Spring 2023

Extracting a Body of Knowledge as a First Step Towards Defining a United Software Engineering Curriculum Guideline

Anton Kiselev
kiseleva@my.erau.edu

Extracting a Body of Knowledge as a First Step Towards Defining a United Software Engineering Curriculum Guideline

By

Anton Kiselev

A Thesis Submitted to the Faculty of Embry-Riddle Aeronautical University

In Partial Fulfillment of the Requirements for the Degree of Master of

Science in Software Engineering

March 2023

Embry-Riddle Aeronautical University

Daytona Beach, Florida

Extracting a Body of Knowledge as a First Step Towards Defining a United Software Engineering Curriculum Guideline

By

Anton Kiselev

This Thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Omar Ochoa, Department of Electrical Engineering and Computer Science, and has been approved by the members of the Thesis Committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in the partial fulfillment of the requirements for the Degree of Master of Science in Software Engineering.

ACKNOWLEDGEMENTS

I want to express my most profound gratefulness to my parents, who have been my constant source of love, encouragement, and support throughout my academic journey. Their undying faith in me and unending sacrifices have greatly aided my educational aspirations.

I also want to express my gratitude to my close friend Tyler Procko, whose suggestions and guidance have been crucial in designing this study. I sincerely appreciated his unwavering encouragement and support.

I'd also want to thank my professor, Dr. Omar Ochoa, for his advice, encouragement, and thoughtful criticism. I am really grateful for his assistance in creating this research since his knowledge and insight were of immeasurable use.

Last but not least, I would like to convey my sincere gratitude to Dr. Massood Towhidnejad, who served as my master's advisor at the university, for his important advice, assistance, and encouragement throughout my graduate studies. I also want to express my gratitude to Dr. Keith Garfield for joining the thesis committee and contributing insightful comments on my thesis.

Thank you, Mom and Dad, Tyler, Dr. Ochoa, Dr. Towhidnejad, and Dr. Garfield, for your unwavering support, guidance, and encouragement throughout my academic journey.

ABSTRACT

In general, the computing field is a rapidly changing environment, and as such, software engineering education must be able to adjust quickly to new needs. Industry adapts to technologies as fast as it can, but the critical issue is a need for recent graduates with the necessary expertise and knowledge of new trends, technologies, and practical experience. The industries that employ graduates of computing degree programs aim to hire those who are familiar with the latest technical traits, tools, and methodologies to meet these needs, and the software engineering curriculum needs to respond quickly to these needs. Still, unfortunately, software engineering curriculums cannot change and adopt new technologies fast. Modifying the curriculum to serve industry needs better is a long and tedious process in an academic setting. It is essential to give software engineers top-notch education and training to make sure they have the information and abilities needed to succeed in their careers. In addition, there are multiple computing curriculum recommendations endorsed by computing professional organizations that provide guidelines for curriculum design. The work proposed for this research plans to develop a method of extracting a body of knowledge and generating an ontology using Natural Language Processing algorithms. This will automate the process of extracting information from curriculum guidelines and models and storing that information in one unified ontology. It is then envisioned that the resulting ontology will be used in future research to assist in creating or validating a Software Engineering curriculum to ensure that all knowledge areas are covered and that the outcomes match the established guidelines and models. This automated extracting a body of knowledge process is the first and fundamental step in defining the United Software engineering Curriculum Guideline.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

Nowadays technologies evolve dynamically and require special knowledge and expertise. Industry adapts to technologies as fast as it can, but the main issue is a lack of recent graduates with the necessary expertise and knowledge of the new trends, technologies and practical experience. The industries that employ graduates of computing degree programs aim to hire those who are familiar with the latest technical traits, tools, and methodologies [1]. To meet these needs, the software engineering curriculum needs to respond quickly to these needs, but unfortunately, software engineering curriculums cannot change and adopt new technologies fast [2].

1.1. Motivation

Modifications to the curriculum, to better serve the needs of industry; in an academic setting is a long and tedious process [3]. By the time a curriculum modification is proposed, submitted through the approval process, and accepted, the industry needs may have changed [4]. To further complicate this process, multiple computing curriculum recommendations have been endorsed by computing professional organizations that provide guidelines for curriculum design. To solve this problem, the main need is to create a United Software Engineering Curriculum Guideline (USECG), which will allow curriculum developers to identify the curriculum needs in order to graduate students with enough knowledge and experience to solve real-world problems. A USECG model will combine the multiple recommendations that other models capture into one model that can be used to generate a curriculum.

1.2. Main Objective

The primary research objective of this project is to define a process to automatically extract information from guidelines and models, which is a crucial step in defining USECG. For that, Natural Language Processing (NLP) will be applied to analyze guidelines and models to find correlations between various knowledge areas. Results will be verified by manual analysis. Using the extracted information, we will create and populate a unified model based on ontology. It is then envisioned that the resulting ontology will be used to assist in creating a Software Engineering curriculum to ensure that all knowledge areas are covered and that the outcomes match the established guidelines and models.

2. Background

Some historical background that is pertinent to the subject at hand is provided in this section. To create a foundation of shared understanding, the reader must have at least a basic comprehension of the concepts of SWEBOK and other guidelines, ontology, and natural language processing. In the context of this thesis, this section provides compelling descriptions of those subjects.

2.1. SWEBOK

SWEBOK is a major milestone in establishing software engineering as a recognized engineering discipline. SWEBOK is currently in its third revision, titled SWEBOK V3. This version is available as a concise guide of approximately 335 pages, as the authors state that it is not feasible to "present the entire body of knowledge for software engineering… that has been developed over more than four decades" [5]. This guide is organized into 15 distinct Knowledge Areas (KAs). Please see Table 2.1. Each KA section begins with a graphic of the expanded KA taxonomy. See Figure 2.1 [5]. Each KA section contains a body of text, composed of sub-sections describing each child node of the KA taxonomy.

Table 2.1 The 15 KAs defined in the SWEBOK

01 – Software Requirements	**06** – Software Configuration Management	**11** – Software Engineering Professional Practice
02 – Software Design	**07** – Software Engineering Management	**12** – Software Engineering Economics
03 – Software Construction	**08** – Software Engineering Process	**13** – Computing Foundations
04 – Software Testing	**09** – Software Engineering Models and Methods	**14** – Mathematical Foundations
05 – Software Maintenance	**10** – Software Quality	**15** – Engineering Foundations

Roughly speaking, the SWEBOK document is a tree-like structure, with specific nested sections being subsumed by more general ones. The SWEBOK guide distinguishes the levels of the KA taxonomies, with the second level of each taxonomy being labelled as *Subareas* and the third (and sometimes fourth) levels *Topics* and *Subtopics*, respectively. For example, *Software Quality* is a KA, *Software Quality Fundamentals* is a Subarea and *Value and Costs of Quality* is a Topic.

