Innovative Computing Review (ICR)

Volume 1 Issue 2, Fall 2021
ISSN(P): 2791-0024 ISSN(E): 2791-0032
Journal DOI: https://doi.org/10.32350/icr
Issue DOI: https://doi.org/10.32350/icr/0102
Homepage: https://journals.umt.edu.pk/index.php/icr

Article: **ReqSpecOnto: Investigating Explicit Software Requirements Specification**

Author(s): Usman Ahmed[1], Amjad Farooq[2], Tayyaba Farhat[3]

Affiliation: [1]Foundation University Islamabad, Pakistan
[2]University of Engineering and Technology, Lahore, Pakistan
3Superior University, Lahore, Pakistan

Citation: A, Usman, F. Amjad, and F. Tayyaba, "ReqSpecOnto: Investigating Explicit Software Requirements Specification", *Innova Comput Rev*, vol. 1, no. 2, pp. 44–70, 2021. https://doi.org/10.32350/icr/0102/03

Journal QR

Article QR

Usman Ahmed

A publication of the
School of Systems and Technology
University of Management and Technology, Lahore, Pakistan

ReqSpecOnto: Investigating Explicit Software Requirements Specification

Usman Ahmed[1]*, Amjad Farooq[2], Tayyaba Farhat[3]

ABSTRACT: The specification of customers' need as software requirements in natural language create ambiguities (in the requirements) and may fail the software project. Generally, customers are unable to define their needs due to the lack of domain understanding, technological constraints, and knowledge gap between the stakeholders and the requirements analysts. One of the most effective approaches to minimize these gaps and ambiguities for requirements specification and validation is the use of ontologies. However, the current approaches are mostly limited to the translation of ambiguous software requirements. This paper discussed, analyzed and compared the current usage of these ontologies and found that these approaches are time-consuming and create complexities in the overall development process. It presented a requirements specification ontology (ReqSpecOnto), bypassing the need for creating an ambiguous Software Requirement Specification (SRS). The upper software requirements ontology is defined in Ontology Web Language (OWL) which can be applied to different software scenarios. A case study of budget and planning system for a state physics lab was selected to specify its requirements as derived ontology from the upper ontology created. The results are validated through HermiT and Pellet reasoners to verify the defined relationships and constraints. Finally, SPARQL queries were used to obtain the necessary requirements.

INDEX TERMS: ontology engineering, requirements specification ontology, semantic relations, software requirements, upper ontology

I. INTRODUCTION

In software development, requirements engineering is a

[1]Foundation University Islamabad, Pakistan
[2]University of Engineering and Technology, Lahore, Pakistan
[3]Superior University, Lahore, Pakistan
*Corresponding Author: usman.ahmed@fui.edu.pk

complex and time-consuming process. The requirements are specified and documented in a software requirements specification (SRS) document. While the main focus in the documentation is on building a bridge between the vague concepts collected from customers and the requirements of the engineering team. Non-technical customers are unable to explain the exact requirements to the technical, experienced analysts due to lack of proper domain knowledge. These vague concepts are documented using natural language that is ambiguous in itself. So, one cannot get a clear and concrete set of requirements for the system development. If these ambiguities are not properly identified and corrected timely, the consequences can be devastating. For instance, any misunderstanding of the exact needs of customers and stakeholders will need a lot of revision after deployment of the system. Multiple studies have shown that maintenance costs may increase up to 90% of the total development cost of the entire project because of requirement errors most of them occur due to miscommunication, ambiguities and implicit requirements [1], [2]. Identification and resolution of these problems during the maintenance phase need painstaking work which not only increases the cost of development but also results in less satisfied clients [3].

The use of modelling languages, however, within the specification document that provides semi-formal representation, does reduce some level of ambiguity. Additionally, modelling languages are still tied to natural language for labelling and this can cause e two major problems. Firstly, naming elements with terms or labels using natural language can introduce multiple interpretations. This limits the use of models as a way of communicating knowledge among the phases of software development. Secondly, the use of natural language delivers semantics that is not machine-process-able. This can cause problems in validating or querying data in modelling tools and restrict the logical reasoning since processes cannot logically retrieve a requirement specification that is not explicitly documented. Therefore, the requirements are validated for consistency and accuracy as a sub-task within requirements engineering. In the past, the

collaboration between stakeholders has long been a focus by several researchers but most of their approaches are downsided [4].

An effort has been made to reduce ambiguities in the SRS through natural language processing (NLP) tools [5]. The researchers either try to find ambiguities in the requirements document or generate models from them. A study was aimed at employing the rule based NLP techniques to find quality defects from industrial based natural language requirements that are annotated by domain experts [6]. Another approach refers to the application of the code smells concept to requirements engineering for identification of defects in requirements [7] but is subjective and employs multiple reviews. One of the main contributions of most NLP tools is to automate the extraction and translation of natural language specified requirements into conceptual models so they can be manually validated by analysts themselves [8], [9]. This results in a lack of precision because it is still manual and dependent on humans to identify and verify errors. While these NLP tools are individual efforts to find ambiguities, they are not aimed at promoting the interoperability of knowledge in multiple phases of software requirements engineering and development. Rather they only support a representation of knowledge of requirements. Hence, a representation for specifying software requirements that can allow machine-process-able semantics in the first place is positively required. This will reduce extra work and make it possible to resolve the ambiguities in requirements. Researchers of semantic web and linked data have been trying to produce a mechanism that can reduce the ambiguities and make requirements testable. Ontologies and their use in several domains have been effective in recent times because of their semantic and syntactic characteristics. Ontologies arrange items or concepts of data in such a way whereby meaning is created among them by using relations, restrictions between data terms, making vocabulary and taxonomies more meaningful, hence fulfilling the overall objectives. In software engineering, its popularity has increased for two main reasons to provide machine reasoning, and to facilitate semantic interoperability [10].

