvement because there is no current method. Manually going through the documentation is how the consultants at SIG work. Therefore, in that sense, it is a slight improvement. However, in its current state, the KG would not be a suitable means for extracting software context from the documentation. The KG provides insufficient information. Therefore, an expert would still have to go through everything manually, in that sense the KG will add additional work for the expert. But it is pointed out that the KG does well to highlight keywords, which can then be manually searched for in the documentation.

Expert 4

In an ideal situation, the KG would be very handy, because a consultant can find a starting point in the KG with information they want to know about and traverse the nodes accordingly. This would be a better and quicker alternative to manually going through the documentation. So it definitely has the potential to be a major improvement on the current process of software context extraction. However, in its current state due to its inaccuracies, it would be more suitable to manually go through the documentation.

Expert 5

The information presented in the KG is helpful, however, the issue it is difficult to comprehend the KG in its current state. This is not just because of the inaccuracies, but also because of the visualization of the KG. There are too many nodes currently present in the graph, and the difficulty of looking at the KG currently is that there is no clear starting point. Capping the number of nodes that are displayed at the expense of covering less data would still make this tool a better alternative to the current way of extracting information (the current way is manually reading the documentation). If 70-80% of the information is covered with fewer nodes, it would still be preferable to the current method of manually reading through documentation. With reading a document, the one clear starting point is the beginning of a certain subsection. Whereas with the KG it is difficult, to pinpoint a single starting point unless the user has an idea. A suggestion for figuring out a starting point would be to add weights to the nodes in terms of importance or mentions from the document.

Chapter 8

Discussion

In this chapter, we discuss the results of our research and experiments. First, we look at the results of our Software Context Ontology (SCO) and discuss how it acts as a formal definition of "software context" (answering **RQ1**). The next step is to discuss the results of our Software Context Named Entity Recognition (SCNER) model, and analyse how effective NER is for extracting software context knowledge (answering **RQ2**). Lastly, we will look at the results collected on the Software Context Knowledge Graph (SCKG) presented to the experts, and discuss the overall methodology used to construct our KG (answering **RQ3**).

8.1 Software Context Ontology (SCO)

This study proposes a new Ontology based on the software context domain. Ontologies are a way to define conceptualizations in a domain formally, and the process of formulating our definition is explained in Chapter 3. To validate the SCO, a consistency checker is used, and the final output of the tool indicates that there are no inconsistencies/ errors in relation to the axiomatic rules. Therefore, in terms of consistency, the SCO is valid. Next, the OOPS! scanner detects the pitfalls of the SCO. These results are displayed in 7.1. Based on the results, the majority of the detected pitfalls are in the Minor severity category. With a much smaller percentage of cases falling in the critical or important categories. Based on these results, the SCO is corrected to remove the relevant Critical and Important pitfalls. We say relevant pitfalls as some detected pitfalls did not apply to our ontology. Hence, once the relevant pitfalls are corrected, we place our SCO into the OOPS! Scanner and only the minor pitfalls remain in our ontology. Meaning that based on the pitfall detections of the OOPS! scanner, our SCO is a syntactically valid ontology.

The reasoner used for our consistency check is HermiT. It validates the consistency of an ontology-based on its axioms, rules, and constraints. However, the reasoner still has limitations in terms of performance, shown when the authors that proposed the reasoner discuss the performance compared to other available reasoners [85]. Their results indicate that the reasoner provides inaccuracies when it comes to complex ontologies. For example, the specific examples they point to in their research are complex cyclic axioms and having numerous individual merges of different ontologies causes inaccuracies in the reasoner (both of which we did not have). Although our ontology is high-level and uses only simple ontological components, it is still a factor to consider when using HermiT reasoner. The OOPS! detector also investigates some semantic pitfalls in our SCO (for example looking at concepts to see if they are equivalent or not, and need to properly be differentiated). The lack of semantic pitfalls detected by OOPS! in our ontology, strongly indicates good quality semantic knowledge. However, this tool alone is not sufficient to semantically validate our ontology. This is because OOPS! is created to investigate pitfalls that can be generalized to ontologies in various domains, while our focus is to validate the semantics in the software context domain. Therefore, at this stage, our SCO requires a more thorough semantic validation process outside using OOPS!. This is somewhat of a minor concern because the majority of the ontology is built on the foundation of information and elicitation sessions with experts in the domain. That said, in order to semantically validate the full ontology, it is still highly recommended to hold formal sessions with experts working in the domain to validate the semantics and the content present within the ontology. During our research, this is done informally, where experts briefly look at our ontology and state the ontology provides a well-formed and thorough definition, however, these brief interactions do not qualify for a thorough validation process, and therefore the semantic validation of the SCO will be left to future work.

To evaluate the quality of the SCO, the FOCA methodology is applied to both versions, the Industry Ontology (IO) and the Supplementary Research Ontology (SRO). Equation 7.1 presents the quality result of the IO and Equation 7.2 presents the quality result of the SRO, where the scores are 0.960 and 0.985, respectively. What this indicates is, based on the FOCA methodology, both the IO and SRO are of high quality (as the scoring system states 0 is the lowest and 1 is the highest). There, is a very minor difference between the scores of the IO and SRO. This is because looking at the results in Table 7.3, Q3 and Q13 are the only two questions with differing results. However, looking at how 3.1, you will notice that Goal 5 is not considered in the formula. Meaning that the only difference between the IO and SRO evaluation quality scores is Q3, whether ontologies are reused or not. The difference between the IO and SRO is that the SRO is the IO with supplementary research by adding existing ontology components. Therefore, we emphasize that the higher quality of our SRO, as indicated by the FOCA evaluation scores, can be attributed to its reuse of existing ontologies.

Evaluating an ontology is not an easy task. The evaluation of ontologies is marked with subjectivity, particularly when taking into account the diverse quality dimensions (QD) that evaluators must consider within evaluation frameworks. We select the FOCA methodology instead of alternatives because the methodology is structured using a GQM approach, which allows evaluators to be as objective as possible given the provided criteria and guidelines to grade the questions. To provide clarification: the goals/QDs are based on existing literature, so topics in FOCA are what you expect from state-of-the-art ontology evaluation frameworks, and with FOCA these frameworks are adapted for a GQM approach. To validate the FOCA framework, Bandeira et al. [72] collects results on evaluators applying the framework, and indicated that their approach is valid for ontology evaluation. The results also indicate that the scores of the experienced ontology evaluators compared to the inexperienced ontology evaluators are quite close. This is very important, as FOCA is created with this in mind and our evaluator has no experience with evaluating ontology, and this is a factor taken into consideration when computing the quality score (specifically the $LExp$ variable in Equation 3.1 takes this into consideration in the computation). Hence, this methodology is usable by our inexperienced evaluator as well. Although it should be pointed out, the validation results are based on 6 participating evaluators, which is a small sample size [72]. The structure of FOCA and exact explanations/criteria on how to score each question left less room for subjective scoring. That said, when considering even with clear criteria for each question, the subjectivity of the evaluator still impacts the overall quality score. Our inexperienced evaluator is also responsible for developing the ontology, and although the evaluator followed the FOCA methodology properly and attempted to be as objective as possible, there is always a risk of unconscious bias. Therefore, as part of future work, it would be important to get one of the industry experts to evaluate the ontology as well.

A limitation of the FOCA formula is that it does not consider clarity when calculating the overall quality score. The metric clarity refers to how clarity of the ontology to a user, for example through the documentation, annotations, vocabulary, etc. of the ontology. When it comes to the quality of an ontology, clarity impacts the overall usability of an ontology. In terms of clarity based on our results, the concepts are well written, however, we are missing annotations which can be important for non-experts of the domain to interpret our ontology. Hence, clarity should also be a consideration for the FOCA formula. For now, the importance of clarity is outlined through our results in Table 7.3.

RQ1: How can the concept of "software context" be defined and represented through the creation of a formal ontology?

To answer **RQ1**, we explore an ontology construction methodology to build an ontology in the domain of software context (specifically for the evaluation of software systems). By adhering to our methodology, we successfully achieve a formal conceptualization of the term "software context". Through the application of a consistency checker, we confirm that the ontology adheres to axiomatic rules, establishing its internal consistency. The utilization of the OOPS! scanner helps us identify and rectify pitfalls, ensuring syntactic validity. Additionally, our ontology scores highly based on the FOCA methodology. Overall, our SCO provides a good formal definition of the concept "software context". However, as discussed, there are still some limitations to consider and as part of future work, it would be highly valuable to perform evaluation sessions with experts.

8.2 Software Context Name Entity Recognition (SCNER)

In our research, we propose an NLP-based approach capable of extracting software context information from text using a NER system. Figures 7.4 and 7.5 illustrate the learning curves of our fine-tuned models. Looking at Figure 7.4, we can see that as the model is being trained at each epoch, both the training loss and validation loss are decreasing. Finally, around the fifth epoch, the results start to converge. If you look at the loss curve, you might notice that the validation loss is slightly higher than the training loss. In the cases of some ML models, this is a sign of underfitting. However, Rodrawangpai and Daungjaiboon [86], provide their learning curves for a variety of fine-tuned BERT variants (including RoBERTa) for a text classification task, and the learning curves are very similar to Figure 7.4. Meaning that our model is not underfitting; this is further confirmed based on our evaluation results from Table 7.4 because the model performs well on unseen data. Figure 7.5 indicates that the model has learned properly based on the shape of the learning curve and is making predictions to the software context labels. Our results are based on exact match true positives as explained in Section 5.5, and the validation F1 score reaches nearly 60%, which is a strong indication that the model has learned to predict defined labels. This is because if the model has not learned from the data during the training process, we expect much lower validation F1 scores that show clear signs of randomness in the predictions. Furthermore, obtaining an exact match is a lot more difficult because based on the BIO tagging, explained in Section 4.2, the model needs to predict the "B" and all the "I" tags when an entity spans multiple words. This is not an easy task for a model, meaning that obtaining an F1 score of around 60% is a strong indication that the model has learned from the data and is not randomly predicting labels.