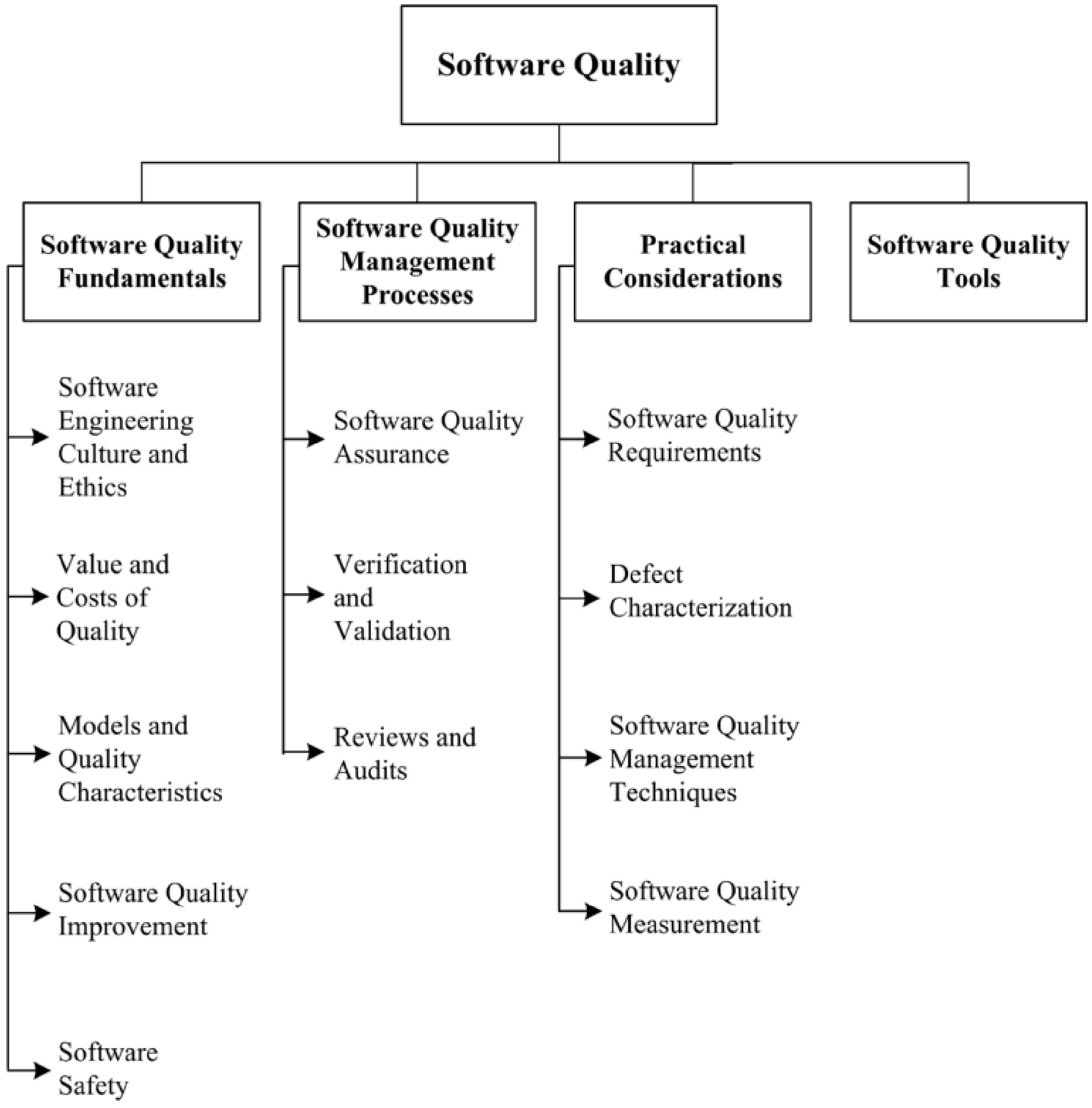

Software Quality
Software Quality Fundamentals
Software Quality Management Processes
Practical Considerations
Software Quality Tools
Software Engineering Culture and Ethics
Value and Costs of Quality
Models and Quality Characteristics
Software Quality Improvement
Software Safety
Software Quality Assurance
Verification and Validation
Reviews and Audits
Software Quality Requirements
Defect Characterization
Software Quality Management Techniques
Software Quality Measurement

Figure 2.1 Taxonomy of the Software Quality KA defined in SWEBOK V3 [5]

2.2. SWECOM

SWECOM is a competency model that represents a set of competencies that a software engineering professional should possess and could potentially be used by educators to develop a software engineering curriculum that meets the needs of the industry [6]. SWECOM is given as a framework that may be customized to meet the needs of companies, programs, and projects. SWECOM is based on the SWEBOK guide. This competency model's skill areas comprise competencies that are broken down into activities rather than job roles because job roles are typically reliant on the organizational context in which work activities take place [6].

2.3. SE2014

SE2014 provides guidance to academic institutions and accreditation agencies about what should constitute an undergraduate software engineering education. The Software Engineering Education Knowledge (SEEK) is a set of knowledge domains that is defined by SE2014 which are based on SWEBOK KA's [7]. The smallest degree of knowledge that may be acquired is represented by the division of each knowledge area into a number of units. Then each unit is further subdivided into a collection of subjects. To ensure consistency with other curriculum reports, the SEEK uses lecture hours, abbreviated to hours, to quantify instructional time; this measure is generally understandable in (and transferable to) cross-cultural contexts. Thus, an hour corresponds to the time required to present the material in a traditional lecture-oriented format; it does not include any additional work that is associated with a lecture [7].

2.4. ABET

ABET is the gold standard for accreditation [8]. ABET does not describe knowledge areas as SWEBOK, or provide a curriculum guideline as SE 2014, but consists of different accreditation criteria both for Baccalaureate and Masters level programs. Specifically, we are looking at criteria for Software programs and similarly named engineering programs. Those criteria are:

- Curriculum – The curriculum must cover a wide range of engineering and computer science courses, as suggested by the program's title and aims. Computing fundamentals, software design and development, requirements analysis, security, verification, and validation, as well as software engineering processes and tools suitable for the creation of complex software systems, must all be covered in the curriculum. Discrete mathematics, probability, and statistics, with applications suitable for software engineering, must also be covered [8].
- Faculty - The course must show that the professors instructing the fundamentals of software engineering are knowledgeable in the field's professional practice and keep up with developments in their fields of professional or scholarly expertise [8].

2.5. Machine Learning and Natural Language Processing

Machine Learning (ML) originates as early as 1950, and its primary definition is a "field of study that gives computers the ability to learn without being explicitly programmed" [9]. It is advantageous for the human operator to actually let the computer "do the work" when trained ML can produce results from unseen input. NLP, a branch of ML, is a field of computer science that focuses on methods for modeling, comprehending, and interpreting human language [10]. Several modern applications that deal with human vocabulary depend on NLP. A large number of software programs that people use on a daily basis include email clients, voice-activated assistants like Apple's Siri and Amazon's Alexa, search engines like Google and Bing, tools for correcting spelling and grammar like Grammarly, and many others [11]. NLP is commonly employed in tasks involving:

- Language modeling - Predicting the following word in a sentence by looking at the history of the previous words
- Information extraction - obtaining pertinent data from text, such as dates or locations.
- Information retrieval - Using a user query to search across a large collection of documents.
- Text summarization - Retaining the meaning while succinctly summarizing a long text.
- Machine translation - a text's translation from one language to another.
- Topic modeling - identifying a text collection's thematic organization.

Information Extraction (IE) is a further function of NLP. Relationship extraction (RE), a part of IE, aims to find and extract relationships between things in the text. For example, from the sentence "John Smith works for Google", RE will extract the relationship type, such as "works for". A variety of NLP techniques, such as named entity recognition (NER), dependency parsing, or semantic role labeling, can be used to extract relationships between things from text [12].
REBEL, which stands for Relationship Extraction By End-to-End Language generation, is an extremely capable IE model for 2021 [13]. The encoder-decoder transformer from Facebook AI, as well as the bidirectional and auto-regressive transformers, constitute the foundation of REBEL (BART) [14]. In the past, NER and Relation Classification (RC) algorithms were used to extract relations from text bodies. However, both methods had drawbacks [13]. The technique is known as End-to-End Relation Extraction, or simply Relation Extraction (RE), in more recent approaches (like REBEL), which do both jobs concurrently. REBEL is a cutting-edge RE model that may generate effective KGs by pre-training BART-large with a specifically curated dataset made up of Wikipedia abstracts and Wikidata entities and relations [13].

A popular Python library for NLP is called spaCy. Tokenization, part-of-speech tagging, named entity recognition, dependency parsing, and other essential NLP operations are all provided by spaCy. NLP is viewed by spaCy as a pipeline process where each step can be finished using a component. REBEL is a spaCy component for the relationship extraction pipeline phase [15].

2.6. Ontology

Ontology is a formal, explicit explanation of the relevant objects and relationships that are believed to exist within a given area of knowledge, along with the names we use to refer to them and our shared understandings of their meanings and attributes. Concepts, qualities and attributes, restrictions on properties and attributes, and, frequently but not always, individuals are included in this description. There are ontologies that represent different facets of reality, such as those related to business, finance, healthcare, history, engineering, mathematics, natural language, and so forth [16].

An ontology is a serialized entity or object that encapsulates information specifically in the field of computer science [17]. An ontology, according to Gruber (1992), is a "explicit statement of conceptualization," and according to Borst (1997), it is a "formal specification of a shared conceptualization" [18, 19]. An ontology, according to Arp, Smith, and Spear (2015), is a particular artifact that represents a certain feature of reality, including its entities and the relationships that exist between them [20]. An ontology is built on a hierarchy of concepts, or taxonomy, and uses a hierarchy of relations to link these concepts together such that a directed graph of related concepts can be created. Ontologies can be relatively concise, with only a few concepts defined, or they can be quite verbose, with strict logic and formality [17, 21, 22].

2.7. Graph Database

A graph database is a particular kind of database that stores and represents data using graph topologies containing nodes, edges, and characteristics [23]. Nodes, edges, and attributes make up an Labelled Property Graph (LPG). Edges represent the relationships between nodes, whereas properties are the traits or qualities of both nodes and edges. Graph databases are particularly helpful for maintaining data with intricate dependencies and linkages, which makes them suitable for use in social networks, recommendation systems, fraud detection, and other areas.

RE necessarily results in a graph of related (or “linked”) entities. As an example, WordNet is a large lexical database grouping English nouns, verbs, adjectives and adverbs into 117,000 sets of synonyms linked by semantic relations [24]. WordNet is a project with its roots in the mid-1980s, and is curated manually.

Automatic NLP entity linkage will be done in this project. Hence, a suitable graph storage platform is required. A graph database management system is called Neo4j. Large and complicated datasets represented as Labelled Property Graphs can be stored, managed, and queried using this open-source, NoSQL database (LPGs). Nodes, edges, and attributes make up an LPG. Edges represent the relationships between nodes, whereas properties are the traits or qualities of both nodes and edges. Nodes represent entities or items, such as people or things. Because it was simple to convert REBEL triplets into a queryable graph, Neo4j was initially used in this work. However, as will be discussed in later sections, the Neo4j graphs were converted into an RDF-compliant format and used with SPARQL in Ontotext's GraphDB platform to maintain semantic interoperability with external RDF namespaces.

3. Review of the Relevant Literature

Many studies have revealed a major discrepancy between the output of SE education and the demands of the software industry in terms of potential software engineers and several attempts were made to resolve that issue.

3.1. Software engineering curricula development and evaluation process

One of the attempts to solve the issue with the gap between industry needs and SE was SECDEP [25]. The authors of the article suggest a framework for creating and assessing software engineering curricula based on the IEEE Standard for Software Engineering Body of Knowledge (SWEBOK). The framework, known as SECDEP, offers a complete and systematic method for creating software engineering courses that adhere to industry standards. The article emphasizes the significance of using industry standards as a reference for creating software engineering curriculum, such as SWEBOK. In order for curriculum to remain relevant and useful in training students for the fast changing area of software engineering, it also highlights the necessity for ongoing evaluation and development. The SECDEP framework consists of ten steps shown in Figure 3.1.