Contributions have been made in multiple phases of requirements engineering to resolve inconsistencies and ambiguities in the requirements specification. A detailed systematic literature review [11] identified the trend of the application of ontologies for supporting requirements specification. Most of the work on the application of ontologies in requirements specification focuses on converting or translating previously developed paper-based SRS into ontology for checking consistency and correctness of the document. These approaches are not only time consuming and complex but are also prone to introduction of ambiguities in the specified requirements. Solutions that support the process models in requirement engineering phases as inter-connected and inter-communicating are needed for better requirement specification, and can contribute to a generic framework in all projects. This can only be achieved if we rely entirely on ontologies for their specification in an attempt to shift the paradigm from trying to give semantics to ambiguous specified requirements to an approach of defining the semantics from the base that can be related to each project. In this paper, a requirements ontology has been developed for the proposed framework to generalize the concepts of requirements engineering. An instance is derived from the requirements ontology of the specific domain of the system to be developed called as domain ontology that is related to the domain of the particular software being considered. The reasoning and use of SPARQL have been applied to validate the requirements and inference check for conflicting requirements. The primary aim and contribution of this paper are to present a framework which is based on model-driven requirements engineering that is used to define, specify and validate software requirements in the form of ontologies. This will entirely replace the use of paper-based SRS with ontologies and the validated specification can be reused to support and automate other software engineering phases. We have formulated the following four research questions.

CQ-1: How software requirements can be formally specified in ontologies?
CQ-2: Can software requirements be validated through ontologies?
CQ-3: How can we extract the requirements to meet the needs of the software engineering team?

The rest of the paper is organized as follows: Section 2 represents literature review. Section 3 describes the methodology adopted to resolve the problem. This is further divided into sub-sections; the discussion on the collection of glossaries, taxonomies and the use of Protégé tool for developing the ontology. The experiment, discussion of results and conclusion are described in Sections 4 and 5 respectively.

II. LITERATURE REVIEW

Requirements engineering is the process of systematically eliciting, analyzing, specifying, validating and managing requirements for a software system [12]. The process of requirements elicitation involves tacit or implicit knowledge which results in ambiguous requirements. Therefore, it involves the risk of misunderstanding [13].After the requirements have been gathered through elicitation, these are documented as a software requirements specification (SRS) document which includes requirements in the form of natural language and is represented as conceptual model[14]. The SRS needs to be validated and verified to identify e conflicts and to gather agreed-upon solutions to resolve them. The specification of requirements is a challenging task that must be properly managed during the process of requirement engineering.

Firstly, every detail acquired in the elicitation phase is included, and if a minor detail of the specification is missed, the development team either skips the additional functionalities or adds what was not required by the customer. In both cases, according to the Kano Model [14], the developed system will be unable to satisfy the customer. Therefore, extra costs will be spent during the maintenance phase. The requirements management also needs to be considered throughout the software development process specifically in requirements engineering due to frequent changes in the requirements.

Ontologies were first developed by Artificial Intelligence researchers [15]. They are used to provide machine-processable semantics so that they can be shared among several other software and tools. Ontology is a formal explicit specification of a shared conceptualization[16]. Ontologies have also been used in requirements engineering for

solving several problems like specifying more accurate and unambiguous requirements, managing requirements knowledge, automating the generation of test cases and domain modelling [10], [17], [18]. The detailed review by [11] accounts for the work of ontologies in software engineering. Here, ontologies are divided into two basic categories; ontologies of domain and of software artefacts. The first category consists of those ontologies that attempt to conceptualize the domain knowledge of software engineering domains or sub-domains. The second category consists of functionalities and characteristics of the development process or that of the functioning of the software thereby supporting its artefacts. Among these domains, ontologies are very generic like in the work of Abran and Mendes who created a 4000-concept ontology on the basis of SWEBOK guide which attempts to conceptualize the whole body of knowledge of software engineering [19]. Although we have also employed SWEBOK guide in our taxonomy building among other sources, but this approach is nevertheless more generic and attempts to conceptualize the entire software engineering knowledge to target multiple applications. On the contrary, our ontology differs from this approach since we are focused on requirements specification and their validation instead of encompassing knowledge of the entire software engineering domain.

An ontology for SQL was developed for the abstraction of the standard database query language [20].Similarly, an MIT project called SIMILE created an ontology for java concepts, while, a lot of work is done in ontologies used as software artefacts. According to Ruiz et al.,[11] most of the work in SE relating to ontologies is focused on their application in domain modelling. The requirements specification of resolving ambiguities and inconsistencies in software requirements focuses on either finding the consistency and correctness, or ensuring the quality of the requirements specification document by converting the SRS into ontologies [21].

The requirements ontology for a specific domain was created to improve the quality of SRS by converting the requirements into the ontology. Their approach

develops a hierarchy of functions followed by relationships of functional requirements and their attributes[22], [23], presenting how the requirements can be analyzed using domain ontologies in order to find deviations from the requirements document. The proposed method focuses on domain ontologies where the author argues that it is essential in the process of requirements engineering. The analyst knows very little about the domain and therefore gets help from the domain experts. According to the study, domain experts can be replaced by using domain ontology so that analyst can refer to the ontology for domain-specific knowledge using inference rules.

Ismail [24] considered a semi-automated approach and created system specification ontology (SSO) to convert the previously developed SRS into ontology. Nevertheless, previous ontology learning approaches like CRCTOL, Text2Onto adopt methods that only extract concepts and their relationships. Therefore, an approach was proposed, which can learn ontology along with individuals and their concepts as a major contribution of their study. Avdeenko et al. [25]presented an approach or guideline using the Protégé tool known as DL Query to validate the correctness of an SRS.

Mahmud et al. [26] created a language called ReSA that allows requirements specification use a domain ontology, and apply it to an automotive domain for requirements specification of a component of that domain. However, this work is different from using an upper ontology in combination with domain ontologies which would be our approach. The creation and use of an upper ontology to support requirements engineering activities have benefits over using a language to specify requirements. But, on the other hand, researchers are more interested in creating a framework for developing requirements specification directly from elicitation without the need for an intermediary ambiguous and inconsistent SRS document. A recent study conducted by [17] as part of their project called, OSTAG or simply, Ontology-based Software Test Case Generation, presents a guideline for developing a requirements ontology to support other software development phases. The main focus of this work was to create an ontology from the software requirements specification

documents which can involve ambiguities and complexities. A study was conducted by[18] who pointed out how it is important to have a formal specification using an ontology to support domain modelling specifically for complex domains. The proposed approach uses an ontology to extract knowledge to create models and then evaluate the result on specified requirements. They have provided an approach of generating test cases using inference rules from ontology [17].The inference rules are coded in Prolog so ontology is serialized in functional style OWL syntax to Prolog using Python code. According to Zong-yong et al. [27], most of the studies on using ontologies in requirements elicitation focus on using a single global ontology which cannot be reused. Therefore, they presented a multiple ontologies framework based on KADS modelling with combining top-level, task, domain and application ontologies to facilitate requirements elicitation and reuse. Amarilis et al. [28] provided an ROF which is the rule based framework of ontology for automatically generating requirements. In this framework they focus on documenting standardized version of SRS in IEEE format and applied it at a university system to document teacher workload. Conformity to the ambiguous SRS standard format can again introduce inconsistencies later even after the explicit specification.