For the evaluation of our model, we calculate the F1 scores present in Table 7.4. The average F1 score is 0.78, which means that our model performs quite well for the task of predicting software context entities. We see that the average precision score is 0.73, while the recall is 0.85. This means that our model predicts more false positives on average than false negatives. False positives correlate to our model making predictions on non-entity types or predicting the wrong label. While false negatives correlate to missing entities. The higher recall means that our model does not miss labelled entities as much as it does mislabel them. When examining the results in Table 7.5, you can see how well the model performs to classify each entity type. Based on the average F1 scores, `Software_Purpose` only has 0.51, while the rest are all higher than 0.77, where the highest is `Internal_Organization` with a score of 0.85. This is very likely because our model is originally pretrained with NSP on non-verb entities. In our dataset `Software_Purpose` entities only contain verbs, which is explained in the annotation guideline in Section 4.2. Whereas the rest of the entities contain nouns and adjectives (with other connecting parts-of-speech). Since the model is already pretrained with millions of noun-based entities, it indicates the highest recognition rates for noun-based entities. Theoretically, exposing the model to more training samples with verb-based entities should greatly improve its prediction capabilities to the same scores. In the 5 training iterations, the F1 score of `Company_Name` ranges from 0.67 to 0.9. Large differences like this are from how the data is shuffled before we split them. It is evident based on all the results that iteration 2 has very few training samples of `Company_Name`. This observation is true about the rest of the entities as well when there are big gaps in the F1 scores between iterations.

After examining the evaluation results of our model, it is clear that the approach performs very well. Let us compare the results to the original reference paper by Huang et al. [35]. They apply the same model, with the same experimental setup on the WikiGold dataset that contains around (1339 samples of data for 4 entities: `PER, ORG, LOC, MISC`) to achieve an F1 score of 0.839, while we use a dataset with 745 samples with 8 entities to achieve an F1 score of 0.78. It should be pointed out that in their research they very likely use exact matches for true positives, which is more difficult to achieve than overlap matches like our evaluation process considers. That said, the data and entities in the WikiGold dataset are a lot less complicated than the entities from our SCD. Meaning, our model has to learn to recognize more complicated data with half the data. It should be noted that our fine-tuned model cannot be compared with Huang et al.'s experimental setup, instead we just want to indicate that their few-shot NER approach is well suited to recognize software context entities using a small dataset.

Our approach has obtains very good results for software context entity recognition, especially for the small dataset used to train, validate and evaluate the model. Although there are still additional considerations to our training approach. While our model performs well on the provided dataset, there can potentially be limitations with the model's generalization ability with other datasets or text within the same domain. This is because, with a smaller dataset, we are more likely to face dataset bias due to a lack of entity diversity. In some cases, it can be that within our dataset the entities follow a very similar sentence structure, and therefore when sentences get more complicated in new unseen data it can be difficult for the model to predict the entities. Additionally, using overlaps has made a guaranteed improvement on the F1 score, instead of using exact matches. If we are to use an exact match for true positives, we would expect the F1 score to range from 55-70%. However, the reason we chose to use overlaps is that we would completely miss a lot of valuable labelled entities. Furthermore, the labels that would be missed are either labels that are missing little information from the exact match or added little information from the exact match. This still meant that our model is predicting very valuable information that overlaps with the exact labelled entity. As our NER model is part of the entity extraction process for our KG development methodology, we clean up the extracted entities with additional processing and obtain high-quality entities, that would have otherwise been lost if we select true positives based on exact matches instead of our current methods of overlap matches.

RQ2: How effectively can relevant context be extracted from software system documentation, using AI techniques, particularly Named Entity Recognition (NER)?

RQ2 focuses on using NER to extract software context from system documentation. The question specifically focuses on the effectiveness of this technique to extract software context. Effectiveness looks at how well this technique performs for our specific use case, and this can be measured using performance metrics.

First and foremost, to answer our research question, we propose an NLP-based approach that leverages a fine-tuned NER system. We validate our approach we collect training and validation results to plot learning curves, that indicate that our model is correctly learning to predict software context entities, validating our model. Once we confirm our model is valid, we evaluate its performance to see how effective the model is to extract software context from system documentation. Our results indicate that with an average F1 score of 0.78, our model performs quite effectively in predicting software context entities. Additionally, our results indicate that our model performs less effectively on certain entities, such as `Software_Purpose`, only achieving an average F1 score of 0.51. While we acknowledge there are challenges related to dataset size and diversity, as previously discussed, our approach's success in capturing complex software context entities confirms its effectiveness. These findings highlight the potential of AI techniques, specifically NER, in improving the extraction of relevant context from software system documentation.

8.3 Software Context Knowledge Graph (SCKG)

In Chapter 6 we outline our approach to constructing KG that attempts to provide a well-rounded depiction of software context based on system documentation. We present the results of our validation in Section 7.3.1. The results indicate the qualitative assessment from 5 experts, where each expert is asked to examine the original software documentation and check if the KG output by our approach contains the knowledge that is expected. After thorough discussions with the experts, we observe that based on the KG provided, there are similar comments about the overall accuracy. The relationships between entities that relate to scalability characteristics are difficult to understand, and semantic meaning is unclear. There are multiple incorrect relationships that are not a good representation of the relationships present within the documentation. The KG does accurately represent a number of relationships, however, this mostly happens when the sentences within the original documentation are structured in a simple manner and are not too complex. Every expert agreed that there seem to be missing relationships based on the text. On the other hand, in terms of coverage, the KG does a good job of covering most of the information about the document, and most experts state that they got a good understanding of what the document is about. This is also mostly linked to the fact that the most important entities are correctly extracted. Furthermore, all the experts point out there is good coverage of the concepts from the entire documentation, and that there we not really any missing concepts, only relations.

Based on the validation sessions and the explanations of the experts, the KG did not present the information that is expected after manually examining the software context from the original document. As indicated by the results, there are inaccurate relationships that provide invalid software context information and cause difficulty in comprehending the software context of the original document. Furthermore, relationships that are expected in the KG are missing. What can be observed from the results is that all the major concerns for inaccuracy come from the relationships within the KG. There are few to no concerns about the entities and the coverage of the entities, in fact, these aspects are praised. One expert did comment that some entities lose some information compared to the original document (e.g. "customers" turns to "customer"), however, this is the result of lemmatizing the extracted entities, therefore a minor issue caused by the processing knowledge step as detailed in Section 6.4.1. Outside of this, all concerns related to the accuracy and validity of the KG relate to the relationships. Therefore, based on the sessions with experts, our approach in its current state is not valid. The invalidity of our approach stems from the use of a rules-based approach for defining the relationships between our extracted entities. In the current process, we define a set of rules that are not directly extracted from the documentation, instead, we formulate a set of rules to generalize the specific entities extracted. Our results indicate that this method of allocating relationships for our KG triples is not very effective, because it does a poor job of generalizing the relationships between much larger and more complex sentences. Furthermore, currently, there are too few rules to capture relationships between all the extracted entities to provide sufficient and clear information; they become too generic to provide substantial semantic meaning. The rules also potentially have an inherent bias, as they are formulated by inspecting training data from the SCD. It is possible that this training data does not offer sufficient diversity to effectively generalize relationships between entities in texts with various contexts. To correct our methodology, we would have to use a new way of extracting relationships. The new technique would have to directly extract information from the text itself. This is because after attempting this rule-based approach, it became evident that it would be quite inefficient to examine a diverse range of software context documentation and formulate rules that are able to generalize relationships. There are too many possibilities as to how sentences describing software documentation can be formulated. Instead, as part of future work, we would like to investigate the use of NLP techniques to directly extract the relationship between entities within the text itself. There is prominent research in this field, and it is much more efficient than trying to construct rules based on the analysis of software documentation samples.

During the evaluation session, we ask the experts a list of questions based on a GQM approach. In Table 7.6, we can see all the scores produced from each session. Based on those scores, we use Equation 6.4 to calculate the results in Table 7.7. What we notice is that during each session, although phrased the same way, with the same context provided, the experts answer the question with a different perspective. For example, experts 1 and 2 both answer the questions much more optimistically, with the perspective that the KG has the potential to provide a good overview of information related to the software context. They definitely point out the flaws but evaluate the KG with the perspective that the KG is currently being researched. On the other hand, expert 3 evaluates the model with the perspective that it is a completed tool that is supposed to be used in industry and should act as a replacement for the current method. The expert said unless it is accurate, it would cause more harm than good in terms of extracting context. While experts 4 and 5 are in the middle in terms of optimism. The perspectives of the experts clearly play a big role in the evaluation scores presented in 7.7, hence why the values are slightly scattered between 0.55 to 0.67. Looking at the quality assessment score based on fundamental KG quality dimensions (QDs), the KG performs alright with lots of room for improvement. From the different perspectives of the experts, we obtain valuable insights into the current state of the KG. When the experts are asked to compare the current method they use to extract software context from documentation to the developed KG, they all state that the KG is a slight improvement. This is because currently there is no method of extracting software context from system documentation outside manual reading. The responses from the evaluation session indicate that for certain experts, using the KG provides a good overview of what documentation will be about, and also allow them to navigate the documentation using the keywords defined in the entities of the KG. Traversing the KG alone would be slightly complicated as there is no starting point like with documentation (the starting point being the beginning of the documentation). Some experts explain that in its current state, the KG would be useful to partially use in collaboration with reading the document, as the KG points out valuable insights on where specific information is (for example, the `Is_Designed_To` relationship points out entities in the documentation to define the purpose of software, however, the KG on its own does not provide sufficient information/clarity on this).

As stated above, the KG provides a quality assessment score between 0.55 to 0.67 based on the sessions with the expert. But based on the scores and qualitative assessments (through answering questions) provided by the experts, there is a strong indication that the KG in its current state is not of good quality, but definitely has the potential to improve. This sentiment is expressed based on all the responses of the expert during the evaluation session. Something else to consider when using the KG quality assessment equation is that each expert has their own custom weighting for each QD based on the ranking they provided. If you look at the rank weights in Table 7.6, there is a slight disparity between the weight of the ranks for certain experts. This disparity also has an impact on the overall quality score, where if certain QDs have a very low rank and others have a very high rank, the entire quality score is skewed by the weight of the higher rank. Of course, this is the intention of having weights in the first place. However, in the future when using this evaluation framework, if clear ranking guidelines are not defined, we could come across special cases where the disparity is large enough to cause an undesirable skew in the quality scores. For example, looking at the results of expert 1, would they want the relevancy of the KG to be 17.5 times more important than the believability, or would this cause a skew that the evaluator did not expect? For now, we use Equation 6.2, however without clear ranking guidelines, we run the risk of undesirable skews in the quality score.

RQ3: To what extent can a Knowledge Graph development methodology be employed to provide a holistic representation of software context from documentation?

In answering this research question, we consider three "holistic" aspects to evaluate the effectiveness of our approach in the representation of software context within a KG. The three aspects are: comprehensive coverage, contextual understanding, and methodological soundness.