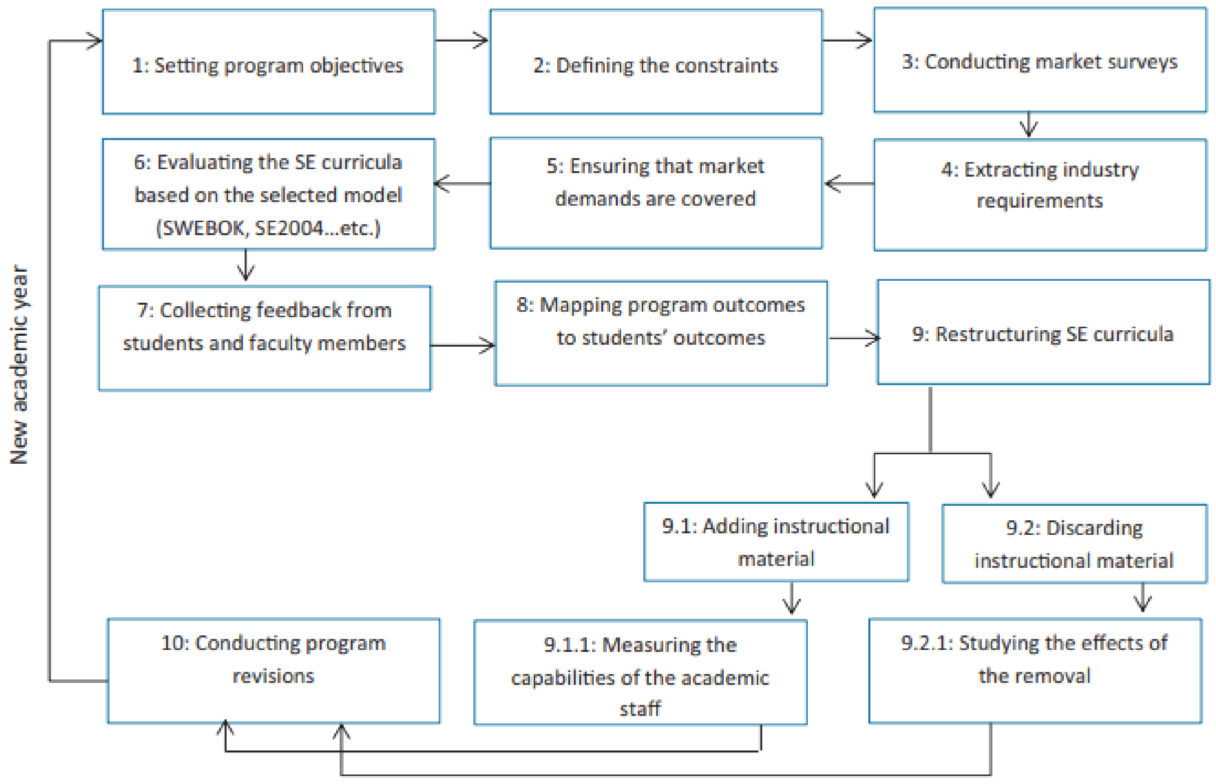

1: Setting program objectives
2: Defining the constraints
3: Conducting market surveys
4: Extracting industry requirements
5: Ensuring that market demands are covered
6: Evaluating the SE curricula based on the selected model (SWEBOK, SE2004...etc.)
7: Collecting feedback from students and faculty members
8: Mapping program outcomes to students' outcomes
9: Restructuring SE curricula
9.1: Adding instructional material
9.2: Discarding instructional material
9.1.1: Measuring the capabilities of the academic staff
9.2.1: Studying the effects of the removal
10: Conducting program revisions
New academic year

Figure 3.1 Steps of SECDEP.

According to authors, by following these steps, SE educators can ensure that their curricula cover the foundation knowledge of software engineering and fulfill the market requirements but there are several disadvantages and challenges in this method [25].

Such challenges and disadvantages are:

- Complexity: To execute the SECDEP framework successfully, a great amount of preparation, money, and knowledge are needed. The structure can be too time-consuming and difficult for some educators.
- Implementation barriers: Certain educational institutions may find it difficult to adopt the SECDEP framework because of financial restrictions, internal regulations, and faculty reluctance to change.

- Limitations of adaptability: The SWEBOK industry standard, on which the SECDEP framework is based, might not be relevant or acceptable for all educational situations or curricula. The framework might not be as adaptable to other approaches of curriculum creation and delivery.
- Limited stakeholder involvement: The analysis, design, development, implementation, and assessment of the curriculum are the main areas of concentration for the SECDEP framework. It might not offer enough chances for stakeholders, such students, industry partners, and academics from different fields, to participate and contribute.
- Evaluation challenges: The SECDEP architecture includes an evaluation step, although determining the efficacy of a curriculum in software engineering can be difficult due to things like the speed at which technology is changing and the variety of employment options available in the industry.

3.2. Industry-Academia Collaboration

In order to bridge the gap between theory and practice in the field of computer science, this work emphasizes the significance of industrial and academic collaboration [4]. Collaboration, according to the authors, is essential to closing the skills gap and ensuring that students are properly equipped for careers in the computer sector.

The work examines various forms of industry-academia collaboration, such as joint programs, research partnerships, and internships. The authors stress the advantages of these partnerships, including giving students practical experience, exposing them to cutting-edge techniques, and promoting the transfer of knowledge between business and academia.

Some of the key disadvantages of IAC method [26]:

- Limited scope: The IAC technique might not fully account for the demands of the industry due to its scope limitations. Usually, just a few industrial partners participate in the cooperation, and they might not fully represent all interests and viewpoints. As a result, there may be a disconnect between the skills being taught in the classroom and what the workforce actually needs.
- Resource-intensive: The IAC technique may be expensive and time-consuming to use in order to create and sustain productive partnerships between business and academics.
- Difficult to sustain: Since that industry demands and priorities are subject to change over time, it may be challenging to maintain the IAC technique over the long run. Keeping up with the most recent industry trends and standards could be difficult.
- Limited involvement: The IAC approach may only receive a limited amount of engagement from industry partners, which might hinder the collaboration's success and its capacity to identify a variety of business requirements.

The writers also talk about the difficulties and impediments to collaboration, like the cultural, financial, and priority contrasts between business and academics. They suggest methods to get beyond these obstacles, including setting up transparent expectations and objectives for collaboration, encouraging partnership and trust, and coordinating incentives and rewards for both business and academia [4].

3.3. SWEBOK Ontology

The development of a SWEBOK ontology began in 2004, with the last publication in 2006 [27, 13, 28]. There are key challenges discussed in [13] of SWEBOK ontology development. Such challenges are:

- Ambiguity and inconsistency in the SWEBOK guide: There is a lot of information in the SWEBOK guide, and some ideas may be explained in several ways, which might cause confusion and inconsistencies. The content of the guide needed to be carefully analyzed and interpreted in order to be resolved.
- Complexity of representing relationships between concepts: It takes a thorough knowledge of the domain to express the intricate relationships that software engineering concepts can have with one another in an ontology. The author ensured the authenticity of these linkages by visualizing them using a variety of methodologies, such as UML diagrams.
- Balancing completeness and conciseness: The SWEBOK guide is filled with a ton of information, and trying to capture it all in an ontology could result in a messy system. In order to make the ontology manageable and user-friendly, the author had to strike a balance between comprehensiveness and conciseness.

It should be mentioned that Alain Abran, one of the authors on all three articles, served as the SWEBOK V3 guide's original editor. It was discovered after emailing the authors of these papers—which purport to give the only serialized SWEBOK ontology—that the ontology artifact was not developed and does not exist.

3.4. An Analysis of the Software Engineering Curriculum

The first analysis of SWEBOK, SE2014, SWECOM and university curriculum was conducted by the author in 2020 [29]. In the paper, authors conducted a manual analysis of SWEBOK KAs, SWECOM competencies, SE2014 curriculum guideline and Embry Riddle Aeronautical University (ERAU) curriculum for Software Engineering. The main objective of the research was to find out how well ERAU curriculum or SE2014 curriculum covers SWEBOK KAs.

The first step in this research was to compare SE2014 and SWECOM to identify any similarities or discrepancies. Then it was decided to apply the same approach to the Embry-Riddle Aeronautical University (ERAU) software engineering curriculum and SWECOM.

The course syllabi for classes listed in the ERAU curriculum and SE2014 guidelines were examined in order to identify the topics and activities that are covered in these courses. Thus, the syllabi were used to identify what topics were covered in the course and how many hours were spent on each topic by dividing the topics over the hours the course had.

The second step was to align SWEBOK knowledge areas with classes from curriculums and assign an experience level based on how many hours were spent on each topic.
Because SWECOM does not prescribe the knowledge level or years of experience with these competency levels, the assumption was made that undergraduate students who graduate from a software engineering program should be at the Entry Level Practitioner level.