Furthermore, there has been a trend of following ontologies and semantic web in requirements engineering for representing knowledge to support a very minor product or project-specific scenarios and domains. It should be noted that ontologies are surely domain-specific that can be used to support communication among sub-domains of the same domain. The use of this approach of creating ontologies to cope with a specific problem in a project can restrict their reuse and communication among other such smaller ontologies. The concept of ontologies was to support interoperability and communication among multiple process models. According to W3C, ontologies are used to communicate knowledge within process models. According to [29] requirements engineering of Model-Driven Development (MDD) that uses model transformations to automate, the software development process lacks complete solutions to requirements models. This review also discusses that models are

used not only to describe requirements in the MDD context, but also in structured or unstructured natural language.

Another recent approach [30] was to combine NLP techniques and ontologies for the purpose of bridging the gap between trained and untrained users from elicitation of requirements to the specification of those requirements. Previous efforts of NLP techniques to translate the requirements into models were made to understand and interpret natural language statements to uncover their meaning and semantics. Combination of these techniques and technologies with ontologies for the same purpose of understanding specified requirements is also the same thing with the addition of previous records being managed as a knowledge base in a way that their relationships are understood. However, these efforts can only be considered as individuals in ontologies which are only project or product-specific in terms of solving problems. Hence, we need an approach that can cater to the needs of maximum projects within a domain, reducing ambiguities in the requirements engineering process and ultimately the cost of development.

III.ONTOLOGY ENGINEERING OF REQUIREMENTS SPECIFICATION ONTOLOGY

In ontology engineering, the ontology development process is driven by scenarios that result from problems that arise in applications or existing ontologies. They also include possible solutions to those problems. According to [31], any proposal of new ontology must include these scenarios and possible solutions that describe the problem stories or scenarios, the semantic objects and relations between them. In this paper, the problem scenario arises from the requirements specification where requirements are documented in natural language in the form of SRS. These are subject to multiple interpretations and ambiguities that are neededto be specified and validated in ontologies.

A. Use Cases

The scenario is the budget and planning system of a state physics lab in the United States. The selected requirements including multiple roles and requirement types are as follows:-

UC-1. End users have viewed into workflow to assess current status, bottlenecks and next steps

UC-2. End user can configure its own view.
UC-3. Admin can configure views of users.
UC-4. Users can request resources and admin can approve those requests.
UC-5. End user can add notes to workflow.
UC-6. All users as per their security role can see forecasts.
UC-7. End user can select reports from predefined reports.
UC-8. Admin can add due dates in calendar that contains lab-level project planning activities.
UC-9. Users have security roles for access to data that are configured by the main admin.
UC-10. End users can drill down from hierarchical data in reports for analysis.

B. Vocabulary

A general vocabulary of software requirements to build upper requirements specification ontology was needed. We collected all terms possibly from the requirement engineering field which are used in the process of requirements specification. The first choice for collecting the terms was a glossary published by the International Requirements Engineering Board of 128 Terminologies. All them were not useful for our purpose. However, another useful source was the SWEBOK (Software Engineering Body of Knowledge). Most of the meanings and terms that were unavailable in the IREB glossary were derived from the SWEBOK. It defines the body of knowledge of the software engineering field and the practices used. The resulting 103 glossary terms have been created for our taxonomy. Table.1 shows the vocabulary of main classes and their sub-classes.

TABLE 1

VOCABULARY REPRESENTATION OF ONTOLOGY

Classes	Object Properties	Data Properties	Sub-Classes	Individuals	Data Types
System	has Requirement, ConsistsOf Context andResource, has View and System Status	name, system domain, programming language, estimated completion time	Requirement, Context, Resource	Budget and Planning System	xsd: string

Classes	Object Properties	Data Properties	Sub-Classes	Individuals	Data Types
Requirement	has Source, Can be Functional Requirement, and Non-Functional Requirement	requirement priority	Functional Requirement, Non-Functional Requirement	Report, Lab-Level	xsd : string
Stakeholder	has influence on Requirement	name, address, age, email	Customer, Supplier, User		xsd : string
Source	Can be Customer, Can be User, Can be Document, Can be Existing System	source name, source location	Document, Existing System	Requirements Definition Document, Oracle Implementation, etc.	xsd : string
Non-Functional Requirement		weight, non-functional priority	Architectural, Development, Quality of Service	Bottlenecks, Security Role, etc.	xsd : string, xsd :int
User	has View, states Requirement	status, type, name, address, email	Admin, End User	Service Admin, View User, Power User, etc.	xsd : string
Functional Requirement	Incorporates Functional Requirement, incorporates Non-Functional Requirement	functional priority	Report, Action, View, Request	User View, Portal, Resource Requests, Monthly Reports, etc.	xsd : string

C. Taxonomy

Taxonomies have proven successful in resolving several problems in computerized systems used in different sciences. They have also been used in software requirements analysis to understand several concepts and to resolve particular problems[32]. They provide basis for the implementation of ontology. For our taxonomy, several published taxonomies on requirements engineering were studied, analyzed and used. The

most difficult task was the placements for NFRs (non-functional requirements) due to different views in software engineering by several researchers. Among the most popular or relevant was an information system requirements taxonomy [33] by Chen et al.