Comprehensive Coverage: Our initial concern is to see if our approach allows a KG to cover all the important software context information in a document, ensuring comprehensive coverage. Our SCKG demonstrates the ability to cover a significant portion of the information present in software documentation. During validation sessions, experts note that the KG effectively captures and represents the most important entities and concepts from the documents. This suggests that the SCKG has the potential to comprehensively cover the information contained in the documentation, especially when dealing with well-structured and straightforward sentences. However, since we use our fine-tuned SCNER model to extract entities, the same limitations we discuss in section 8.2 apply. Furthermore, when it comes to covering the relationships, we know our approach has limitations as it misses relationships that are present in the system documentation.

Contextual Understanding: For contextual understanding, we check whether the KG is able to provide a good overview of the contextual significance of all the information obtained from the documentation. Through our discussion, we discover that our approach in its current state is partially invalid because of the method used to assign relationships between entities. Due to the inaccuracies created by the relationships present in the SCKG, in terms of contextual understanding, our approach currently lacks the ability to provide a deep understanding of how various software context elements are interconnected within a software ecosystem. This is not only because of the inaccuracies in the relationships, but also due to missing relationships that are described within the system documentation. As part of our future work, we plan to investigate alternative techniques that can accurately extract relationships directly from documentation.

Methodological Soundness: Methodological soundness is about checking whether a KG graph is a good way of representing software context from documentation in the first place. Currently, based on our research, using a KG as a representation of software context from documentation shows promise but also reveals areas for improvement. Our evaluation of the KG also indicates there is a potential to produce high-quality KGs to represent software context. This is because, although the overall quality assessment score is lower due to inaccuracies that undermine the KG's validity, there is still evidence from the analysis of QDs and QMs in Table 7.6 that the SCKG performs well for certain QDs and QMs. For example, the SCKG performs well in entity extraction and coverage. However, it faces challenges in accurately defining relationships between entities, especially when documents contain complex sentences. This issue stems from the rule-based relation mapping approach that we use.

The KG's current limitations of our KG development approach are primarily related to the rule-based matching method used for the relationship definition. It leads to inaccuracy and smaller coverage of the relationships within system documentation, consequently affecting the contextual understanding and methodological soundness of the SCKG. However, our evaluation results show that directly extracting relations from the text has the ability to improve the quality of the SCKG a great deal. To enhance the quality and effectiveness of our approach, the primary focus of future work should be to explore alternative methods, such as natural language processing (NLP) techniques, to directly extract relationships from text.

Chapter 9

Related work

In this chapter, we present studies in the field related to those that are covered in our research.

9.1 Ontology Construction

The development of formal ontologies has been a fundamental aspect of knowledge representation in various domains, providing a structured framework for organizing concepts, relationships, and domain-specific knowledge. Within the field of software engineering, the construction, and validation of ontologies gain prominence due to their potential to improve information retrieval, software understanding, and the decision-making processes. Existing literature offers a range of methodologies and approaches for ontology construction, each with its unique considerations and trade-offs. For example, Noy and McGuinness [42] present their Ontology Development 101, where they provide a step-by-step guide on how to develop an ontology for declarative frame-based systems. In their work, they address certain complex issues, such as defining class hierarchies and properties of classes and instances. Fernández-López et al. [87] introduce a paper outlining a series of steps that make up the ontology development process. They also propose a life cycle approach for constructing ontologies using evolving prototypes and present Methodology, a structured methodology employed for building ontologies from scratch. These two ontology construction methodologies, although written over 20 years ago, still greatly influence modern-day methodologies. For example, a relevant paper that is published by Sun et al. [40], where they adopt a comprehensive ontology construction method based on the Dev. 101 [42] and Methontology [87] methods to build a software testing ontology. In another example, after thorough discussions and analyses of prominent ontology-building methodologies, Yun et al. [38] propose a knowledge engineering approach to construct domain ontology. This approach involves combining the software development life cycle standard IEEE 1074-2006 with the design ontology criteria proposed by T. R. Gruber [22]. However, Yun et al. [38] discuss both Dev. 101 [42] and Methontology [87] and highlight the limitations of the methodologies for their specific research goals, but still discuss their value. This just goes to show that there is still great importance placed on these fundamental development techniques. Since the paper by Sun et al. [40] and Yun et al. [38] incorporate the fundamentals of ontology development while highlighting the importance of each step, we incorporate aspects of their studies into our process as well. This is described in Section 3.1.

9.1.1 Ontologies for the Software Engineering Domain

To our knowledge, there have been no ontologies constructed for the domain of software context for system evaluation. In Section 3.6, we introduce existing ontologies based on related work that we have studied and incorporated as part of our Software Context Ontology (SCO). Outside these ontologies, there are a few other ontologies in the software engineering domain. As there are no software context ontologies, it is important to consider other research in the software domain. The purpose is to see if our ontology construction process is drastically different, and if so, understand if there is a good reason why.

The SCO provides a definition of software context and assists with the evaluation of the software. Similar to the SCO, there are ontologies in the software engineering domain that assist software engineers with the tasks of the domain they are developed for. In order to provide projects with a common language and improve the distributed software development process, Rocha et al. [88] introduces the knowledge base known as DKDOnto, a domain-specific ontology. To express software engineering knowledge for a multi-site software development, Wognthongtham et al.[89] provide a software engineering ontology. The software engineering ontology acts as a communication framework, facilitating the exchange of semantic project data. Its primary users are software engineers engaged in the exchange of project data and domain knowledge within the realm of software engineering. Furthermore, the development and analysis of this ontology are explained in a follow-up paper [90]. After examination, all the papers use methodologies that incorporate standard ontology construction practices, therefore there are no drastic differences from our approach.

9.1.2 Ontology Evaluation Frameworks

Outside the FOCA framework [72] there are multiple ontology evaluation frameworks. All the available frameworks evaluate the ontology based on a set of criteria, whether that is based on a goal, quality dimension (QD), or synonym. Gruber [22] first discusses how ontologies might help knowledge-sharing activities before outlining a set of standards that should be followed when creating ontologies for these uses. Gómez-Pérez [91] provides a comprehensive overview of earlier studies on ontology evaluation, including the criteria (consistency, completeness, conciseness, expandability, and sensitiveness) that are employed. Note that the criteria from Gruber and Gómez-Pérez influenced many current ontology evaluation frameworks, including FOCA methodology. Raad and Cruz [92] present ontology evaluation strategies while outlining their benefits and shortcomings in order to solve the problem of developing an effective ontology assessment method. In their paper, they present a survey of ontology evaluation methods in which the criteria they use to conduct the survey are also influenced by the works of Gruber and Gómez-Pérez.

9.2 Named Entity Recognition (NER) for Software Related Text

In recent years, NER has increasingly been adopted for information extraction from software-related documents, however, the field is very much still in its infancy. For this reason, to the best of our knowledge, there is currently no research being done to investigate software context extracted in the realm of NER. That said, prior research in NER has yielded valuable insights and oversight of potential techniques that contribute to the effective extraction of software context information.

There is a variety of NER for software-related text, however, all the research investigates different types of text. Ye et al. [12] introduce a model that can recognize a broad category of software entities for a wide range of popular programming languages, platforms, and libraries. It is created using a Conditional Random Fields (CRF) learning-based architecture and uses data for Q&A discussions on Stack Overflow. Reddy et al. [13] present NERSE, a tool that enables the user to identify 22 software-specific entities. They train on software-specific entity categories using CRF and Bidirectional Long Short-Term Memory Conditional Random Fields (BiLSTM-CRF). Zhou et al. [15] propose an approach for bug-specific entity recognition called BNER with the CRF model and word embedding technique. This research is further developed later by Zhou et al. [14], where they propose a deep neural network approach for bug-specific entity recognition called DBNER using bidirectional long short-term memory (LSTM) with Conditional Random Fields decoding model (CRF). The bug-specific entities are based on bug reports, which are free-form texts that may contain code, abbreviations, and software-specific vocabularies.

More recent research investigates software-related NER using a transformer-based approach, similar to our research. Using 15,372 phrases tagged with 20 fine-grained item categories, Tabassum et al. [11] present a NER dataset for the computer programming area in this study. Furthermore, from 152 million phrases from StackOverflow, they trained in-domain BERT representations to propose BERTOverflow. The transformer models BERT, RoBERTa, and ALBERT are used by Malik et al. [75] to extract entities related to software requirements. They annotate three requirement datasets—DOORS, SRE, and RQA—with various sets of software-specific attributes for this reason.

9.3 Knowledge Graph Development

Knowledge Graph (KG) development is a fundamental aspect of representing structured information, and it has found applications in various domains, including but not limited to healthcare, finance, and the semantic web. In the context of software engineering, KGs play a pivotal role in capturing the intricate relationships between software artifacts and their contextual information. Previous research efforts explore different methodologies for constructing KGs in software engineering. For our research, we followed the KG Development Process proposed by Tamašauskaitė and Groth [24], where they conduct a conceptual analysis of existing KG development literature and provide a systematic review. Their paper provides an overview of methodologies used for a variety of KGs and defines a global view of the KG development steps that are applicable to any domain.

In relation to the software engineering domain, there are not many KGs. Jihu et al. [93] propose a software development knowledge graph (SoftKG) using Wikipedia taxonomy. The purpose of this KG is to assist with the problem of finding solutions when running into programming problems. There are limitations to using search engines or consulting senior developers: software development information is scattered across various websites, requiring multiple searches, and lacks unified organization, making it inefficient; SoftKG aims to address these limitations.

Additionally, to our knowledge, there are no investigations done on transforming documentation into KGs. In relation to documentation and, KGs, Stylianou et al. [94] propose a solution to enhance Document Management Systems (DMS) by introducing a framework called Doc2KG (Document-to-Knowledge-Graph). This framework handles both the creation and real-time updating of knowledge graphs from textual information, overcoming limitations related to the use of user-created metadata. Therefore, this work involves analysing information from documentation in order to update KGs, whereas in our research we analyse text to construct a completely new KG.

9.3.1 Relation Extraction (RE)

In our research, we use a rule-based approach to match the relationships between entities. Since we do not directly extract relations from text, our approach cannot be considered RE. This is our alternative to RE, based on our available resources. In the current scope of RE, machine learning and deep learning-based approaches are increasingly studied due to their ability to obtain high-performing results. In this section, we will describe a variety of RE approaches, many of which are described in a literature survey provided by Detroja et al. [26].

Unsupervised methods do not require any labelled data. The majority of unsupervised RE techniques take a clustering-based strategy. A technique that uses hierarchical clustering for the purpose of determining semantic similarity using entity similarity graphs and hypernymy graphs is WEBRE [95]. WEBRE functions in two stages: first, it discovers semantic classes that represent a significant number of relations; second, it groups related relations together.