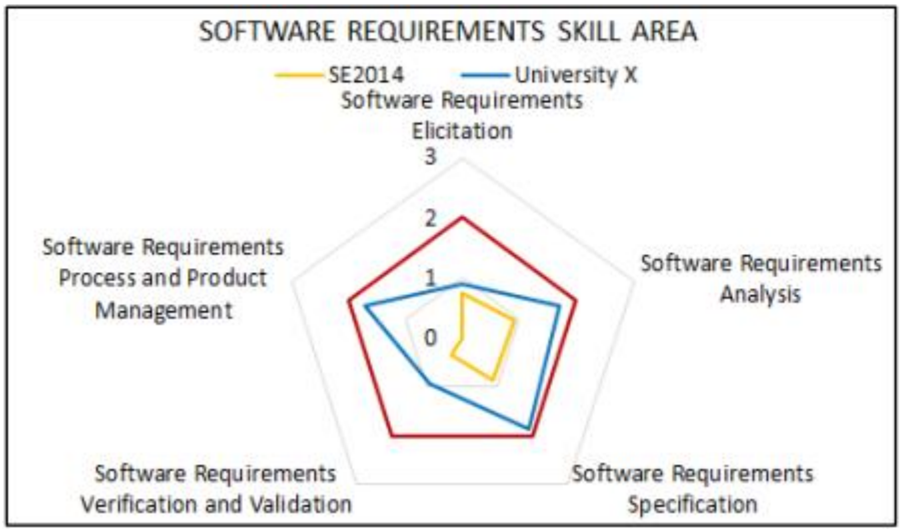


Figure 3.2 Gray-box analysis for Software Requirements

The resulting data were plotted into a radar graph to facilitate the understanding of the data Figure 3.2. The foundation for radar graphs was KAs from SWEBOK, such as requirements, design, etc. Subareas from KA were used as axes for each chart.

This research indicated a gap between the SWECOM level and the resulting outcome of the SE2014 and ERAU curricula for various SWEBOK KAs. That research became a foundation for this thesis and proposing USECG as the ultimate goal.

4. Approach

It was first proposed to manually create the SWEBOK knowledge using the Protégé 5 ontology-editing tool [14]**.** This approach was abandoned, nevertheless, as determining the relationships required deep philosophical reflection or the reuse of existing constructions (e.g., WordNet terms, Wikidata relations). For the automatic digestion and reformation of bodies of unstructured text that undergo consistent research and publication, relationship extraction and knowledge graph building are other themes of NLP. Moreover, for further development and maintaining, the USECG automatic approach is much more favorable than a manual one. See Figure 4.1.

The process described in this section was applied to each KA in SWEBOK individually, after that for all SWEBOK KA's combined and later for a few chapters from SWECOM to demonstrate that the concept works.

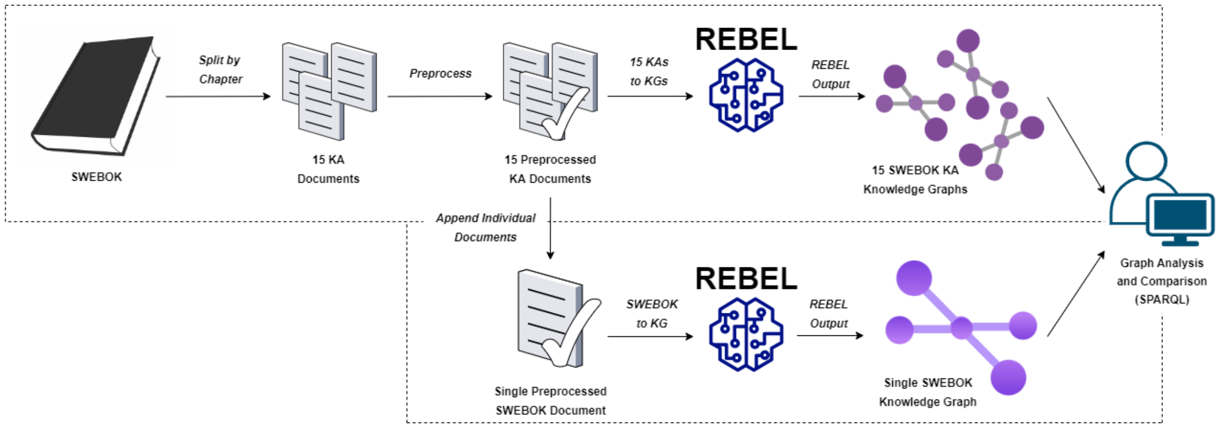

SWEBOK
Split by Chapter
15 KA Documents
Preprocess
15 Preprocessed KA Documents
15 KAs to KGs
REBEL
REBEL Output
15 SWEBOK KA Knowledge Graphs
Graph Analysis and Comparison (SPARQL)
Append Individual Documents
Single Preprocessed SWEBOK Document
SWEBOK to KG
REBEL
REBEL Output
Single SWEBOK Knowledge Graph

Figure 4.1 The proposed method of knowledge graph from SWEBOK Using Rebel

The Google Colab online operating system was used to run all information and relation extraction, knowledge graph construction, and related programming Later was purchased Google Colab Pro due to the fact more computational resources need to process entire SWEBOK. Table 4.1 provides information on the features of the Google Colab and Google Colab Pro environment and the machine that was used. Although the employed spaCy and REBEL code might be run locally on a device, Google Colab and Google Colab Pro enables incredibly efficient and easy package imports into an execution environment, whereas maintaining the intricate import needs on a local Python installation would be challenging. In order to preserve a consistent and reproducible basis, Google Colab and Google Colab Pro was used in this study. Google Colab Pro was purchased because running all SWEBOK chapters combined required more computation resources when running all chapters individually. Important to mention that Google Colab Pro is paid service $9.99 per month but provided more resources.

Table 4.1 Computing Specifications

Google Colab (Free Edition) Environment	**Google Colab Pro Environment**	**Personal Computer Specifications**
• Python 3 Google Compute Engine backend • 12.7 GB RAM • 107.7 GB disk	• Python 3 Google Compute Engine backend • 88.5 GB RAM • 166.8 GB disk • 100 computing units	• OS: 64-bit Windows 10 • CPU: Intel Core i-12900KS • GPU: Nvidia GeForce RTX 3090Ti • Memory: 32 GB DDR5 5200 MHz RAM • Storage: NVMe SSD (reads 2400 MB/s, writes 1750 MB/s)

4.1. Preprocessing SWEBOK

The PDF version of each KA chapter was converted into Microsoft Word documents that were manually pre-processed. The following are the rules for manual formatting for each chapter:

1. All text was made lowercase, because, for example, “Software requirements” and “software requirements” were identified as separate entities by the model.
2. The chapter number and chapter name were removed, as these seemed to confuse the model.

3. The page numbers were removed, as they have no significance to the meaning of the text.
4. Example sentences were removed.
    i. For instance, the sentence *"… the throughput requirement for a call center would, for example, depend on how the telephone system, information system, and the operators all interacted under actual operating conditions…"* would be removed, because this sentence would add extra entities which may cause incorrect relationship linking.
    ii. However, an example sentence such as this would remain untouched: *"… some are quality concerns that all software must address—for example, performance, security, reliability, usability, etc.…"*.
5. Bulleted or numbered lists were divested of their list elements and consolidated into blocks of text. The spaCy library does not work well with lists.
6. All occurrences of plural key nouns in each chapter were converted into single format with (s) at the end. For example, "software requirement" and "software requirements" were converted into "software requirement(s)".
7. As was found out later "C++" and "C#" occurrences were confusing model and thus throwing an error in entity recognition and relation extraction.
8. Had to manually resolve doubled pronouns issue in several sentences.
    i. For example, *"…in addition to faults resulting from requirements and design, faults introduced during construction can result in serious quality problems—for example, security vulnerabilities.* ***this*** *includes not only faults in security functionality but also faults elsewhere that allow bypassing of* ***this*** *functionality and other security weaknesses or violations…*" was throwing the error due to "this"

highlighted pronouns occurring in the same sentence. The pipeline was able to distinguish to what each “this” pronoun was referring to.

ii. It was manually fixed to the following sentence: *“…in addition to faults resulting from requirements and design, faults introduced during construction can result in serious quality problems—for example, security vulnerabilities.* ***quality problems*** *include not only faults in security functionality but also faults elsewhere that allow bypassing of* **this** *functionality and other security weaknesses or violations…”*

Please note that much of this can be automated with appropriate use of code constructs, e.g., regular expressions or automated pdf readers.

4.2. First spaCy Implementation

The first relation extraction strategy used ordinary spaCy, which has several helpful utilities for fundamental text processing, to quickly establish a baseline without relying on sophisticated ML algorithms. This baseline was constructed using the Kolonin and Ismail code [30, 15] . Software Requirements, the first KA of the SWEBOK, underwent preprocessing (as described in Section 4.1), and programmatic spaCy relation extraction code was then applied to it. There was no machine learning used. A set of triplets with the following structure were created by this spaCy implementation: "subject" "relation" "object". With pre-written Pandas and Matplotlib code, this triplet collection was shown graphically. The graph was packed with terms in a star pattern due to the large number of entities and unary links, unreadable, making it impossible to interact with and unappealing to look. See Figure 4.2. The characteristics of the first entity-relation graph are given below:

- 341 relations (197 unique) between 477 unique entities
- Generated in approximately 40 minutes and 43 seconds