Afreen et al. [34]classifies NFRs in conflicted NFRs that can help understand the relations and the conflicts. In other words, it is important to establish the properties of the object at a later stage and the disjoints to define which entities are different or related to each other. Another taxonomy by Odeh et al. [22]based on a service-oriented view of software requirements was more appropriate to our scenario, and was therefore used for the placements of the main classes of NFRs although not all of its structure was used. Therefore, we took three of the main classes as shown here: QoS, Development and Architectural (Constraints) for the classification of NFRs in our taxonomy. Another confusion was the exact number of classes because their previous standard (i.e. ISO-9126) has been replaced now with the ISO-25010 standard. And both the standards are slightly different with the removal of some classes and an addition of some new classes in the new standard. Some of the weaknesses of ISO-9126 standard for NFRs are reported in literature[35].For this purpose, we removed all the sub-classes not included in the latest standard. Similarly, we added the new sub-classes to the placements where they were exactly required. Mostly, privacy and security are considered as a single entity. But according to [36]confidentiality or security is erroneously equated with privacy by some security practitioners. So, it is established that security and privacy are two different entities. Therefore, we considered security and privacy as different sub-classes of Quality of Service class in our taxonomy.

1) Stakeholder Taxonomy: The stakeholder sub-class of person class in the ontology is divided into three sub-classes, namely, customer, user and supplier as shown in Fig. 1. The customer can be any individual or organization that requires software whereas, the supplier is the team of requirements engineers, software developers and project managers. The User sub-class has two types of users; Admin and the End User.

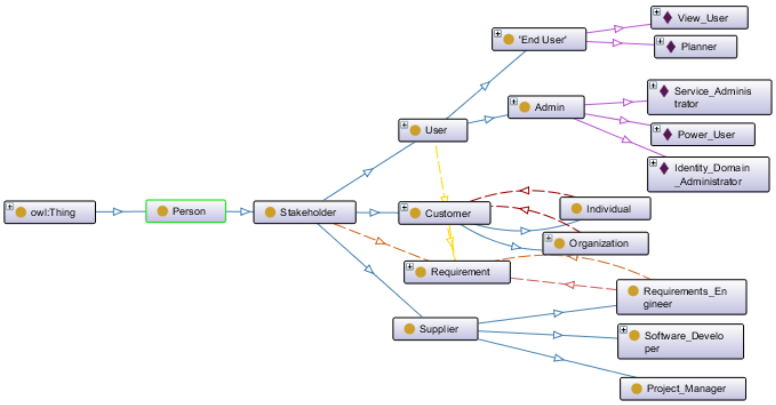

Fig. 1. Stakeholder Taxonomy

2) System Taxonomy: System sub-class consists of three sub-classes namely context, requirement and resource, as shown in Fig. 2. The context sub-class can be changed according to the system. The requirement is further divided into Functional and Non-functional Requirement sub-classes. The Resource can be Hardware, Software and Data.

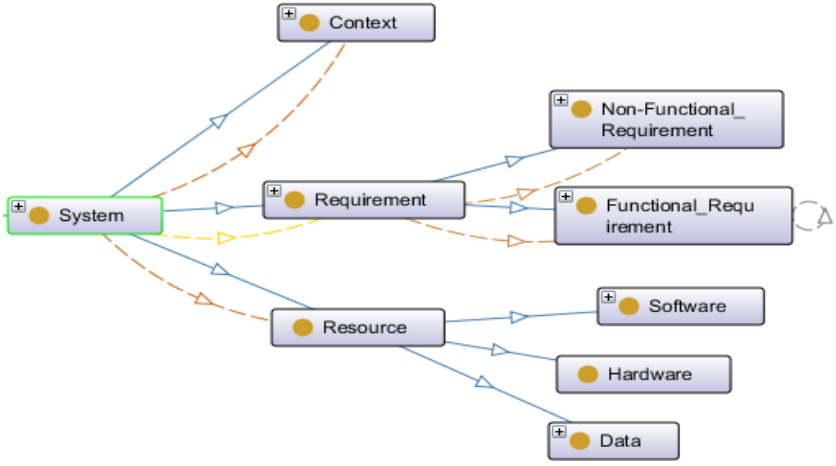

Fig. 2. System Taxonomy

3) Functional Requirement Taxonomy: The Functional Requirement class has some of the generic sub-classes like Action, View, and Report. that can be reused in many scenarios. This

class can be altered to create other sub-classes to suit the exact scenario like we have created Workflow, Forecast, and Calendar. shown in Fig. 3. The individuals shown here are also representing the requirements of a particular system under consideration.

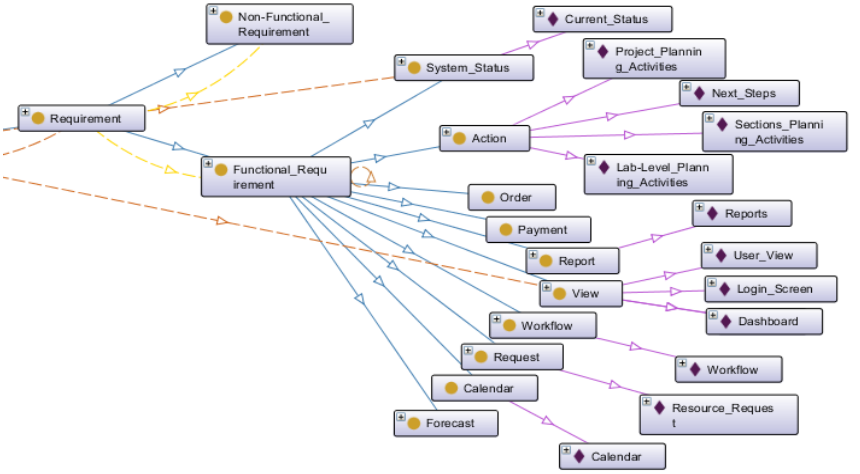

Fig. 3. Functional Requirement Taxonomy

4) Non-functional Requirement Taxonomy: This is the largest class in our ontology consisting of all the ISO-25010 standard quality constraints or non-functional requirements divided into three sub-classes namely, Architectural, Quality of Service, and Development, shown in Fig. 4. They can be used to create object properties with other functional requirement classes in ontology to define exact system requirements.