Supervised methods on the other hand require a large amount of training data, which is annotated with a set of entities and relations. It trains a classifier using training data, and the classifier then utilizes test data to identify relationships. There are two categories of supervised methods: Feature-based approaches and Kernel-based approaches. In feature-based methods, such as those presented by Rink and Harabagiu [96], Kambhatla [97], and Zhou et al. [98], a collection of features is crafted for each relation within the training data. Subsequently, a classifier is trained to extract new instances of relations. Kernel-based methods, as explored by Bunescu and Mooney [99], Culotta and Sorensen [100], and Zhang et al. [101], do not use preprocessing steps for to design the features. Instead, these approaches rely on kernel functions to assess the similarity between representations of two relation instances, employing Support Vector Machine (SVM) for classification. The effectiveness of a kernel-based RE system depends on the thoughtful design of these kernel functions.

Many deep learning architectures can be employed for NLP tasks like RE. Mokoto and Bansal [102] propose an end-to-end neural model for entity and RE. Their model employs a bidirectional tree-structured LSTM-RNNs stacked on bidirectional sequential LSTM-RNNs, enabling it to capture word sequence and dependency tree substructure information. This architecture allows for the simultaneous representation of entities and relations within a single model, sharing parameters. Transformer-based models are also being greatly researched for RE, particularly for domain-specific text. These models can be fine-tuned for specific downstream tasks (like how we fine-tune a transformer for NER in this thesis). Typically, the difference between each task is their NLP pipeline, while the model architecture is the same. For example, in the research conducted by Yang et al. [103], three transformer-based models (BERT, RoBERTa, and XLNet) are evaluated for RE for the clinical domain. The findings showcase the superior performance of the RoBERTa-clinical RE model, achieving the highest F1-score of 0.8958 on the 2018 MADE1.0 dataset. Furthermore, there are techniques to assist with the lack of data (few-shot RE frameworks), that can be applied in collaboration with transformer models. For example, Dong et al. [104] propose a framework that takes into account both label-agnostic and label-aware semantic mapping information to enhance low-resource/few-shot RE. Their findings reveal that integrating these two forms of mapping information during both the pretraining and fine-tuning stages leads to a substantial enhancement in model performance for low-resource RE tasks. The frameworks can be adapted to use various encoders, including BERT and RoBERTa. In relation to future work, we suggest adapting this few-shot relation extract approach using the same RoBERTa model we use for NER.

9.3.2 Knowledge Graph Evaluation

With the increase in KG development over the past decade, research to evaluate KGs has also slowly advanced. Research related to KG evaluation follows very similar structures; researchers provide a set of criteria that an evaluator must follow to evaluate the overall quality, and in many cases, the criteria show similarities. This is a good indication, as it means that there are certain quality dimensions that are of higher importance regardless of the knowledge domain. The criteria we used for our research is a practical quality assessment framework for KGs that is customizable to a particular domain [83]. In the framework, Huaman added quality dimensions (QDs) and quality metrics (QMs) that are unique to KGs to the existing state-of-the-art frameworks. We also incorporated certain aspects of the requirements of a KG proposed in the practical KG evaluation framework by Chen et al. [84]. In their paper, they conduct a comprehensive study of existing frameworks and propose their own framework designed to assess the appropriateness and quality of "fit for purpose" of KGs. In other related research; Wang et al. [105] provide a survey on the quality control of KGs. Within their paper, they define 6 evaluation dimensions for KG quality and analyse their correlations and distinctions. They also discuss quality control methods applied during KG construction, considering these quality dimensions. Lastly, the paper discusses strategies for enhancing KG quality across multiple dimensions after its construction. Seo et al. [106] propose slightly different, more objective quantitative assessment criteria. Their paper presents six structural quality metrics designed to evaluate the quality of knowledge graphs. These metrics have been employed to analyse cross-domain KGs found on the internet. These graphs include well-known sources such as Wikidata, DBpedia, YAGO, Google Knowledge Graph, and Freebase, as well as Raftel (Naver's KG).

9.4 Summary

In this section, we summarize the literature reviewed in the previous sections, focusing on specific topics of our research subdomains. Table 9.1 presents a summary comparing various aspects of the papers discussed in this chapter. Additionally, we also include our SCO, SCNER, and SCKG to highlight the differences between our research and the research in related work.

It's important to note that we haven't compared KG Development Methods or KG Evaluation Criteria. The reason is that these related works employ different methods and criteria compared to our SCKG and to each other. Therefore, it would not be a sensible comparison in our summary. Furthermore, we exclude the papers related to KGs as there are no concepts to compare them to. The terms *Ontology Development Methods*, *Ontology Evaluation Criteria*, *NER Approach*, *Defining Relations* summarize the comparable information presented across our work and the related work. In Table 9.1, a cross denotes that this topic is prevalent in the research and is used by the proposed approach (either a paper or our work). The columns are organized by topic, with the order from SCO to SCNER focusing on ontologies, from SCNER to SCKG related to NER, and from SCKG onwards addressing relationship definitions.

Table 9.1: Summary of Concepts used/followed in related work and our SCO, SCNER, and SCKG.

		SCO	**[38]**	**[40]**	**[88]**	**[89]**	***SCNER***	**[12]**	**[13]**	**[15]**	**[14]**	**[11]**	**[75]**	***SCKG***	**[95], [96], [97], [100], [99], [101]**	**[102], [103]**	**[104]**
Ontology Development Methods	**Dev 101. [42]**	X		X													
	Methontology [87]	X		X	X												
	Others	X	X			X											
Ontology Evaluation Criteria	**FOCA [72]**	X		X													
	Gruerber Criteria [22]	X	X	X													
	Gómez-Pérez Criteria [91]	X		X													
	Others		X		X	X											
NER Approach	**Deep Learning**						X		X		X	X	X				
	Machine Learning							X	X	X	X						
	Few-Shot Learning						X										
Defining Relations	**Deep Learning RE**															X	X
	Machine Learning RE														X		
	Few-Shot Learning RE																X
	Rule-based Relation Matching													X			

Chapter 10

Conclusion

The project's inception was driven by the increasing necessity to effectively obtain contextual information from software documentation throughout the process of developing and evaluating software. There was a noticeable research gap in this specific domain, presenting an opportunity to enhance the methods of extracting and presenting software context-related information, thereby contributing to the advancement of software engineering. To investigate this research gap, we researched the process to define the concept of "software context" using a formal ontology. This was done by following a thorough ontology development process led by academic literature. Once fully developed and evaluated, we used this definition of software context to aid in our second research goal. To investigate how effectively software context can be extracted from documentation using a Natural Language Processing (NLP) based approach, specifically Named Entity Recognition (NER). By constructing a custom dataset with 8 software context entities, we were able to fine-tune a few-shot NER model to predict software context entities within the text. Lastly, with the model developed, we explored a Knowledge Graph (KG) development approach in an attempt to provide a comprehensive overview of software context in system documentation.

Through our investigation, we contribute a formal definition of software context using an ontology. Using the HermiT reasoner, the OOPS! scanner, and the FOCA methodology, our results indicate that our ontology is consistent, syntactically valid, and of high quality. Our results imply that our Software Context Ontology (SCO) is a well-built ontology that provides a concise, high-level definition of the software context domain. Furthermore, in its current state, it should be reusable for additional research or use cases, however, it is still suggested to first have validation and evaluation sessions with industry experts before being used. It should also be noted that there were limitations to the FOCA methodology, and therefore the almost perfect score of 0.985 is not the best representation and the FOCA formula does not consider other very important quality dimensions (QDs).

In the course of our exploration, we contribute a specialized NER dataset designed for the software context domain. Additionally, we contribute our NER-based approach for extracting software context information from documentation. The approach performed very well, resulting in our models obtaining on average an F1 score of 0.78 With consideration that our model was only fine-tuned on 745 data samples, while most NLP models are fined on much larger amounts of data (even millions of samples), we obtain a very well-performing model that can be further improved in future research. Although our results are very good, it is also important to consider the possibility of dataset bias, and therefore in the future, it will be important to evaluate the performance of our approach using a much larger and diverse range of unseen data as well.

Our final contribution is the investigation of a KG development approach to present software context from documentation. Our results indicate that in its current state, the approach is not fully valid. This is mainly due to the method used to connect relationships between entities; the rule-based approach creates inaccuracies, missing information and reduces the overall quality of the KG. However, our results, show a strong indication that a relation extraction method to our approach will provide us with a fully valid, high-quality KG. Therefore, this will be one of the main focuses of future work.

10.1 Future work

In this section, we discuss the possible directions for future work to supplement the work of this thesis.

10.1.1 Refinement of Software Context Ontology (SCO)

For future work, it is important to investigate and implement a more thorough semantic validation process for the SCO, involving formal validation sessions with domain experts. Without this session, we cannot confidently conclude that the SCO is semantically valid, as this is a necessary part of the scientific method. It would also be valuable to fix the "Minor" pitfalls detected by the OOPS! scanner on the SCO to improve the overall quality of the ontology. Furthermore, we would like to extend the FOCA ontology evaluation formula to incorporate considerations of clarity and usability, to provide a more well-rounded overview of our SCO.

10.1.2 Enhancement of Software Context Named Entity Recognition (SC-NER) Model

In order to improve the current SCNER model, we would like to explore techniques to enhance the generalization ability of the SCNER model with larger and more diverse datasets, addressing dataset bias issues. Data augmentation would be a possible solution to this issue. One potential paper to investigate for instance, written by Dai and Adel provides [107] an investigation of data augmentation techniques that have show the improvement of the performance of transformer-based NER models.

Additionally, we would like to consider incorporating more training samples with verb-based entities to improve predictions for specific entity types like `Software_Purpose`. Understanding how the model behaves in these instances will give us more clarity on how to adapt the NER model for the KG development approach.

In future research, we would also like to experiment with different NER models and architectures to potentially achieve better F1 scores and entity extraction capabilities. In recent years, there have been newer models developed that are more lightweight and outperform RoBERTa (like ALBERT [74]). However, it would require additional resources to adapt these models to this style of training and therefore should be investigated for future research.

Lastly, we would like to compare the performance of the SCNER model with other state-of-the-art NER models in the context of software documentation to understand its strengths and weaknesses. This process would require fine-tuning other state-of-the-art NER models under the same condition to provide a fair comparison.