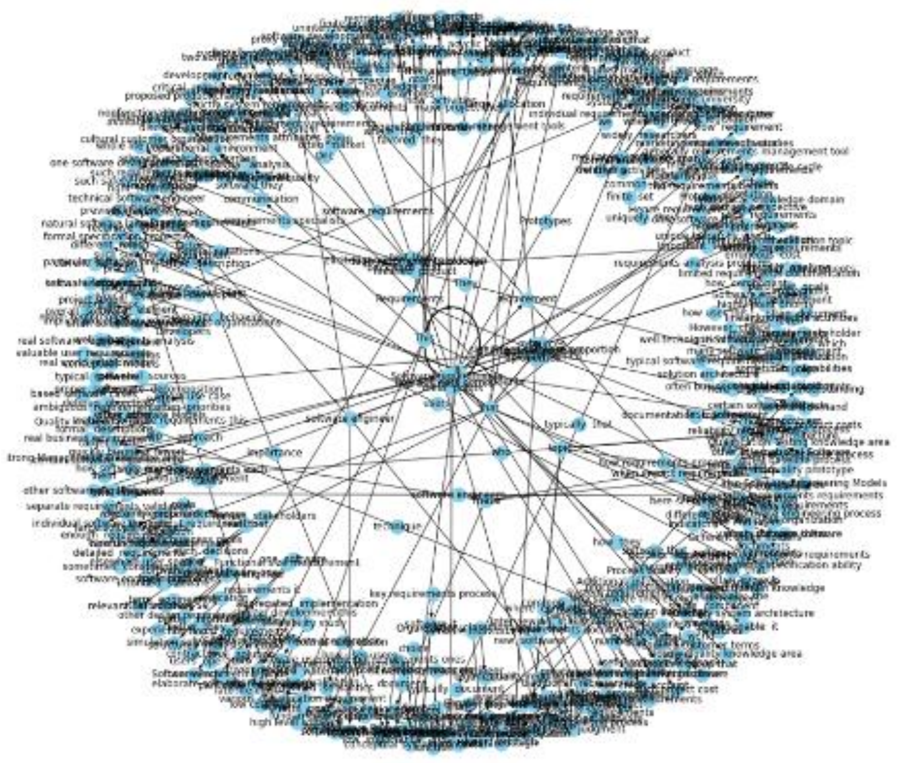


Figure 4.2 Plain spaCy implementation

Triplets listed in see Table 4.2 show how inadequately informed this programmed approach of relation extraction is. Some topics and objects are too vague to be deemed useful, and the connections between them are carelessly taken from the text without any consideration for their semantic context (see Table 4.3). This output is not supported by a taxonomy (such as a hierarchy of subclasses). This is why it is preferable to employ well-formed, reusable relations rather than those that are taken directly from the individual texts, such as those from Wikidata. The outcomes of the subsequent REBEL implementation serve as proof of this.

Table 4.2 Examples of "good" and "bad" triples from plain spaCy RE.

Good triplet	Bad triplet
<organizations> <use> <verification software plans>	<that> <is> <real world>
<software> <comply with> <regulatory authorities>	<these> <include> <international support software>
<scenarios> <provide> <valuable elicitation>	<who> <comprises> <software>
<project management quality> <constrained by> <available resources>	<they> <mediate between> <technical software engineer>
<dynamic behavior> <understood through> <textual user description>	<it> <provide> <realistic product costs>

Table 4.3 The relations extracted from the Software Requirements KA with plain spaCy.

Relation	Occurrences
is	60
are	16
include	10
has	6
provide	5
is important	5
concerned with	4
refers to	4
have	4
be	4

4.3. REBEL Implementation

In this approach with REBEL, there are three component steps that each preprocessed KA document was subjected to, in order to derive an entity-relation graph:

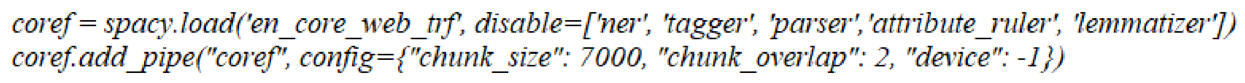
1) Coreference resolution
2) Entity linking
3) Relation extraction

According to spaCy documentation, the module *'en_core_web_trf'* is pre-trained for large text for higher accuracy. Below in Table 5.2, provided results of executing each chapter separately. All chapters followed the rules listed in Section 4.1. As mentioned in Section 4.1 some chapters were throwing errors due to using “C++”, or “C#” or duplicated pronouns in the same sentences. To determine the problem strings in problematic chapters, and because the errors thrown by REBEL were insufficient, the method employed involved splitting the chapter into halves and executing REBEL against each half, repeating this process until only two sentences were left, where the one causing an error was determined to have a string REBEL could not parse.

4.3.1. Coreference Resolution

In spaCy, the act of finding and replacing textual references to the same entity with a single, consistent representation is known as coreference resolution. Text summarization, question answering, and text production are a few examples of activities that can benefit from this in natural language processing. For example, in the sentence "Sundar Pichai is the CEO of Google. He lives in USA," the mention "Sundar Pichai" and "He" refer to the same person. Coreference resolution would replace the second mention with "Sundar Pichai" to make the text more concise. spaCy provides a coreference resolution functionality as a separate module which can be added to the pipeline. Once added to the pipeline, it will resolve any coreference in the text and replace the mentions with their corresponding entities.

For coreference resolution step was setup a spaCy pipeline with following parameters:



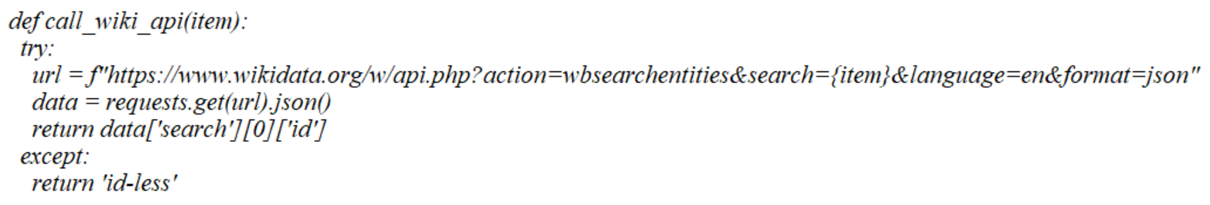
```
coref = spacy.load('en_core_web_trf', disable=['ner', 'tagger', 'parser','attribute_ruler', 'lemmatizer'])
coref.add_pipe("coref", config={"chunk_size": 7000, "chunk_overlap": 2, "device": -1})
```

Figure 4.3 Initializing coreference pipeline

4.3.2. Named Entity Linking

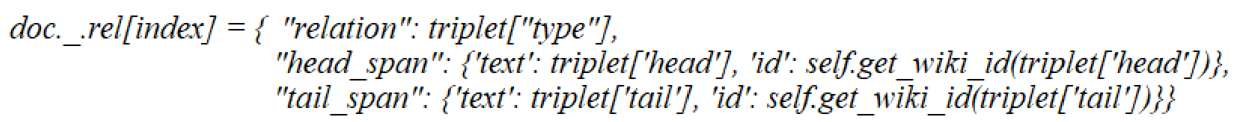
In spaCy, the act of tying identified entities in the text to the relevant articles in a knowledge base, such Wikipedia or DBpedia, is known as entity linking. This enables you to distinguish between entities and offer more context and details about them. For example, if the text contains the entity "Steve Jobs," entity linking can link this mention to the Wikipedia page for the co-founder of Apple Inc. A alternative technique to entity linking was used because the model doesn't allow it. To seek up entities on Wikidata, the system solely used the search entities Wikidata API, see Figure 4.4.

All self-loops are first ignored by the system. Connections that start and terminate at the same thing are called self-loops. The head and tail elements of the relation will then be found in the text using a regex search. There have also been instances that the Rebel model sometimes has hallucinations of things that are not really in the original text [31]. Consequently, a step that ensures that both entities are present in the text was implemented before adding them to the results. The system then uses the Wikidata API to map extracted entities to Wikidata ids, see Figure 4.5. This is a compressed version of entity disambiguation and linking, as previously mentioned; other methods, such the ExtEnd model, are also accessible.



```
def call_wiki_api(item):
  try:
    url = f"https://www.wikidata.org/w/api.php?action=wbsearchentities&search={item}&language=en&format=json"
    data = requests.get(url).json()
    return data['search'][0]['id']
  except:
    return 'id-less'
```

Figure 4.4 Wikidata API call



```
doc._.rel[index] = {  "relation": triplet["type"],
                      "head_span": {'text': triplet['head'], 'id': self.get_wiki_id(triplet['head'])},
                      "tail_span": {'text': triplet['tail'], 'id': self.get_wiki_id(triplet['tail'])}}
```

Figure 4.5 Entity linking using Wikidata API

4.3.3. Relationship Extraction

In spaCy, the process of locating and extracting relationships between things in text is referred to as relationship extraction. Natural language processing activities like information extraction, text summarization, and question answering can all benefit from this. To identify relationships between entities, spaCy has built-in named entity recognition (NER) and dependency parsing tools. The dependency parser of the library may be used to determine the relationships between those entities based on the grammatical dependencies between the words in the text and the NER model of the library can be used to identify entities in text.