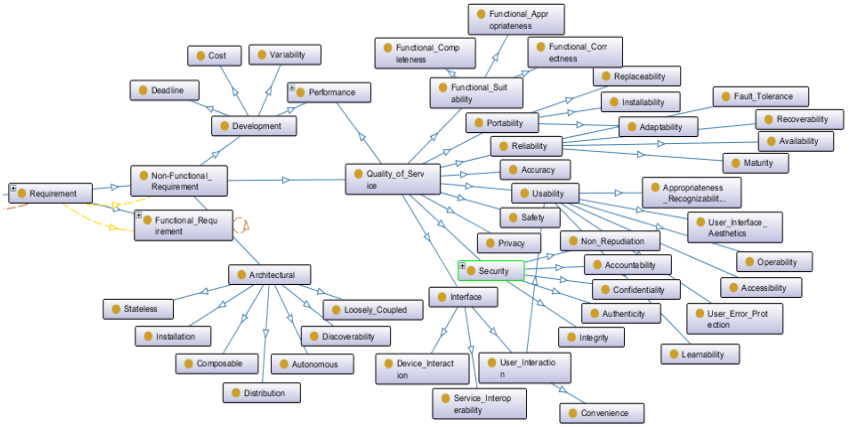

Fig. 4. Non-functional Requirement Taxonomy

D. Semantic Relations

Most of the semantic relations of requirements specification ontology concepts and considered scenarios are shown in Fig. 5. These concepts are shown as their positions in the taxonomy.

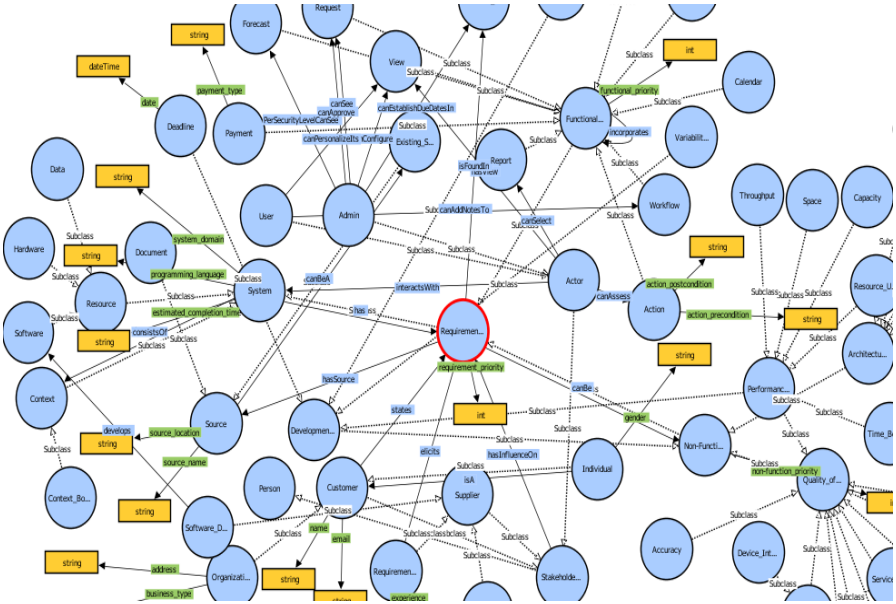

Fig. 5. Semantic relations in ontology

E. Scenario Constraints

In the scenario, view user is an end user individual. Functions of this role are defined in the system, CanDrillDownFrom Hierarchical Data, PerSecurityLevelCanSee Forecasts, has User View, has Security Role, CanSelect Reports. The restrictions are represented as negative object property assertions in Fig. 6 which state that the View UserCanAssess Bottlenecks or Current Status, CanSee Resource Request, CanApprove Resource Request, CanEstablishDueDatesIn Calendar, CanConfigure User View, CanConfigure Security Role and CanAddNotesTo Workflow. These negative object property assertions are all the restrictions on View User or the functions that this user cannot perform.

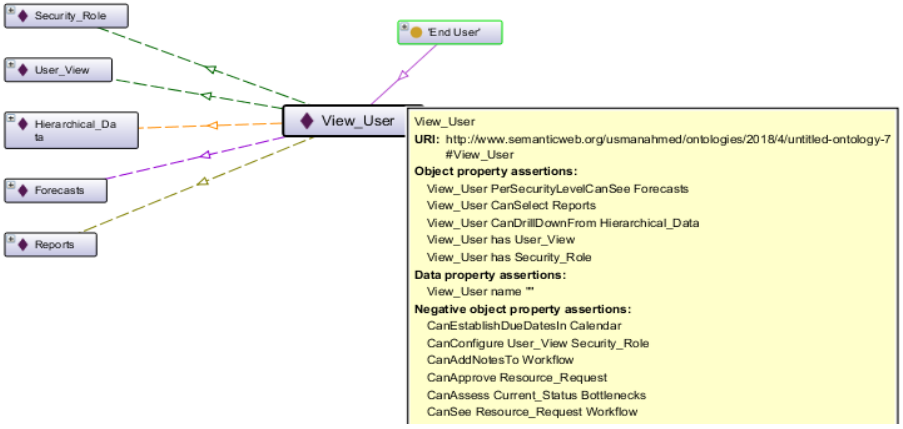

Fig. 6. Constraints of view user

The Planner is also an End User individual in the scenario. In Fig.7 functions of this role are defined as, CanSee Workflow, CanAddNotesTo Workflow, PerSecurityLevelCanSee Forecasts, CanSelect Reports, CanAssess Next Steps, CanAssess Current Status, CanPersonalizeIts User View, has Security Role, CanDrillDownFrom Hierarchical Data, has User View, CanAssess Bottlenecks. Whereas, the constraints and restrictions are: CanConfigure Security Role or User View, CanSee Resource Request and CanApprove Resource Request.