10.1.3 Improvement of Software Context Knowledge Graph (SCKG)

The one major focus of our future work is to fully validate our SCKG. Therefore, we must investigate and implement ML or deep learning techniques to directly extract relationships between entities from the text, improving the accuracy and validity of the KG. In Section 9.3.1 we introduced the current scope of ML and deep learning-based techniques used for relations extraction that directly extract relations from text. These techniques set a clear research direction to directly address the concerns related to the semantic meaning, accuracy of the relationships, and overall quality of the KG and our approach. We suggest adapting the framework developed by Dong et al. [104]. This framework incorporates both label-agnostic and label-aware semantic mapping information, aiming to improve the effectiveness of low-resource relation extraction. The framework was implemented using the BERT_{Base}[1] for both encoders and achieved state-of-the-art results for few-shot relation extraction. This is ideal because then insufficient training data from SCD will also not be a concern.

It is important that we further validate and refine the KG evaluation framework by involving a larger number of expert evaluators to reduce potential bias and increase the reliability of the quality assessment scores. Therefore, we would like to conduct an empirical study where we compare the performance of the current method of software context information extraction vs our tool/approach for software context

[1]https://huggingface.co/bert-base-uncased

information extraction. This empirical study would be performed once our current approach moves away from rule-based methods and directly extract relations from the software documents. We would additionally expect to perform this on multiple KGs constructed with multiple software documents because, in our validation and evaluation sessions, we were only able to investigate the KG from one document.

The scalability of our developed tool to handle larger software documentation should also be further investigated, as one of the experts pointed out, ensuring that they can effectively process and extract context from diverse sources. Currently, the representation of the KG becomes excessively crowded even with small documentation, thus lacking scalability. Additionally, as part of accommodating for scale, we would like to improve the overall UX/UI of the knowledge graph. So that instead of outputting a single image, a user is able to click on nodes and edges to point out where the information was extracted from, zoom in and out of the KG, reposition the nodes and improve the overall visualization of the KG to accommodate for larger documentation inputs.

10.1.4 Exploration in combination with other AI techniques

As part of future work, it would be interesting to investigate the integration of other AI techniques, particularly topic modelling, to provide additional context and insights to the developed tools for software context understanding. Topic modelling has the potential to save lots of additional time, by pinpointing the general topics discussed within large amounts of text, therefore saving time for traversing through large documents. Exploring the application of this technique within the framework of the tools developed through our research holds significant potential for future research.

Acknowledgements

Throughout the course of my thesis, I have been very fortunate to receive invaluable support and guidance from several people. I would first like to express my deepest gratitude to my supervisor, Dr. A.M. Oprescu. I am grateful for the valuable advice, knowledge, and perspectives that Dr. Oprescu shared during our meetings, particularly when discussing the scope of my thesis and how to approach using the scientific method.

Next, I would like to extend my heartfelt thanks to my supervisor, Xander Schrijen. Throughout the entire process, you were always ready to have a chat about my ideas and guide me in the right direction. I greatly enjoyed our weekly meetings where I received great advice on how to tackle challenges, and I appreciate that Xander was always available to assist me if I came to him with questions.

I would also like to show my gratitude to my daily supervisor, Damian Frölich, for his unwavering support and willingness to engage in discussions when I encountered difficulties. Our conversations helped me get a clear understanding of how to tackle these challenges.

Lastly, I would like to express my appreciation to my colleagues at SIG. Your support and willingness to assist with the numerous interviews have made a significant impact on this project. Thank you all for being an integral part of this research project.

Bibliography

[1] *What is the system context?* https://t2informatik.de/en/smartpedia/system-context/, [Accessed 16-Aug-2022].

[2] N. Heist, S. Hertling, D. Ringler, and H. Paulheim, “Knowledge graphs on the web - an overview,” *CoRR*, vol. abs/2003.00719, 2020. arXiv: 2003.00719. [Online]. Available: https://arxiv.org/abs/2003.00719.

[3] C. Reyes-Peña and M. Tovar-Vidal, “Ontology: Components and Evaluation, a Review,” *Research in Computing Science*, vol. 148, no. 3, pp. 257–265, Dec. 2019, ISSN: 1870-4069. DOI: 10.13053/rcs-148-3-21. [Online]. Available: http://rcs.cic.ipn.mx/2019_148_3/Ontology_%5C%20Components%5C%20and%5C%20Evaluation_%5C%20a%5C%20Review.pdf (visited on 04/14/2023).

[4] F. Ruy, R. Falbo, M. Barcellos, S. Dornelas Costa, and G. Guizzardi, “Seon: A software engineering ontology network,” Nov. 2016, pp. 527–542, ISBN: 978-3-319-49003-8. DOI: 10.1007/978-3-319-49004-5_34.

[5] P. Bourque, R. E. Fairley, and I. C. Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd. Washington, DC, USA: IEEE Computer Society Press, 2014, ISBN: 0769551661.

[6] *What is a knowledge graph?* https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/, [Accessed 29-Dec-2022].

[7] D. Fensel *et al.*, “Introduction: What is a knowledge graph?” In *Knowledge Graphs: Methodology, Tools and Selected Use Cases.* Cham: Springer International Publishing, 2020, pp. 1–10, ISBN: 978-3-030-37439-6. DOI: 10.1007/978-3-030-37439-6_1. [Online]. Available: https://doi.org/10.1007/978-3-030-37439-6_1.

[8] H. Ye, N. Zhang, H. Chen, and H. Chen, *Generative Knowledge Graph Construction: A Review*, arXiv:2210.12714 [cs], Dec. 2022. DOI: 10.48550/arXiv.2210.12714. [Online]. Available: http://arxiv.org/abs/2210.12714 (visited on 12/02/2022).

[9] I. Melnyk, P. Dognin, and P. Das, *Knowledge Graph Generation From Text*, arXiv:2211.10511 [cs], Nov. 2022. DOI: 10.48550/arXiv.2211.10511. [Online]. Available: http://arxiv.org/abs/2211.10511 (visited on 11/22/2022).

[10] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, “Code and named entity recognition in stackoverflow,” *CoRR*, vol. abs/2005.01634, 2020. arXiv: 2005.01634. [Online]. Available: https://arxiv.org/abs/2005.01634.

[11] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, *Code and Named Entity Recognition in StackOverflow*, arXiv:2005.01634 [cs], Nov. 2020. DOI: 10.48550/arXiv.2005.01634. [Online]. Available: http://arxiv.org/abs/2005.01634 (visited on 01/18/2023).

[12] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, “Software-Specific Named Entity Recognition in Software Engineering Social Content,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, Mar. 2016, pp. 90–101. DOI: 10.1109/SANER.2016.10.

[13] M. Veera Prathap Reddy, P. V. R. D. Prasad, M. Chikkamath, and S. Mandadi, “NERSE: Named Entity Recognition in Software Engineering as a Service,” en, in *Service Research and Innovation*, H.-P. Lam and S. Mistry, Eds., ser. Lecture Notes in Business Information Processing, Cham: Springer International Publishing, 2019, pp. 65–80, ISBN: 9783030322427. DOI: 10.1007/978-3-030-32242-7_6.

[14] C. Zhou, B. Li, and X. Sun, "Improving software bug-specific named entity recognition with deep neural network," en, *Journal of Systems and Software*, vol. 165, p. 110 572, Jul. 2020, ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110572. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220300534 (visited on 01/20/2023).

[15] C. Zhou, B. Li, X. Sun, and H. Guo, "Recognizing Software Bug-Specific Named Entity in Software Bug Repository," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, ISSN: 2643-7171, May 2018, pp. 108–10 811.

[16] W. Lv, Z. Liao, S. Liu, and Y. Zhang, "MEIM: A Multi-source Software Knowledge Entity Extraction Integration Model," en, *Computers, Materials & Continua*, vol. 66, no. 1, pp. 1027–1042, 2020, ISSN: 1546-2226. DOI: 10.32604/cmc.2020.012478. [Online]. Available: https://www.techscience.com/cmc/v66n1/40495 (visited on 01/23/2023).

[17] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A Survey on Knowledge Graphs: Representation, Acquisition and Applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 494–514, Feb. 2022, arXiv:2002.00388 [cs], ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2021.3070843. [Online]. Available: http://arxiv.org/abs/2002.00388 (visited on 07/26/2023).

[18] P. Cimiano and H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semant. Web*, vol. 8, no. 3, pp. 489–508, Jan. 2017, ISSN: 1570-0844. DOI: 10.3233/SW-160218. [Online]. Available: https://doi.org/10.3233/SW-160218.

[19] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs," in *International Conference on Semantic Systems*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:8536105.

[20] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017. DOI: 10.1109/TKDE.2017.2754499.

[21] O. Corcho, M. Fernández-López, and A. Gómez-Pérez, "Ontological Engineering: Principles, Methods, Tools and Languages," en, in *Ontologies for Software Engineering and Software Technology*, C. Calero, F. Ruiz, and M. Piattini, Eds., Berlin, Heidelberg: Springer, 2006, pp. 1–48, ISBN: 9783540345183. DOI: 10.1007/3-540-34518-3_1. [Online]. Available: https://doi.org/10.1007/3-540-34518-3_1 (visited on 04/13/2023).

[22] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993, ISSN: 1042-8143. DOI: https://doi.org/10.1006/knac.1993.1008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1042814383710083.

[23] M. Chmielewski, M. Paciorkowska, and M. Kiedrowicz, "A semantic similarity evaluation method and a tool utilised in security applications based on ontology structure and lexicon analysis," in *2017 Fourth International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*, 2017, pp. 224–233. DOI: 10.1109/MCSI.2017.46.

[24] G. Tamašauskaitė and P. Groth, "Defining a Knowledge Graph Development Process Through a Systematic Review," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, 27:1–27:40, Feb. 2023, ISSN: 1049-331X. DOI: 10.1145/3522586. [Online]. Available: https://dl.acm.org/doi/10.1145/3522586 (visited on 04/04/2023).

[25] F. Li, W. Xie, X. Wang, and Z. Fan, "Research on optimization of knowledge graph construction flow chart," in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9, 2020, pp. 1386–1390. DOI: 10.1109/ITAIC49862.2020.9338900.

[26] K. Detroja, C. Bhensdadia, and B. S. Bhatt, "A survey on relation extraction," *Intelligent Systems with Applications*, vol. 19, p. 200 244, 2023, ISSN: 2667-3053. DOI: https://doi.org/10.1016/j.iswa.2023.200244. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667305323000698.

[27] M. Marrero, J. Urbano, S. Sánchez-Cuadrado, J. Morato, and J. M. Gómez-Berbís, "Named Entity Recognition: Fallacies, challenges and opportunities," en, *Computer Standards & Interfaces*, vol. 35, no. 5, pp. 482–489, Sep. 2013, ISSN: 0920-5489. DOI: 10.1016/j.csi.2012.09.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548912001080 (visited on 07/03/2023).