For relations extraction step was setup a separate spaCy pipeline with following parameters:

```
rel_ext = spacy.load('en_core_web_trf', disable=['ner', 'lemmatizer', 'attribute_rules', 'tagger'])
rel_ext.add_pipe("babelscape", config={'device':-1, 'model_name':'Babelscape/rebel-large'})
```

Figure 4.6 Initializing RE pipeline

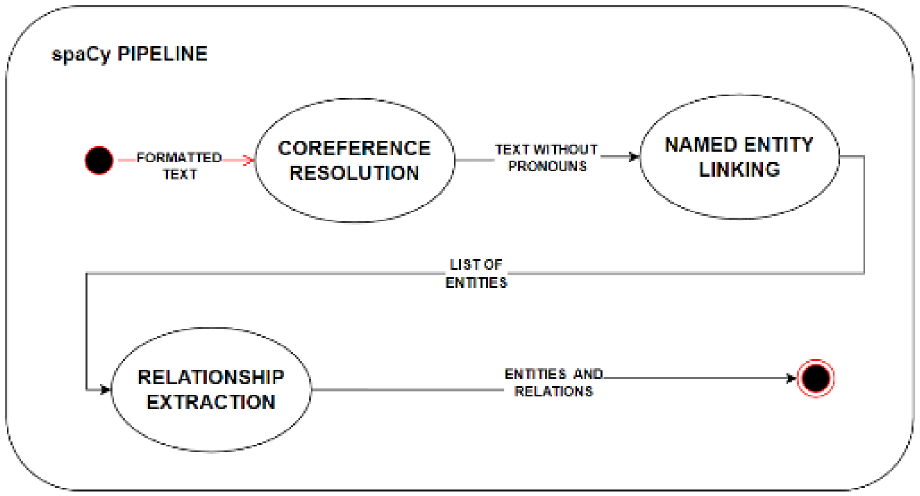


Figure 4.7 DFD model of spaCy pipeline

4.4. Combining spaCy and Neo4j

After completing spaCy pipeline, the process runs a special cypher to upload all triplets into neo4j sandbox. The online Neo4j graph database sandbox allows the storage of Labelled Property Graphs

(LPGs) for three days, with an extension of seven days per request. To prevent the loss of work, the Neo4j sandbox data was migrated to the Neo4j desktop application for further exploration and usage.

4.5. Converting Neo4j to OWL format

To work with and analyze the LPGs in a standardized format, the neosemantics (n10s) RDF toolkit was installed. This was used to output Terse RDF Triple Language (TTL, or "turtle") files for each KA graph, entire SWEBOK and SWECOM. With these turtle files, the free graph database platform, GraphDB, was used to perform all analyses and visualizations of the KA graphs. The primary barrier to using Neo4j is learning its query language, Cypher, which was not seen as a valid time investment.

4.6. Processing Entire SWEBOK

For processing entire SWEBOK, all preprocessed individual chapters were combined into one single word document and parsed through spaCy pipeline using Google Colab Pro. The result of execution is recorded in Table 5.2

4.7. **Processing SWECOM**

For SWECOM were chosen several chapters to test the procedure. Chapters 3-5 were chosen due to a higher amount of text rather than tables or figures as in another chapters. Selected chapters were preprocessed with the same rules described in Section 4.1. The result of this procedure is shown in Section 5.3. The fact that not all SWECOM was processed is due to a huge amount of tables, and the data in the tables needs to be represented as a text for further processing. That Would take a greater amount of time when processing SWEBOK chapters.

5. Results

In this section presented a detailed analysis of the experimental results, including the statistical measures used to evaluate the performance of the system. Overall, the findings indicate that by addressing some of the major issues and shortcomings in the existing methods, the suggested technique has the potential to define USECG that can significantly reduce gap between software engineering curriculum and industry.

5.1. Relations from REBEL Component

Table 5.1 shows all encountered relations after processing chapters from SWEBOK with their definition from a Wikidata [32]. Wikidata relations offer a standardized approach to express relationships between things, making it simpler to integrate data from many sources and systems. This is one of the main advantages of utilizing Wikidata relations.

Table 5.1 Wikidata relations and their definitions [32]

Encountered REBEL relation	**Wikidata definition**
author	main creator(s) of a written work (use on works, not humans)
applies_to_jurisdiction	the item (institution, law, public office, public register...) or statement belongs to or has power over or applies to the value (a territorial jurisdiction: a country, state, municipality, ...)
based_on	the work(s) used as the basis for subject item
country	sovereign state that this item is in (not to be used for human beings)
depicts	depicted person, place, object or event
designed_by	person(s) or organization which designed the object
developer	organization or person that developed the item

discoverer_or_inventor	subject who discovered, first described, invented, or developed this discovery or invention
different_from	item that is different from another item, with which it may be confused
diplomatic_relation	diplomatic relations of the country
facet_of	the main aspect of this topic
field_of_this_occupation	field corresponding to this occupation or profession
field_of_work	specialization of a person or organization
followed_by	the immediately following item in some series of which the subject is part
follows	the immediately prior item in some series of which the subject is part
has_cause	underlying cause, thing that ultimately resulted in this effect
has_effect	effect of this item
has_part	object is a part of this subject
has_parts_of_the_class	the subject instance (the subject is not a class) has one or more parts of the object class
has_subsidiary	subsidiary of a company or organization
inception	time when an entity begins to exist
instance_of	this item is a concrete object (instance) of this class, category or object group
item_operated	equipment, installation or service operated by the subject
main_subject	primary topic of a work
manufacturer	manufacturer or producer of this product
opposite_of	item that is the opposite of this item

organizer	person or institution organizing an event
owned_by	owner of the subject
owner_of	entities owned by the subject
parent_organization	parent organization of an organization, opposite of subsidiaries
part_of	subject is a part of that object
participant	person, group of people or organization that actively takes/took part in the event
platform	platform for which a work was developed or released, or the specific platform version of a software product
practiced_by	type of agents that study this subject or work in this profession
product_or_material_produced	material or product produced by a government agency, business, industry, facility, or process
said_to_be_the_same_as	this item is said to be the same as that item, but it's uncertain or disputed
studied_by	subject is studied by this science or domain
studies	the object that an academic field studies; distinct from field of work
subclass_of	all of these items are instances of those items; this item is a class of that item
use	main use of the subject (includes current and former usage)
used_by	item or concept that makes use of the subject (use sub-properties when appropriate)
uses	item or concept used by the subject or in the operation

5.2. Results Of REBEL Component and Neo4j

The results of each KA in neo4j shown in Table 5.2. The average time for processing individual chapters with free Google Colab was 40 mins. Average time with Google Colab Pro is 7 mins 7 seconds. The visual representation of a partial graph for Requirements KA is shown in Figure 5.1. The graph in neo4j is more appealing, showing relations and interactable. In addition, neo4j cypher language allows to display desired data, such as show all entities which has relation "instance_of" or "used_by".

Table 5.2 Execution time and word count for each SWEBOK chapter.

SWEBOK Chapter	**Preprocessed Word Count**	**REBEL execution time (minutes, seconds)** with Google Colab	**REBEL execution time (minutes, seconds)** with Google Colab Pro	**Number of Entities**	**Number of Relations**
01 – Requirements	6614	40m13s	6m43s	308	317
02 – Design	4525	37m42s	5m10s	306	279
03 – Construction	5133	29m21s	5m32s	314	324
04 – Software Testing	6617	33m51s	6m44s	375	390
05 – Software Maintenance	4271	28m09s	4m39s	228	264
06 – Software Configuration Management	5514	29m44s	5m14s	261	280
07 – Software Engineering Management	5157	31m40s	5m33s	319	334
08 – Software Engineering Process	5593	33m22s	5m9s	261	284
09 – Software Engineering Models and Methods	4159	26m12s	4m32c	235	252
10 – Software Quality	6134	32m45s	6m42s	335	375
11 – Software Engineering Professional Practice	5261	34m26s	5m46s	293	299
12 – Software Engineering Economics	6153	44m56s	7m44s	424	460

13 – Computing Foundations	12846	93m53s	17m41s	715	891
14 – Mathematical Foundations	7277	61m26s	11m10s	414	516
15 – Engineering Foundations	7057	49m39s	8m28s	449	493
All Chapters	92359		127m28s	3958	5761