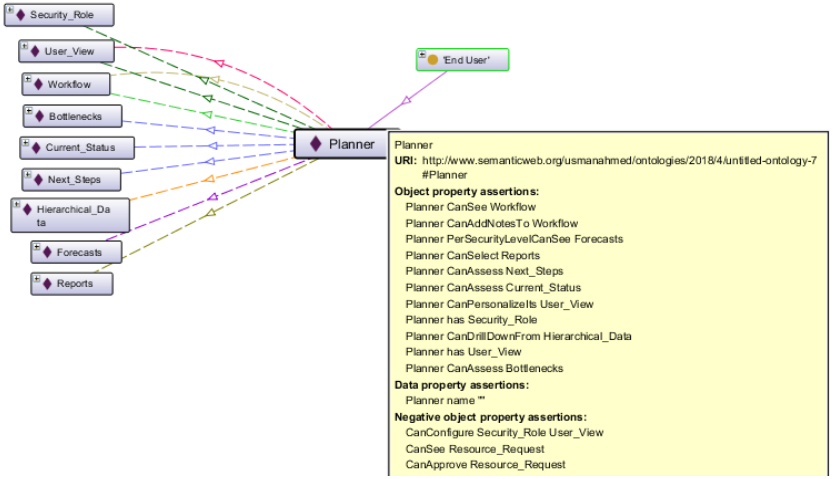

Fig. 7. Constraints of Planner

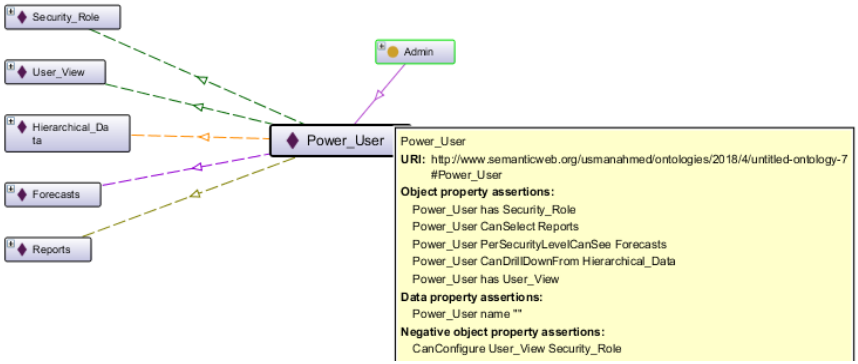

Fig. 8. Constraints for Power User

Power User, in Figure 8, is the individual of Admin class and its functions are defined as: has Security Role, CanSelect Reports, PerSecurityLevelCanSee Forecasts, CanDrillDownFrom Hierarchical Data and has User View. On the contrary, the restrictions are: CanConfigure User View and CanConfigure Security Role. Similarly, all the roles in ontology are defined as the requirements of the system.

IV. EXPERIMENT AND RESULTS

Protégé is a tool that helps researchers build ontologies to support expert systems [37]. To validate the ontologies, it provides third-party reasoners like HermiT, FaCT++ and others [38]. Reasoners provide validation of the ontologies by knowledge discovery and by detecting and finding inconsistencies or contradictions. These reasoners are based on mathematical models. They provide logical deductions based on inference rules that are defined and specified in description logic. They either use forward or backward chaining to perform inference [39]. We have used HermiT and Pellet reasoners in our research. HermiT is based upon hypertableau calculus and provides reasoning for OWL based ontology files.

The developed requirement specification ontology formally defines the software requirements. Inconsistencies can be detected and prevented. Therefore, it answers our RQ-2 by defining the classes Admin, End User, Functional Requirement, Non-Functional Requirement and their individuals and then performing reasoning on them.

A. SPARQL Validation

SPARQL is a SQL like query language that is used with ontologies. For our validation, we made a few questions to retrieve different requirements in different manners. Queries are used to answer questions and received in the form of data retrieved from our ontology. The queries are written in the DL Query portion in Protégé. They can be used to extract requirements from the ontology. In the following figures, we have presented different style of software requirements elicitation according to software engineering team. In Fig. 9, information is retrieved from requirements specification ontology on all the users who should have a user view in the system.

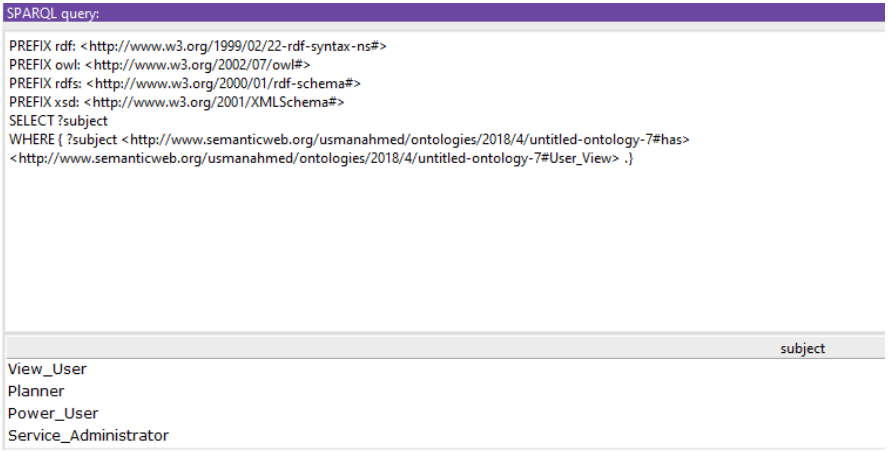

Fig. 9. Users having User View Requirement

SPARQL query extracts all individuals having a user view that are, View User, Planner, Power User and Service Administrator. Now let's consider that the software engineering team needs to extract requirements related to a particular functionality e.g. all roles that can approve the resource requests in the system.

Fig. 10 shows that only one role, the Identity Domain Administrator in the system CanApprove the Resource Requests. In this way we can get requirements related to a single functionality or object property. Now let's consider that all requirements related to a particular user are required. Then such applied SPARQL query has yielded the following result.

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?subject
WHERE { ?subject <http://www.semanticweb.org/usmanahmed/ontologies/2018/4/untitled-ontology-7#CanApprove>
<http://www.semanticweb.org/usmanahmed/ontologies/2018/4/untitled-ontology-7#Resource_Request> .}
```

subject
Identity_Domain_Administrator

Fig. 10. Extracting Users for a Particular Functionality

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?predicate ?object
WHERE {<http://www.semanticweb.org/usmanahmed/ontologies/2018/4/untitled-ontology-7#Service_Administrator> ?predicate  ?object .}
```

predicate	object
rdf:type	owl:NamedIndividual
has	Security_Role
rdf:type	Admin
CanAssess	Bottlenecks
CanDrillDownFrom	Hierarchical_Data
CanAssess	Next_Steps
CanAssess	Current_Status
CanAddNotesTo	Workflow
PerSecurityLevelCanSee	Forecasts
CanEstablishDueDatesIn	Calendar
CanSee	Workflow
has	User_View
CanSelect	Reports

Fig. 11. All Requirements of a User

Therefore, in Fig. 11, all the requirements related to Service_Administrator -- has Security Role, has User View, CanAssess Bottlenecks, Next Steps and Current Status, CanDrillDownFrom Hierarchical Data,CanAddNotesTo Workflow, CanSee Workflow, PerSecurityLevelCanSee Forecasts, CanEstablishDueDatesIn Calendar and CanSelect Reports -- are retrieved using a simple SPARQL query. In other words, we have retrieved all its requirements for

the system under consideration. Lastly, we are interested in extracting requirements related to a single individual like Bottlenecks.