[28] B. Mohit, "Named Entity Recognition," en, in *Natural Language Processing of Semitic Languages*, ser. Theory and Applications of Natural Language Processing, I. Zitouni, Ed., Berlin, Heidelberg: Springer, 2014, pp. 221–245, ISBN: 9783642453588. DOI: 10.1007/978-3-642-45358-8_7. [Online]. Available: https://doi.org/10.1007/978-3-642-45358-8_7 (visited on 07/26/2023).

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2018. DOI: 10.48550/ARXIV.1810.04805. [Online]. Available: https://arxiv.org/abs/1810.04805 (visited on 07/26/2023).

[30] A. Parnami and M. Lee, *Learning from Few Examples: A Summary of Approaches to Few-Shot Learning*, en, Mar. 2022. [Online]. Available: https://arxiv.org/abs/2203.04291v1 (visited on 07/27/2023).

[31] R. Vilalta, C. Giraud-Carrier, P. Brazdil, and C. Soares, "Inductive transfer," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 545–548, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_401. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_401.

[32] Z. Wang, K. Zhao, Z. Wang, and J. Shang, *Formulating Few-shot Fine-tuning Towards Language Model Pre-training: A Pilot Study on Named Entity Recognition*, arXiv:2205.11799 [cs], Oct. 2022. DOI: 10.48550/arXiv.2205.11799. [Online]. Available: http://arxiv.org/abs/2205.11799 (visited on 06/16/2023).

[33] A. Vaswani *et al.*, *Attention Is All You Need*, arXiv:1706.03762 [cs], Jul. 2023. DOI: 10.48550/arXiv.1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762 (visited on 07/27/2023).

[34] Y. Liu *et al.*, *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, arXiv:1907.11692 [cs], Jul. 2019. DOI: 10.48550/arXiv.1907.11692. [Online]. Available: http://arxiv.org/abs/1907.11692 (visited on 07/27/2023).

[35] J. Huang *et al.*, *Few-Shot Named Entity Recognition: A Comprehensive Study*, arXiv:2012.14978 [cs], Dec. 2020. DOI: 10.48550/arXiv.2012.14978. [Online]. Available: http://arxiv.org/abs/2012.14978 (visited on 06/16/2023).

[36] E. Zheltonozhskii, C. Baskin, A. Mendelson, A. M. Bronstein, and O. Litany, "Contrast to Divide: Self-Supervised Pre-Training for Learning with Noisy Labels," in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, arXiv:2103.13646 [cs], Jan. 2022, pp. 387–397. DOI: 10.1109/WACV51458.2022.00046. [Online]. Available: http://arxiv.org/abs/2103.13646 (visited on 08/08/2023).

[37] A. Ghaddar and P. Langlais, "WiNER: A Wikipedia annotated corpus for named entity recognition," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Taipei, Taiwan: Asian Federation of Natural Language Processing, Nov. 2017, pp. 413–422. [Online]. Available: https://aclanthology.org/I17-1042.

[38] H. Yun, J. Xu, J. Xiong, and M. Wei, "A Knowledge Engineering Approach to Develop Domain Ontology:" ng, *International Journal of Distance Education Technologies*, vol. 9, no. 1, pp. 57–71, Jan. 2011, ISSN: 1539-3100, 1539-3119. DOI: 10.4018/jdet.2011010104. [Online]. Available: https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jdet.2011010104 (visited on 05/26/2023).

[39] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa, "Oops! (ontology pitfall scanner!): An on-line tool for ontology evaluation," *Int. J. Semant. Web Inf. Syst.*, vol. 10, no. 2, pp. 7–34, Apr. 2014, ISSN: 1552-6283. DOI: 10.4018/ijswis.2014040102. [Online]. Available: https://doi.org/10.4018/ijswis.2014040102.

[40] Z. Sun, C. Hu, C. Li, and L. Wu, "Domain ontology construction and evaluation for the entire process of software testing," *IEEE Access*, vol. 8, pp. 205 374–205 385, 2020. DOI: 10.1109/ACCESS.2020.3037188.

[41] M. C. Suárez-Figueroa and A. Gómez-Pérez, "Ontology Requirements Specification," en, in *Ontology Engineering in a Networked World*, M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, and A. Gangemi, Eds., Berlin, Heidelberg: Springer, 2012, pp. 93–106, ISBN: 9783642247941. DOI: 10.1007/978-3-642-24794-1_5. [Online]. Available: https://doi.org/10.1007/978-3-642-24794-1_5 (visited on 04/14/2023).

[42] N. Noy, "Ontology Development 101: A Guide to Creating Your First Ontology," 2001. [Online]. Available: https://www.semanticscholar.org/paper/Ontology-Development-101%5C%3A-A-Guide-to-Creating-Your-Noy/c15cf32df98969af5eaf85ae3098df6d2180b637 (visited on 04/14/2023).

[43] J. Malone *et al.*, "The Software Ontology (SWO): A resource for reproducibility in biomedical data analysis, curation and digital preservation," *Journal of Biomedical Semantics*, vol. 5, no. 1, p. 25, Jun. 2014, ISSN: 2041-1480. DOI: 10.1186/2041-1480-5-25. [Online]. Available: https://doi.org/10.1186/2041-1480-5-25 (visited on 04/14/2023).

[44] V. Bosilj Vuksic, L. Brkic, and K. Tomicic-Pupek, "Understanding the Success Factors in Adopting Business Process Management Software: Case Studies," en, *Interdisciplinary Description of Complex Systems*, vol. 16, no. 2, pp. 194–215, 2018, ISSN: 1334-4676, 1334-4684. DOI: 10.7906/indecs.16.2.1. [Online]. Available: http://indecs.eu/index.php?s=x%5C&y=2018%5C&p=194-215 (visited on 04/24/2023).

[45] P. Lückmann and C. Feldmann, "Success Factors for Business Process Improvement Projects in Small and Medium Sized Enterprises – Empirical Evidence," en, *Procedia Computer Science*, CENTERIS 2017 - International Conference on ENTERprise Information Systems / ProjMAN 2017 - International Conference on Project MANagement / HCist 2017 - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN/HCist 2017, vol. 121, pp. 439–445, Jan. 2017, ISSN: 1877-0509. DOI: 10.1016/j.procs.2017.11.059. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917322524 (visited on 04/24/2023).

[46] A. Tarhan and O. Turetken, "Critical Success Factors of Business Process Management: Investigating the Coverage of Business Process (Management) Maturity Models," Zenodo, Tech. Rep., Dec. 2016. [Online]. Available: https://zenodo.org/record/3604451 (visited on 04/24/2023).

[47] S. Malviya, M. Vierhauser, J. Cleland-Huang, and S. Ghaisas, "What Questions do Requirements Engineers Ask?" In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, ISSN: 2332-6441, Sep. 2017, pp. 100–109. DOI: 10.1109/RE.2017.76.

[48] D. Zowghi and C. Coulin, "Requirements elicitation: A survey of techniques, approaches, and tools," in *Engineering and Managing Software Requirements*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 19–46, ISBN: 978-3-540-28244-0. DOI: 10.1007/3-540-28244-0_2. [Online]. Available: https://doi.org/10.1007/3-540-28244-0_2.

[49] T. Iqbal and M. Suaib, "Requirement elicitation technique: -a review paper," *International Journal of Computer & Mathematical Sciences*, vol. 3, p. 1, Dec. 2014.

[50] *ISO/IEC TS 25011:2017*, en. [Online]. Available: https://www.iso.org/standard/35735.html (visited on 05/29/2023).

[51] G. Antoniou and F. van Harmelen, "Web ontology language: Owl," in *Handbook on Ontologies*, S. Staab and R. Studer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 67–92, ISBN: 978-3-540-24750-0. DOI: 10.1007/978-3-540-24750-0_4. [Online]. Available: https://doi.org/10.1007/978-3-540-24750-0_4.

[52] D. Kalibatiene and O. Vasilecas, "Survey on Ontology Languages," en, in *Perspectives in Business Informatics Research*, J. Grabis and M. Kirikova, Eds., ser. Lecture Notes in Business Information Processing, Berlin, Heidelberg: Springer, 2011, pp. 124–141, ISBN: 9783642245114. DOI: 10.1007/978-3-642-24511-4_10.

[53] M. A. Musen, "The protégé project: A look back and a look forward," *AI Matters*, vol. 1, no. 4, pp. 4–12, 2015. DOI: 10.1145/2757001.2757003. [Online]. Available: https://doi.org/10.1145/2757001.2757003.

[54] S. Peldszus, J. Bürger, T. Kehrer, and J. Jürjens, "Ontology-driven evolution of software security," en, *Data & Knowledge Engineering*, vol. 134, p. 101 907, Jul. 2021, ISSN: 0169-023X. DOI: 10.1016/j.datak.2021.101907. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169023X21000343 (visited on 04/21/2023).

[55] M. J. Blas, S. Gonnet, and H. Leone, "An ontology to document a quality scheme specification of a software product," en, *Expert Systems*, vol. 34, no. 5, e12213, Oct. 2017, ISSN: 02664720. DOI: 10.1111/exsy.12213. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/exsy.12213 (visited on 04/21/2023).

[56] A. Konys and Z. Drążek, "Ontology Learning Approaches to Provide Domain-Specific Knowledge Base," en, *Procedia Computer Science*, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020, vol. 176, pp. 3324–3334, Jan. 2020, ISSN: 1877-0509. DOI: `10.1016/j.procs.2020.09.065`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1877050920319608` (visited on 04/12/2023).

[57] A. Konys, "Knowledge systematization for ontology learning methods," *Procedia Computer Science*, vol. 126, pp. 2194–2207, 2018, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia, ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2018.07.229`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1877050918312043`.

[58] G. Kim and Y. Suhh, "Ontology-based semantic matching for business process management," *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, vol. 41, no. 4, pp. 98–118, Nov. 2010, ISSN: 0095-0033. DOI: `10.1145/1899639.1899645`. [Online]. Available: `https://doi.org/10.1145/1899639.1899645` (visited on 04/24/2023).

[59] A. Kayed, N. Hirzalla, A. A. Samhan, and M. Alfayoumi, "Towards an Ontology for Software Product Quality Attributes," in *2009 Fourth International Conference on Internet and Web Applications and Services*, May 2009, pp. 200–204. DOI: `10.1109/ICIW.2009.36`.

[60] S. Motogna, I. Ciuciu, C. Serban, and A. Vescan, "Improving Software Quality Using an Ontology-Based Approach," en, in *On the Move to Meaningful Internet Systems: OTM 2015 Workshops*, I. Ciuciu *et al.*, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2015, pp. 456–465, ISBN: 9783319261386. DOI: `10.1007/978-3-319-26138-6_49`.