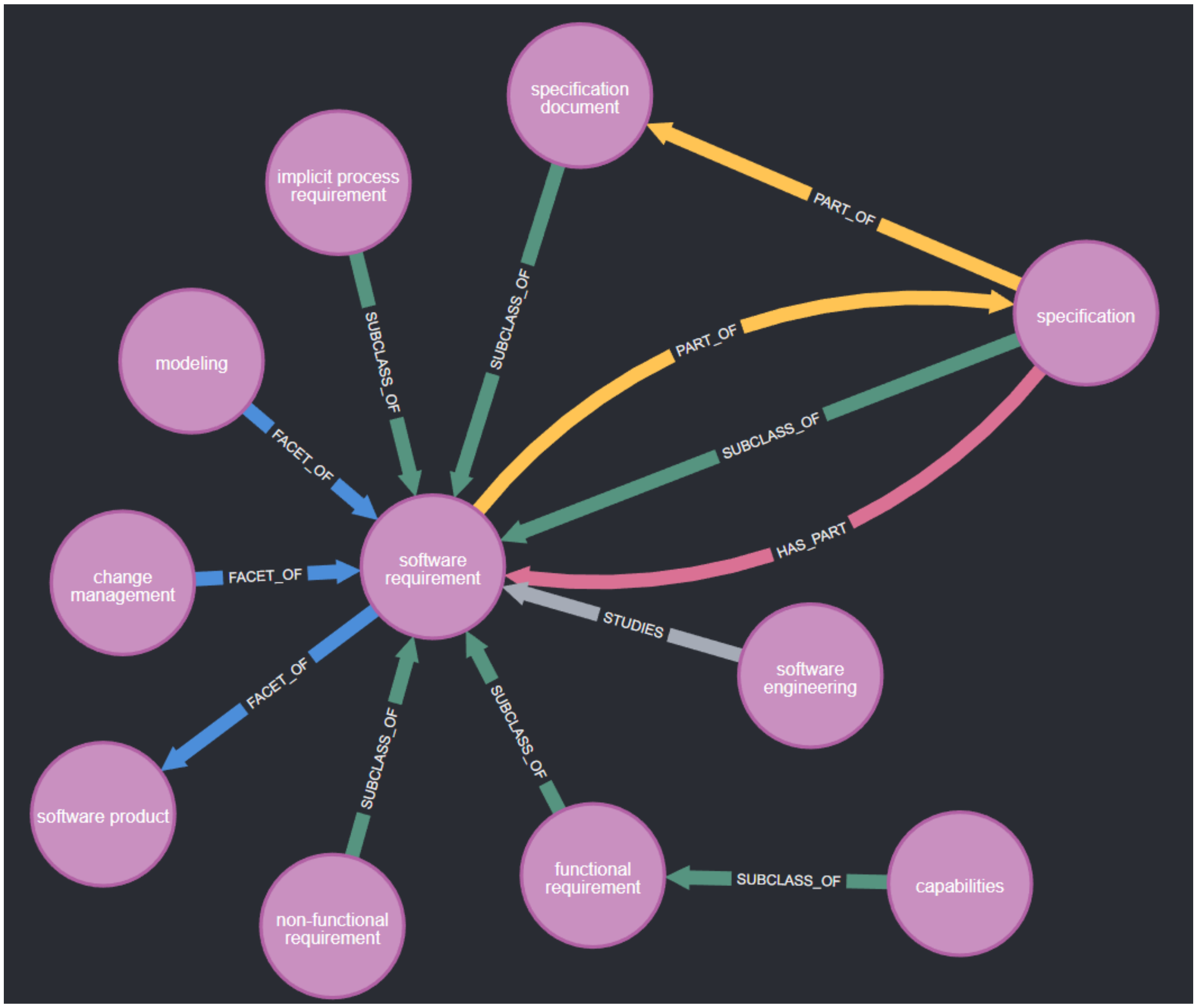


Figure 5.1 Example of extracted data from Requirements KA

Figure 5.1 shows only 11 entities from 308 from Requirements KA. This graph with a few entities allows us to see what a various relation between entities.

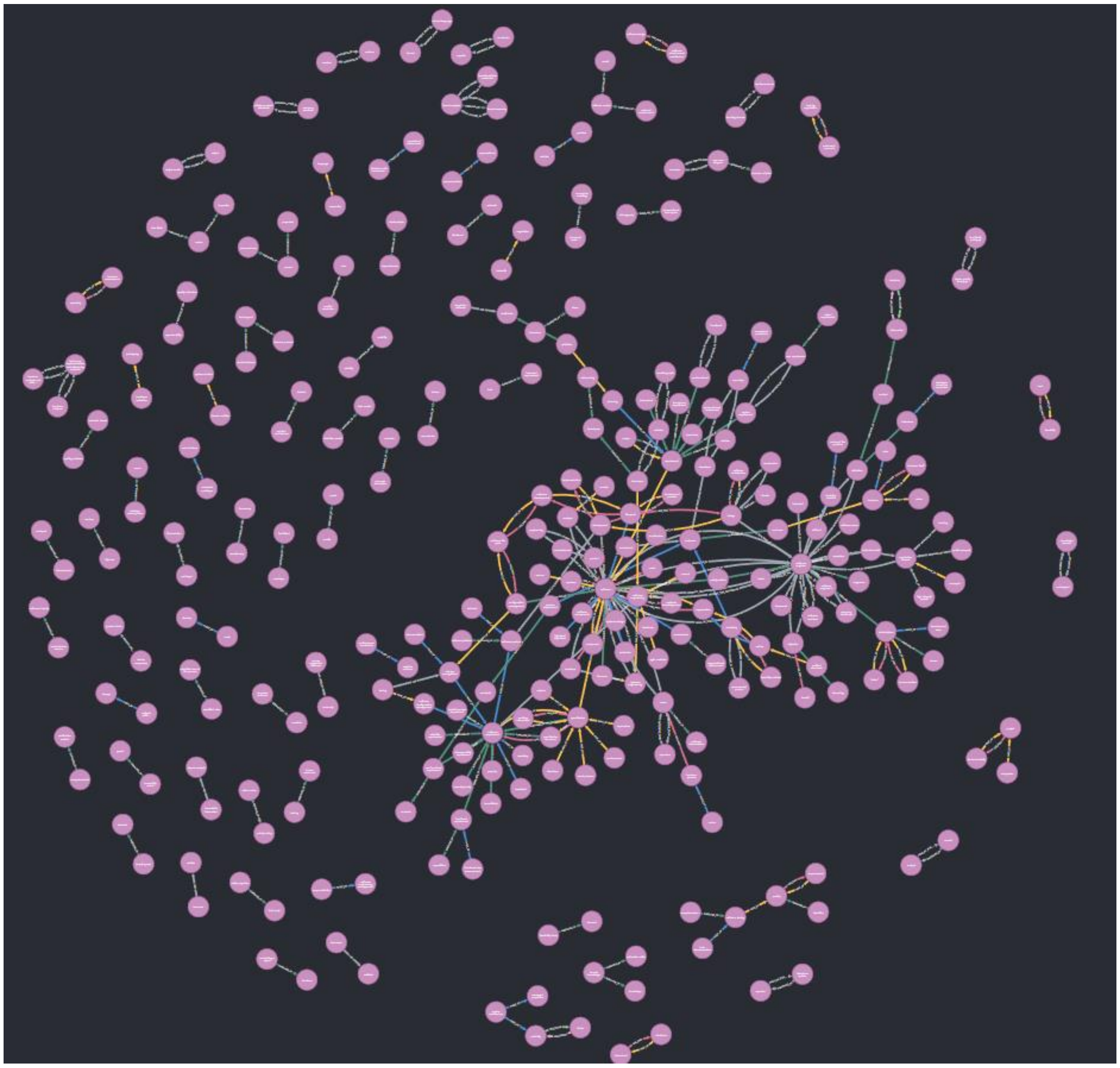


Figure 5.2 Full graph for Requirements KA

Figure 5.2 shows entire graph for Requirements KA form SWEBOK. As can be seen some of the nodes are not attached to the main cluster. This will be discussed in Section 6.

5.3. Results of SWECOM

As was mentioned above for SWECOM were used a few chapters. Table 5.3 shows the final results of executing preprocessed chapters from SWECOM.

Table 5.3 SWECOM execution results

Word Count	**REBEL execution time (minutes, seconds)** with Google Colab Pro	**Number of Entities**	**Number of Relations**
1670	1m55s	97	88

Figure 5.3 shows a few entities with their relations. As can be seen some of the entities are not attached to the main cluster of nodes. As shown in Figure 5.1, Figure 5.2 and Figure 5.3 some nodes were not connected with a main cluster. That happened because of predefined entailment prediction value in REBEL component for triples assignment set to 0.75 [31].

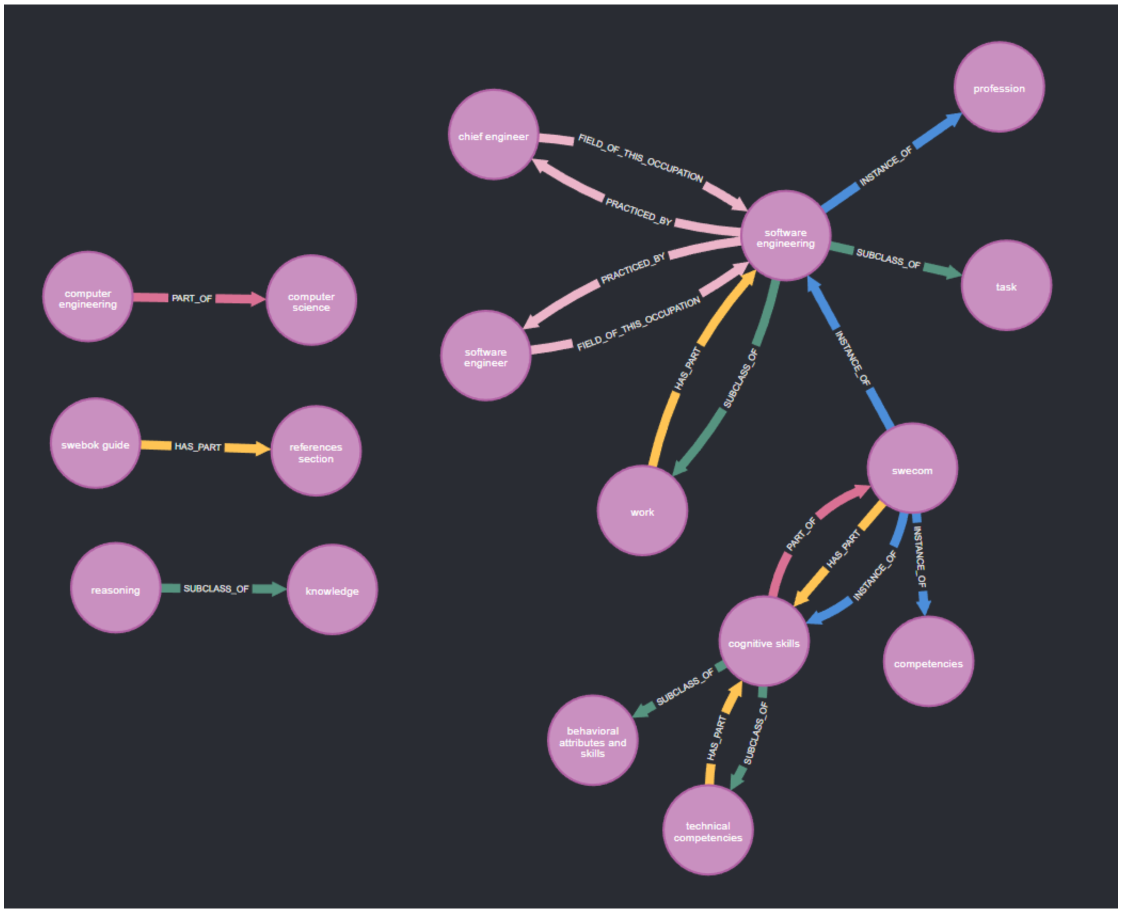


Figure 5.3 SWECOM neo4j graph

6. Discussion, Conclusion, and Future Work

In this part, we examine the meaning of these findings and make judgments regarding the possibility of the suggested strategy to advance software engineering methods. We specifically go through the approach's advantages and disadvantages, as well as the ramifications and practical concerns of adopting it.