In Fig. 12, SPARQL extracts all users and their functionalities related to the Bottlenecks individual, Planner and Service Administrator CanAssess Bottlenecks.

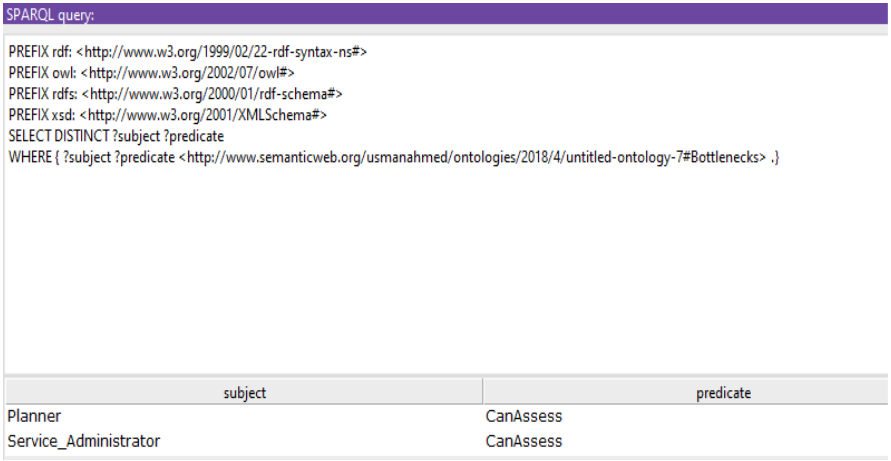

Fig. 12. Requirements related to Bottlenecks

This concludes our section of SPARQL validation and proves our RQ-1 and RQ-3 defined in the beginning. Therefore, results indicated that the software requirements can be defined through ontologies and extracted from ontology through SPARQL to meet the needs of the software engineering team.

V. CONCLUSION

Software requirements specification process creates ambiguous software requirements specification document (SRS) which can result in failure of the entire project due to the ambiguous requirements. The notion that ontologies are used to specify knowledge in a machine-processable form seems promising keeping in view its utilization in computers, medical sciences, and many other domains. This form of specification also promises to remove ambiguities that occur in natural languages. However, the current approaches of translating paper-based SRS to ontology are complex and time-consuming since they need to deal with ambiguous requirements. Our approach of semantically specifying software requirements

using a generic software requirements specification upper ontology minimizes the complexities that occur in translating paper-based SRS in ontologies. It creates a validated knowledge base of semantically specified requirements that can be used to access them, conveniently. Additionally, as it is not restricted to a single domain, generic ontology can be used in many scenarios to capture a particular system's requirements to support other software engineering phases as well. The upper ontology uses a glossary of common concepts needed in specifying requirements related to the users, as well as functional and non-functional requirements and system components to create a domain ontology of a particular system as upper ontology's individuals.

In this paper, we specified ten different requirements for budget and planning system of a state physics lab, including multiple user types, functionalities and constraints. For this purpose, HermiT and Pellet reasoners were employed to verify relationships and constraints defined in the ontology. SPARQL was used to validate our competency questions. In the future, we intend to expand this ontology and apply it to multiple systems for the specification of their requirements.

REFERENCES

[1] Sayed Mehdi Hejazi Dehaghani and N. Hajrahimi, "Which factors affect software projects maintenance cost more?," *Acta Informatica Medica*, vol. 21, no. 1, p. 63, 2013.

[2] D. M. Fernández *et al.*, "Naming the pain in requirements engineering," *Empir Software Eng*, vol. 22, no. 5, pp. 2298–2338, Oct. 2017, doi: 10.1007/s10664-016-9451-7.

[3] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *Journal of Systems and Software*, vol. 82, no. 6, pp. 981–992, Jun. 2009, doi: 10.1016/j.jss.2008.12.036.

[4] U. Ahmed, "A review on knowledge management in requirements engineering," in *2018 International Conference on Engineering and Emerging Technologies (ICEET)*, Feb. 2018, pp. 1–5. doi: 10.1109/ICEET1.2018.8338650.

[5] L. Zhao *et al.*, "Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study," *arXiv e-prints*, vol. 2004, p. arXiv:2004.01099, Apr. 2020.

[6] A. Ferrari *et al.*, "Detecting requirements defects with NLP patterns: an industrial experience in the railway domain," *Empir Software Eng*, vol. 23, no. 6, pp. 3684–3733, Dec. 2018, doi: 10.1007/s10664-018-9596-7.

[7] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with Requirements Smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, Jan. 2017, doi: 10.1016/j.jss.2016.02.047.

[8] D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry, "Reducing Ambiguities in Requirements Specifications Via Automatically Created Object-Oriented Models," in *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, Sep. 2007, pp. 103–124. doi: 10.1007/978-3-540-89778-1_10.

[9] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner, "ARSENAL: Automatic Requirements Specification Extraction from Natural Language," in *NASA Formal Methods*, Jun. 2016, pp. 41–46. doi: 10.1007/978-3-319-40648-0_4.

[10] V. Castaneda, L. Ballejos, M. L. Caliusco, and M. R. Galli, "The Use of Ontologies in Requirements Engineering," *Global Journal of Research In Engineering*, vol. 10, no. 6, Nov. 2010, Accessed: Jan. 27, 2018. [Online]. Available: https://www.engineeringresearch.org/index.php/GJRE/article/view/76

[11] F. Ruiz and J. R. Hilera, "Using Ontologies in Software Engineering and Technology," in *Ontologies for Software Engineering and Software Technology*, Springer, Berlin, Heidelberg, 2006, pp. 49–102. doi: 10.1007/3-540-34518-3_2.

[12] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, 1st ed. Wiley Publishing, 1998.

[13] V. Gervasi *et al.*, "Unpacking Tacit Knowledge for Requirements Engineering," in *Managing Requirements Knowledge*, Springer, Berlin, Heidelberg, 2013, pp. 23–47. doi: 10.1007/978-3-642-34419-0_2.