[61] N. Bajnaid, "An ontological approach to model software quality assurance knowledge domain," 2013. [Online]. Available: `https://www.semanticscholar.org/paper/An-ontological-approach-to-model-software-quality-Bajnaid/bf3f4cd2e94075db65bf51d3cf7c24afac1d023e` (visited on 04/21/2023).

[62] I. Lera, P. P. Sancho, C. Juiz, R. Puigjaner, J. Zottl, and G. Haring, "Performance assessment of intelligent distributed systems through software performance ontology engineering (SPOE)," en, *Software Quality Journal*, vol. 15, no. 1, pp. 53–67, Mar. 2007, ISSN: 1573-1367. DOI: `10.1007/s11219-006-9004-1`. [Online]. Available: `https://doi.org/10.1007/s11219-006-9004-1` (visited on 04/21/2023).

[63] I. Lera, C. Juiz, and R. Puigjaner, "Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems," en, *Science of Computer Programming*, Special Issue on Quality system and software architectures, vol. 61, no. 1, pp. 27–37, Jun. 2006, ISSN: 0167-6423. DOI: `10.1016/j.scico.2005.11.003`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S016764230600013X` (visited on 04/21/2023).

[64] V. Cortellessa, "How far are we from the definition of a common software performance ontology?" In *Proceedings of the 5th international workshop on Software and performance*, ser. WOSP '05, New York, NY, USA: Association for Computing Machinery, Jul. 2005, pp. 195–204, ISBN: 9781595930873. DOI: `10.1145/1071021.1071044`. [Online]. Available: `https://doi.org/10.1145/1071021.1071044` (visited on 04/21/2023).

[65] A. Grimán., L. Chávez., M. Pérez., L. Mendoza., and K. Domínguez., "Towards a maintainability evaluation in software architectures," in *Proceedings of the Eighth International Conference on Enterprise Information Systems - Volume 5: ICEIS*, INSTICC, SciTePress, 2006, pp. 555–558, ISBN: 978-972-8865-43-6. DOI: `10.5220/0002462005550558`.

[66] M. Alenezi, H. A. Basit, F. I. Khan, and M. A. Beg, "A Comparison Study of Available Sofware Security Ontologies," in *Proceedings of the Evaluation and Assessment in Software Engineering*, ser. EASE '20, New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 499–504, ISBN: 9781450377317. DOI: `10.1145/3383219.3383292`. [Online]. Available: `https://doi.org/10.1145/3383219.3383292` (visited on 04/21/2023).

[67] F. de Franco Rosa, M. Jino, and R. Bonacin, "Towards an Ontology of Security Assessment: A Core Model Proposal," en, in *Information Technology - New Generations*, S. Latifi, Ed., ser. Advances in Intelligent Systems and Computing, Cham: Springer International Publishing, 2018, pp. 75–80, ISBN: 9783319770284. DOI: `10.1007/978-3-319-77028-4_12`.

[68] J. Zhou, E. Niemelä, A. Evesti, A. Immonen, and P. Savolainen, "OntoArch Approach for Reliability-Aware Software Architecture Development," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*, ISSN: 0730-3157, Jul. 2008, pp. 1228–1233. DOI: 10.1109/COMPSAC.2008.91.

[69] N. Baliyan, A. Verma, N. Baliyan, and A. Verma, *Recent Advances in the Evaluation of Ontology Quality*, English, ch., Jan. 2001. [Online]. Available: https://www.igi-global.com/gateway/chapter/www.igi-global.com/gateway/chapter/215071 (visited on 07/14/2023).

[70] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Journal of Web Semantics*, vol. 5, no. 2, pp. 51–53, 2007, Software Engineering and the Semantic Web, ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2007.03.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570826807000169.

[71] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa, "OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation," ng, *International Journal on Semantic Web and Information Systems*, vol. 10, no. 2, pp. 7–34, Apr. 2014, ISSN: 1552-6283, 1552-6291. DOI: 10.4018/ijswis.2014040102. [Online]. Available: https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/ijswis.2014040102 (visited on 07/14/2023).

[72] J. Bandeira, I. I. Bittencourt, P. Espinheira, and S. Isotani, "FOCA: A Methodology for Ontology Evaluation," 2016. (visited on 07/14/2023).

[73] H. Nakayama, T. Kubo, J. Kamura, Y. Taniguchi, and X. Liang, *doccano: Text annotation tool for human*, Software available from https://github.com/doccano/doccano, 2018. [Online]. Available: https://github.com/doccano/doccano.

[74] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," *CoRR*, vol. abs/1909.11942, 2019. arXiv: 1909.11942. [Online]. Available: http://arxiv.org/abs/1909.11942.

[75] G. Malik, M. Cevik, S. Bera, S. Yildirim, D. Parikh, and A. Basar, "Software requirement specific entity extraction using transformer models," en, *Proceedings of the Canadian Conference on Artificial Intelligence*, May 2022. DOI: 10.21428/594757db.9e433d7c. [Online]. Available: https://caiac.pubpub.org/pub/rl8j1yb1 (visited on 06/16/2023).

[76] [Online]. Available: https://ec.europa.eu/research/participants/docs/h2020-funding-guide/cross-cutting-issues/open-access-data-management/data-management_en.htm.

[77] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 532–538, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_565.

[78] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *CoRR*, vol. abs/1508.07909, 2015. arXiv: 1508.07909. [Online]. Available: http://arxiv.org/abs/1508.07909.

[79] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. arXiv: 1711.05101. [Online]. Available: http://arxiv.org/abs/1711.05101.

[80] A. Gupta, *A comprehensive guide on optimizers in deep learning*, Oct. 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/.

[81] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "Software-Specific Named Entity Recognition in Software Engineering Social Content," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, Mar. 2016, pp. 90–101. DOI: 10.1109/SANER.2016.10.

[82] N. Heist, S. Hertling, D. Ringler, and H. Paulheim, *Knowledge Graphs on the Web – an Overview*, en, arXiv:2003.00719 [cs], Mar. 2020. [Online]. Available: http://arxiv.org/abs/2003.00719 (visited on 11/22/2022).

[83] E. Huaman, *Steps to Knowledge Graphs Quality Assessment*, arXiv:2208.07779 [cs], Aug. 2022. DOI: 10.48550/arXiv.2208.07779. [Online]. Available: http://arxiv.org/abs/2208.07779 (visited on 08/01/2023).

[84] H. Chen, G. Cao, J. Chen, and J. Ding, “A Practical Framework for Evaluating the Quality of Knowledge Graph,” en, in *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding*, X. Zhu, B. Qin, X. Zhu, M. Liu, and L. Qian, Eds., ser. Communications in Computer and Information Science, Singapore: Springer, 2019, pp. 111–122, ISBN: 9789811519567. DOI: 10.1007/978-981-15-1956-7_10.

[85] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, “HermiT: An OWL 2 reasoner,” *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, May 2014. DOI: 10.1007/s10817-014-9305-1. [Online]. Available: https://doi.org/10.1007/s10817-014-9305-1.

[86] B. Rodrawangpai and W. Daungjaiboon, “Improving text classification with transformers and layer normalization,” *Machine Learning with Applications*, vol. 10, p. 100 403, 2022, ISSN: 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2022.100403. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827022000792.

[87] M. Fernández-López, A. Gomez-Perez, and N. Juristo, “Methontology: From ontological art towards ontological engineering,” *Engineering Workshop on Ontological Engineering (AAAI97)*, Mar. 1997.

[88] R. Rocha *et al.*, “DKDOnto: An Ontology to Support Software Development with Distributed Teams,” *Procedia Computer Science*, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia, vol. 126, pp. 373–382, Jan. 2018, ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.07.271. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091831247X (visited on 08/19/2023).

[89] P. Wongthongtham, N. Kasisopha, E. Chang, and T. Dillon, “A Software Engineering Ontology as Software Engineering Knowledge Representation,” in *2008 Third International Conference on Convergence and Hybrid Information Technology*, vol. 2, Nov. 2008, pp. 668–675. DOI: 10.1109/ICCIT.2008.301.

[90] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville, “Development of a Software Engineering Ontology for Multisite Software Development,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 8, pp. 1205–1217, Aug. 2009, ISSN: 1558-2191. DOI: 10.1109/TKDE.2008.209.

[91] A. Gómez-Pérez, “Ontology evaluation,” in *Handbook on Ontologies*, S. Staab and R. Studer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 251–273, ISBN: 978-3-540-24750-0. DOI: 10.1007/978-3-540-24750-0_13. [Online]. Available: https://doi.org/10.1007/978-3-540-24750-0_13.

[92] J. Raad. and C. Cruz., “A survey on ontology evaluation methods,” in *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015) - KEOD*, INSTICC, SciTePress, 2015, pp. 179–186, ISBN: 978-989-758-158-8. DOI: 10.5220/0005591001790186.

[93] J. Wang, X. Shi, L. Cheng, K. Zhang, and Y. Shi, “Softkg: Building a software development knowledge graph through wikipedia taxonomy,” in *2020 IEEE World Congress on Services (SERVICES)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2020, pp. 151–156. DOI: 10.1109/SERVICES48979.2020.00042. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SERVICES48979.2020.00042.

[94] N. Stylianou, D. Vlachava, I. Konstantinidis, N. Bassiliades, and V. Peristeras, “Doc2KG: Transforming Document Repositories to Knowledge Graphs,” ng, *International Journal on Semantic Web and Information Systems*, vol. 18, no. 1, pp. 1–20, Feb. 2022, ISSN: 1552-6283, 1552-6291. DOI: 10.4018/IJSWIS.295552. [Online]. Available: https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/IJSWIS.295552 (visited on 08/11/2023).

[95] B. Min, S. Shi, R. Grishman, and C.-Y. Lin, “Towards Large-Scale Unsupervised Relation Extraction from the Web:” ng, *International Journal on Semantic Web and Information Systems*, vol. 8, no. 3, pp. 1–23, Jul. 2012, ISSN: 1552-6283, 1552-6291. DOI: 10.4018/jswis.2012070101. [Online]. Available: https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jswis.2012070101 (visited on 08/19/2023).

[96] B. Rink and S. Harabagiu, “Utd: Classifying semantic relations by combining lexical and semantic resources,” in *Proceedings of the 5th international workshop on semantic evaluation*, 2010, pp. 256–259.

[97] N. Kambhatla, "Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations," in *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions*, ser. ACLdemo '04, Barcelona, Spain: Association for Computational Linguistics, 2004, 22–es. DOI: 10.3115/1219044.1219066. [Online]. Available: https://doi.org/10.3115/1219044.1219066.