6.1. Discussion

The study's findings indicate that defining USECG can provide a unified model to assess software engineering curriculums, which has a lot of promise. To fully benefit from the advantages of the suggested strategy, further research and development are still needed in a few key areas.

The proposed method can be further improved. Such as using PDF extracting library from spaCy. This library allows to convert any PDF document into spaCy objects without converting it first into text and then parsing. This library was not used because it was unclear what particular sections of the SWEBOK would resolve as once processed, e.g., tables, figure captions, page numbers, etc. With the manual preprocessing step, such components of the SWEBOK were controlled.

The knowledge of the software engineering field is concretized at a mid-level OWL level in the knowledge graph that was created. However, this knowledge graph is not an ontology, at least not in the sense that it has an ontological structure. There are only concepts related to software that are connected by extremely generic qualities, not an explicit rule system (like SWRL rules or OWL limitations). Future studies should look at integrating higher-level ontological structure into SWEBOK knowledge graphs generated by NLP in order to facilitate interoperability with other discipline BOKs (like SEBoK [33]) and take advantage of OWL's reasoning capabilities.

Overall, even if the suggested technique still necessitates some manual work, its potential benefits for enhancing software engineering education are considerable, and it marks a substantial advancement toward the creation of a more integrated and thorough software engineering curriculum.

6.2. Conclusion

The suggested approach is a crucial first step towards establishing the USECG. The advantages it offers for the growth of the USECG in the future are highly promising, even if it now requires some human effort, such as text preparation. Software engineering educators and curriculum developers will be able to process different guidelines and models using this technique to create a single, comprehensive ontology that can be used to check the accuracy of current software engineering curricula or provide recommendations for brand-new ones. In order to guarantee that students obtain a well-rounded and thorough education in software engineering, the USCEG will also assist in identifying areas where extra information or attention may be required. The USCEG may also be used to improve interaction and cooperation between academics and industry professionals, ensuring that software engineering education is kept up to date and pertinent to market demands.

The created knowledge graph clearly shows the Wikidata relations, therefore this may be utilized as the foundation for creating a comprehensive taxonomy. The graph is also not positioned in relation to any higher-level ontology (e.g., BFO). As the act of classifying objects according to a "is-a" hierarchy is the foundation of the whole area of ontology, an ontology cannot be said to exist without a taxonomy, or hierarchy, as its foundation.

It is important to research more NLP methods as well. Here, a minimally sophisticated system using commercial NLP techniques and graph data platforms is described. Science is now interested in the automatic creation and populating of knowledge graphs using NLP.

6.3. Future Work

Future work should concentrate on improving and streamlining the whole process, such as minimizing or eliminating text preprocessing, improving relationship extraction, processing the rest of the guidelines and creating a hierarchy of entities. It might be worth training a new model to eliminate loose nodes as shown in graphs above.

In addition, more work should be put into encouraging the approach's acceptance within the software engineering community by bringing to light its potential advantages and offering tools and assistance to make it easier to put into practice. Overall, the study's findings indicate that, while there is still work to be done, the suggested strategy has a bright future for defining USECG, enhancing the creation and maintenance of the software engineering curriculum, and perhaps making a substantial contribution to the discipline.

REFERENCES

[1] N. E. A. M. Almi, N. A. Rahman and D. Purusothaman, "Software Engineering Education: The Gap Between Industry's Requirements and Graduates' Readiness," *IEEE,* 2011.

[2] S. Hanna, H. Jaber, A. Almasalhem and F. A. Jaber, "Reducing the Gap between Software Engineering Curricula and Software Industry in Jordan," *Journal of Software Engineering and Applications*, 2014.

[3] L.-Q. Kuang and X. Han, "The Research of Software Engineering Curriculum Reform," in *International Conference on Medical Physics and Biomedical Engineering*, 2012.

[4] K. Beckman and S. Khajenoori, "Collaborations: Closing the Industry–Academia Gap," *IEEE,* 1997.

[5] P. Bourque and R. E. Fairley, SWEBOK: Guide to the software engineering body of knowledge, IEEE Computer Society, 2014.

[6] "Software engineering competency model: IEEE Computer Society," *IEEE,* 2014.

[7] M. Ardis, D. Budgen, G. W. Hislop, J. Offutt, M. Sebern and W. Visser, "SE 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," 2015.

[8] "ABET," [Online]. Available: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-engineering-programs-2021-2022/. [Accessed 10 August 2022].

[9] A. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development,* vol. 44, no. 1,2, pp. 206-226, 2000.

[10] "“What Is Natural Language Processing?”," IBM, [Online]. Available: https://www.ibm.com/topics/natural-language-processing. [Accessed 2023].

[11] S. Vajjala, B. Majumder, A. Gupta and H. Surana, Practical Natural Language Processing, O'Reilly, 2020.

[12] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *ArXiv,* vol. abs/1810.04805, 2019.

[13] O. Mendes and A. Abran, "Issues in the development of an ontology for a emerging engineering discipline," in *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering*, 2005.

[14] M. A. Musen, "The Protégé Project: A Look Back and a Look Forward," *Association of Computing Machinery Specific Interest Group in Artificial Intelligence,* vol. 1, no. 4, pp. 4-12, 2015.

[15] H. Ismail, "Relationship Extraction from Any Web Article," December 2021. [Online]. Available: https://hami-asmai.medium.com/relationship-extraction-from-any-web-articles-using-spacy-and-jupyter-notebook-in-6-steps-4444ee68763f. [Accessed 2023].

[16] B. Florian and M. Kaltenböck, Linked Open Data: The Essentials., Viena, 2011.

[17] N. Guarino, D. Oberle and S. Staab, "What is an Ontology?," in *Handboof on Ontologies*, 2009, pp. 1-17.

[18] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition,* vol. 5, no. 2, pp. 199-220, 1993.

[19] W. Borst, "Construction of Engineering Ontologies," *Institute for Telematica and Information Technology,* 1997.

[20] R. Arp, B. Smith and A. D. Spear, Building Ontologies with Basicv Formal Ontology, The MIT Press, 2015.

[21] O. Lassila and D. McGuiness, "The role of frame-based representation on the semantic web," 2001.

[22] M. Uschold and M. Gruininger, "Ontologies and Semantics for Seamless Connectivity," *SIGMOD Record,* vol. 33, no. 4, pp. 59-64, 2004.

[23] I. Robinson, J. Webber and E. Eifrem, Graph Databases, 2nd Edition, O`Reilly Media, Inc., 2015.

[24] Princeton University, "About WordNet," 2010. [Online]. Available: https://wordnet.princeton.edu/. [Accessed 2023].

[25] A. Alarifi, M. Zarour, N. Alomar, Z. Alshaikh and M. Alsaleh, "SECDEP: Software engineering curricula development and evaluation," *Elsevier,* 2016.

[26] W. C. Johnson, "Challenges in university-industry collaborations.," in *Universities and Business: PArtnering for the Knowledge Society*, 2006, pp. 211-222.

[27] A. Abran and O. Mendes, "Software engineering ontology: a development methodology," 2004.

[28] A. Abran, J. J. Cuadrado, E. García-Barriocanal, O. Mendes, S. Sánchez-Alonso and M. A. Sicilia, "Engineering the Ontology for the SWEBOK: Issues and Techniques," in *Ontologies for Software Engineering and Software Technology*, Berlin, Springer, 2006.

[29] M. Towhidnejad, O. Ochoa and A. Kiselev, "An Analysis of the Software Engineering Curriculum Using the," in *ASEE Southeastern Section Conference*, 2020.

[30] A. Kolonin, "Application aspects of social data processing (Social intelligence technologies or Social computing)," Novosibirsk: Novosibirsk State University, 2021.

[31] P.-L. H. Cabot and R. Navigli, "REBEL: Relation Extraction By End-to-end Language generation," in *Findings of the Association for Computational Linguistics: EMNLP 2021,* Association for Computational Linguistics, 2021, p. 2370–2381.

[32] Wikidata, "Wikidata:Database reports/List of properties/all," [Online]. Available: https://www.wikidata.org/wiki/Wikidata:Database_reports/List_of_properties/all.. [Accessed 2023].

[33] "Guide to the Systems Engineering Body of Knowledge (SEBoK)," SEBoK, 19 May 2021. [Online]. Available: https://sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK). [Accessed 2023].

[34] W. C. Johnson, "Challenges in university-industry collaborations.," in *Universities and Business: Partnering for the Knowledge Society*, 2006, pp. 211-222.

[35] J. W. E. E. Ian Robinson, Graph Databases, 2nd Edition, O'Reilly Media, Inc., 2015.