[14] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[15] D. Fensel, "Ontologies," in *Ontologies*, Springer, Berlin, Heidelberg, 2001, pp. 11–18. doi: 10.1007/978-3-662-04396-7_2.

[16] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, Jun. 1993, doi: 10.1006/knac.1993.1008.

[17] V. Tarasov, H. Tan, M. Ismail, A. Adlemo, and M. Johansson, "Application of Inference Rules to a Software Requirements Ontology to Generate Software Test Cases," in *OWL: Experiences and Directions – Reasoner Evaluation*, Springer, Cham, 2016, pp. 82–94. doi: 10.1007/978-3-319-54627-8_7.

[18] I. Dubielewicz, B. Hnatkowska, Z. Huzar, and L. Tuzinkiewicz, "Domain Modeling Based on Requirements Specification and Ontology," in *Software Engineering: Challenges and Solutions*, Springer, Cham, 2017, pp. 31–45. doi: 10.1007/978-3-319-43606-7_3.

[19] 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*, Springer, Berlin, Heidelberg, 2006, pp. 103–121. doi: 10.1007/3-540-34518-3_3.

[20] C. Calero and M. Piattini, "An Ontological Approach to SQL:2003," in *Ontologies for Software Engineering and Software Technology*, Springer, Berlin, Heidelberg, 2006, pp. 197–215. doi: 10.1007/3-540-34518-3_7.

[21] D. Dermeval *et al.*, "Applications of ontologies in requirements engineering:

a systematic review of the literature," *Requirements Eng*, vol. 21, no. 4, pp. 405–437, Nov. 2016, doi: 10.1007/s00766-015-0222-6.

[22] Y. Odeh and M. Odeh, "A NEW CLASSIFICATION OF NON-FUNCTIONAL REQUIREMENTS FOR SERVICE-ORIENTED SOFTWARE ENINEERING," May 2018.

[23] H. Kaiya and M. Saeki, "Using Domain Ontology as Domain Knowledge for Requirements Elicitation," in *14th IEEE International Requirements Engineering Conference (RE'06)*, Sep. 2006, pp. 189–198. doi: 10.1109/RE.2006.72.

[24] M. Ismail, "Ontology Learning from Software Requirements Specification (SRS)," in *Knowledge Engineering and Knowledge Management*, Nov. 2016, pp. 251–255. doi: 10.1007/978-3-319-58694-6_39.

[25] T. Avdeenko and N. Pustovalova, "The ontology-based approach to support the completeness and consistency of the requirements specification," in *2015 International Siberian Conference on Control and Communications (SIBCON)*, May 2015, pp. 1–4. doi: 10.1109/SIBCON.2015.7147184.

[26] N. Mahmud, C. Seceleanu, and O. Ljungkrantz, "ReSA: An ontology-based requirement specification language tailored to automotive systems," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Jun. 2015, pp. 1–10. doi: 10.1109/SIES.2015.7185035.

[27] L. Zong-yong, W. Zhi-xue, Y. Ying-ying, W. Yue, and L. Ying, "Towards a Multiple Ontology Framework for Requirements Elicitation and Reuse," in *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, Jul. 2007, vol. 1, pp. 189–195. doi: 10.1109/COMPSAC.2007.216.

[28] A. P. Yanuarifiani, F.-F. Chua, and G.-Y. Chan, "Feasibility Analysis of a Rule-Based Ontology Framework (ROF) for Auto-Generation of Requirements Specification," in *2020 IEEE*

2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), Sep. 2020, pp. 1–6. doi: 10.1109/IICAIET49801.2020.9257838.

[29] G. Loniewski, E. Insfran, and S. Abrahão, "A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development," in *Model Driven Engineering Languages and Systems*, Oct. 2010, pp. 213–227. doi: 10.1007/978-3-642-16129-2_16.

[30] L. van Rooijen, F. S. Bäumer, M. C. Platenius, M. Geierhos, H. Hamann, and G. Engels, "From User Demand to Software Service: Using Machine Learning to Automate the Requirements Specification Process," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, Sep. 2017, pp. 379–385. doi: 10.1109/REW.2017.26.

[31] M. Grüninger and M. S. Fox, "Methodology for the Design and Evaluation of Ontologies," 1995.

[32] J. D. Palmer, Y. Liang, and L. Want, "Classification as an Approach To Requirements Analysis," *Advances in Classification Research Online*, vol. 1, no. 1, pp. 131–138, Oct. 1990, doi: 10.7152/acro.v1i1.12472.

[33] B. Chen and Q. Dong, "A Taxonomy System for Information System Requirements," in *Proceedings of the International Conference on Information Engineering and Applications (IEA) 2012*, Springer, London, 2013, pp. 633–643. doi: 10.1007/978-1-4471-4847-0_78.

[34] N. Afreen, A. Khatoon, and M. Sadiq, "A Taxonomy of Software's Non-functional Requirements," in *Proceedings of the Second International Conference on Computer and Communication Technologies*, Springer, New Delhi, 2016, pp. 47–53. doi: 10.1007/978-81-322-2517-1_6.

[35] R. E. Al-Qutaish, "An Investigation of the Weaknesses of the ISO 9126 International Standard," in *2009 Second International Conference on Computer and*

Electrical Engineering, Dec. 2009, vol. 1, pp. 275–279. doi: 10.1109/ICCEE.2009.83.

[36] S. Pearson, "Privacy, Security and Trust in Cloud Computing," in *Privacy and Security for Cloud Computing*, Springer, London, 2013, pp. 3–42. doi: 10.1007/978-1-4471-4189-1_1.

[37] J. H. Gennari *et al.*, "The evolution of Protégé: an environment for knowledge-based systems development," *International Journal of Human-Computer Studies*, vol. 58, no. 1, pp. 89–123, Jan. 2003, doi: 10.1016/S1071-5819(02)00127-1.

[38] H. Zhao, S. Zhang, and J. Zhao, "Research of Using Protégé to Build Ontology," in *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, May 2012, pp. 697–700. doi: 10.1109/ICIS.2012.126.

[39] S. Abburu, "A survey on ontology reasoners and comparison," *International Journal of Computer Applications*, vol. 57, no. 17, 2012.