[98] Z. GuoDong, S. Jian, Z. Jie, and Z. Min, "Exploring various knowledge in relation extraction," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ser. ACL '05, Ann Arbor, Michigan: Association for Computational Linguistics, 2005, pp. 427–434. DOI: 10.3115/1219840.1219893. [Online]. Available: https://doi.org/10.3115/1219840.1219893.

[99] R. C. Bunescu and R. J. Mooney, "A shortest path dependency kernel for relation extraction," in *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, ser. HLT '05, Vancouver, British Columbia, Canada: Association for Computational Linguistics, 2005, pp. 724–731. DOI: 10.3115/1220575.1220666. [Online]. Available: https://doi.org/10.3115/1220575.1220666.

[100] A. Culotta and J. Sorensen, "Dependency tree kernels for relation extraction," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ser. ACL '04, Barcelona, Spain: Association for Computational Linguistics, 2004, 423–es. DOI: 10.3115/1218955.1219009. [Online]. Available: https://doi.org/10.3115/1218955.1219009.

[101] M. Zhang, G. Zhou, and A. Aw, "Exploring syntactic structured features over parse trees for relation extraction using kernel methods," *Information Processing Management*, vol. 44, no. 2, pp. 687–701, 2008, Evaluating Exploratory Search Systems Digital Libraries in the Context of Users' Broader Activities, ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2007.07.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457307001628.

[102] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," *CoRR*, vol. abs/1601.00770, 2016. arXiv: 1601.00770. [Online]. Available: http://arxiv.org/abs/1601.00770.

[103] X. Yang, Z. Yu, Y. Guo, J. Bian, and Y. Wu, "Clinical relation extraction using transformer-based models," *CoRR*, vol. abs/2107.08957, 2021. arXiv: 2107.08957. [Online]. Available: https://arxiv.org/abs/2107.08957.

[104] M. Dong, C. Pan, and Z. Luo, "MapRE: An effective semantic mapping approach for low-resource relation extraction," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2694–2704. DOI: 10.18653/v1/2021.emnlp-main.212. [Online]. Available: https://aclanthology.org/2021.emnlp-main.212.

[105] X. Wang *et al.*, "Knowledge graph quality control: A survey," en, *Fundamental Research*, vol. 1, no. 5, pp. 607–626, Sep. 2021, ISSN: 2667-3258. DOI: 10.1016/j.fmre.2021.09.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667325821001655 (visited on 07/21/2023).

[106] S. Seo, H. Cheon, H. Kim, and D. Hyun, *Structural Quality Metrics to Evaluate Knowledge Graphs*, arXiv:2211.10011 [cs], Dec. 2022. DOI: 10.48550/arXiv.2211.10011. [Online]. Available: http://arxiv.org/abs/2211.10011 (visited on 07/21/2023).

[107] X. Dai and H. Adel, "An analysis of simple data augmentation for named entity recognition," *CoRR*, vol. abs/2010.11683, 2020. arXiv: 2010.11683. [Online]. Available: https://arxiv.org/abs/2010.11683.

Appendix A

Competency Questions

The standard competency questions asked during interviews during the ontology development process (Note that follow-up questions were also asked but are not listed here):

1. When provided with a new software system to evaluate and after reading the guidelines of the contract, what are the next steps that need to be taken to start the project? (Can you outline your approach/workflow)
2. Would it be useful to understand the business process the software system is meant to support before analyzing the system? If yes, why would this help?
3. What are common practices/ examples of business processes that a software system supports? And how are they implemented within the system?
4. Is there external information that would be useful to know when applying the SIG models to measure a system?
5. Would you actively search to see if the system meets ISO standards outside the SIG model?
6. Outside of ISO standards, are there any other standards or protocols that SIG models follow to evaluate software systems?
7. What are the typical stakeholders of the software system that you would usually like to have background information about? Users, developers, administrators?
8. Here is a list of potential points of background information that could be important, are they important to evaluate a software system and why?:
    (a) Purpose and goal of the software
    (b) The integration and behaviour of a software system to connected systems
    (c) Security features and policies
    (d) Performance metrics and/or other metrics included in articles, documentation, manuals, etc.
    (e) Testing and quality assurance processes
    (f) Maintenance and support processes
    (g) Risk management
    (h) Role of system users and permissions
    (i) Collection and storage of system data
    (j) Collection, storage, and analysis of user feedback and system usage data
    (k) Information about monitoring the system
    (l) The structure and architecture of the system
    (m) Organizations, parties, or stakeholders involved with the system
    (n) Information about the system life cycle
    (o) Hardware to host the system
    (p) Processes to run the system (i.e. process of providing them to users)
    (q) Procedures/ instructions (i.e. operation instructions)
    (r) Technical requirements: Licenses, Platform/OS, Dependencies, additional software, technology stack, CI/CD or deployment
    (s) Development practices
    (t) Software's compliance and regulatory requirements
    (u) Documentation and training materials

Appendix B

Dataset Construction Search Queries

The following terms were used in search queries when collecting data for each scalability entity in the Software Context Dataset:

- **Transaction_Scalability:**transaction scalability, throughput scalability, concurrent transaction handling, resource scaling strategy, elastic resource provisioning, load scalability, load balancing techniques, component load scalability, on-demand resource provisioning, resource elasticity, auto-scaling strategies, measuring transactional capacity, capacity measurement techniques, transactional workload measurement, scalability performance metrics, real-time scalability monitoring
- **Development_Scalability:** scalable software development, technology stack, system architecture documentation, source code documentation, development planning process, agile development planning, software development lifecycle planning, technical debt, system automation, deployment automation, continuous integration, CI CD (Continuous Integration/Continuous Deployment), automation issue management, quality assurance, testing and quality control, code review, coding standards, coding guidelines, architectural and design constraints
- **Data_Scalability:** data scalability, Scalable data management, Scaling data infrastructure, Managing increasing data volume, Data storage capacity, database volume, Handling large databases, database volume monitoring, Capacity monitoring for databases, Database volume tracking and alerts, Defining database volume limits, Database capacity thresholds,performance-intensive data types, data filtering, Filtering large datasets efficiently, Scalable data filtering methods, In-memory database scalability, scalable data archiving strategy, Scalable operational data management, Scalable reporting data management, Data redundancy and replication scalability, Optimizing data partitioning for scalability, Scalable data distribution techniques

Appendix C

Additional Knowledge Graph Validation and Evaluation Information

C.1 Documents used during Sessions with Experts

C.1.1 Score table provided to experts

Table C.1: Score criteria provided to experts to allocate a value to each quality assessment question.

Score Criteria	Values
Very Poor/ Missing	0
Poor	25
Satisfactory	50
Good	75
Excellent	100

C.1.2 Original Documentation Provided to Experts and KG input

"Sigrid helps you to improve your software by measuring your system's code quality, and then compares the results against a benchmark of 10,000 industry systems to give you concrete advice on areas where you can improve.

Sigrid performs code quality checks that have been designed by the Software Improvement Group, which have been used by thousands of development teams over the past 20 years to help improve their software. SIG's approach is based on the ISO 25010 standard for software quality, and has been accredited to ensure alignment with the standard.

In Scrum, developers and product owners need to collaborate to prioritize the sprint backlog. This can be challenging because priorities can be slightly different, or at least perceived differently.

At the end of 2020, we had over 45,000 customers located in over 100 countries, with millions of users.

Sigrid uses Amazon Aurora features a distributed, fault-tolerant, self-healing storage system that auto-scales up to 128TB per database instance. It delivers high performance and availability with up to 15 low-latency read replicas, point-in-time recovery, continuous backup to Amazon S3."

C.1.3 KG Edges Provided to Experts

A pretty printed version of the triples and the direction for the edges, from the KG, was presented to the experts during the Validation and Evaluation sessions:

```
Sigrid   ————————— Is_A ————————————> Software
SIG ————————— Develops ————————————> Sigrid
Sigrid   ————————— Is_Designed_To ————————————> improve
Sigrid   ————————— Is_Designed_To ————————————> measure
Sigrid   ————————— Is_Designed_To ————————————> compare
Sigrid   ————————— Has_Development_Scalability_Characteristics_From ————————————> code quality
SIG ————————— Is_A ————————————> Company
Software Improvement Group ————————— Belongs_To ————————————> SIG
thousand of development team ————————— Belongs_To ————————————> SIG
Sigrid   ————————— Has_Development_Scalability_Characteristics_From ————————————> code quality check
Sigrid   ————————— Has_Development_Scalability_Characteristics_From ————————————> ISO 25010 standard
     for
Sigrid   ————————— Has_Development_Scalability_Characteristics_From ————————————> software quality
developer   ————————— Belongs_To ————————————> SIG
product owner ————————— Belongs_To ————————————> SIG
Sigrid   ————————— Has_Development_Scalability_Characteristics_From ————————————> Scrum
Sigrid   ————————— Has_Development_Scalability_Characteristics_From ————————————> sprint backlog
45,000 customer  ————————— Is_Userbase_Information_Of ————————————> Sigrid
100 country  ————————— Is_Userbase_Information_Of ————————————> Sigrid
million  of  user   ————————— Is_Userbase_Information_Of ————————————> Sigrid
Amazon Aurora ————————— Is_A ————————————> Software
Amazon S3 ————————— Is_A ————————————> Software
SIG ————————— Develops ————————————> Amazon Aurora
SIG ————————— Develops ————————————> Amazon S3
Amazon Aurora ————————— Has_Data_Scalability_Characteristics_From ————————————> self heal storage
     system
Amazon Aurora ————————— Has_Data_Scalability_Characteristics_From ————————————> 128 TB per database
     instance
Amazon S3 ————————— Has_Data_Scalability_Characteristics_From ————————————> continuous backup
Amazon S3 ————————— Has_Transaction_Scalability_Characteristics_From ————————————> high performance
Amazon S3 ————————— Has_Transaction_Scalability_Characteristics_From ————————————> low latency read
```

C.2 Additional KG Evaluation Results

Table C.2: The original rankings for each goal/ quality dimension (QD) that were provided by the experts.

Goals	Expert 1 Rank	Expert 2 Rank	Expert 3 Rank	Expert 4 Rank	Expert 5 Rank
Accuracy	5	1	3	1	1
Appropriate amount	3	5	2	3	4
Ease of understanding	2	7	3	3	2
Interoperability	6	6	1	3	4
Relevancy	1	3	3	2	1
Believability	7	2	3	1	2
Completeness	4	4	3	1	3