Ontologies for Software Engineering: Past, Present and Future

Article *in* Indian Journal of Science and Technology · March 2016

DOI: 10.17485/ijst/2016/v9i9/71384

3 authors, including:

Akshi Kumar
Netaji Subhas University of Technology
167 PUBLICATIONS **2,899** CITATIONS
SEE PROFILE

Rohit Beniwal
Delhi Technological University
28 PUBLICATIONS **139** CITATIONS
SEE PROFILE

ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(9), DOI: 10.17485/ijst/2016/v9i9/71384, March 2016* ISSN (Online) : 0974-5645

Ontologies for Software Engineering: Past, Present and Future

M. P. S. Bhatia[1], Akshi Kumar[2] and Rohit Beniwal[1*]

[1]Division of Computer Engineering, Netaji Subhas Institute of Technology, Dwarka – 110078, Delhi, India; mpsbhatia@nsit.ac.in, rohitbeniwal@yahoo.co.in
[2]Department of Computer Engineering, Delhi Technological University, Delhi - 110042, India; akshi.kumar@gmail.com

Abstract

Background/Objectives: Research in recent years has probed integration amongst research field of Software Engineering and Semantic Web technology, addressing the advantages of applying Semantic techniques to the field of Software Engineering. Prolifically published studies have further substantiated the benefits of ontologies to the field of Software Engineering, which clearly motivate us to explore further opportunities available in this collaborated field. This paper is a survey expounding such opportunities while discussing the role of ontologies as a Software Life-Cycle support technology. **Method/Statistical Analysis:** Survey centred on providing an overview of the state-of-art of all the ontologies available for Software Engineering followed by their categorization based on software life cycle phases and their application scope. **Findings:** Characterization of ontologies as a Software Life-cycle support technology, instigated by the increasing need to investigate the interplay between Semantic Web & Software Engineering with the ultimate goal of enabling & improving Software Engineering capabilities.**Application/Improvements:** This paper discusses the practical and potential applications of ontologies in the field of Software Engineering followed by the issues and challenges that will keep this field dynamic and lively for years to come.

Keywords: Life Cycle, Ontologies, Semantic Web, Semantic Web Enabled Software Engineering, Software Engineering, Support Technology

1. Introduction

"Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries "[1]. It is a machine processable "Web of Data"[2]. It is well recognized as an effective means of improving visibility of knowledge on the Web. At the core of Semantic Web is ontology that is used to explicitly represent our conceptualizations. Ontologies are built to model a domain and support reasoning over the concepts. Ontology engineering in Semantic Web is primarily supported by languages such as RDF, RDFS and OWL[3].

Software Engineering is "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software"[4]. Software development is a multifaceted task with new challenges being imposed frequently. The problem manifolds owing to overwhelming demands of the customers, numerous participants, shorter time frames, different geographical and virtual locations of software development teams, information overload and related issues. Sharing and reusing available information saves the effort in development and maintenance of software systems. As a major operative challenge this fosters the need to explore strategic and supporting technologies.

Recent studies have publicized the collaboration among research fields of Software Engineering (SE) and Semantic Web Technology (SW), which demonstrate the benefits of integrating semantic techniques with Software Engineering[5-9]. Ontologies have emerged as a key player in this direction[7,8,10,11]. A rising trend to exploit ontologies to exchange and interconnect Software Engineering knowledge across Web has been recognized and accepted

* *Author for correspondence*

by the Software Engineering community. This team-up of Software Engineering with Sematic Web has further drawn attraction of standardization bodies and opened up new research avenues. Ontology-Driven Architecture (ODA), which is W3C's Software Engineering Best Practices Working Group's effort, tries to bring out best practices for using ontologies in Software Engineering[12]. Ontology Definition Metamodel (ODM) standard of the Object Management Group (OMG)'s[13] permits integrating web ontology languages (i.e. ontologies) into the software development process that is based on model-driven engineering principles[14]. Thus, motivation to lead from study-based research to acceptance in tool and practice is abundant.

In this paper, we characterize ontologies as a Software Life-cycle support technology, instigated by the increasing need to investigate the interplay between Semantic Web and Software Engineering with the ultimate goal of enabling and improving Software Engineering capabilities. This paper studies the role of ontologies in the field of Software Engineering in the past and present. It further discusses the practical and potential applications, followed by the issues and challenges that will keep the field dynamic and lively for years to come[15].

The rest of this paper is organized as follows: section 2 briefly discusses the core software life cycle phases followed by section 3 that demonstrate the use of ontologies as software life-cycle support technology depicting their phase-wise correspondence. Section 4 expounds ontologies and their application scope; section 5 illustrates the applications of ontologies in various Software Engineering areas; section 6 discusses the research directions for future work and finally section 7 concludes the paper.

2. Software Life Cycle Phases

The goal of this section is to describe the software life cycle phases, artifacts used and produced in them, and various modeling approaches that can be used during life cycle phases.

The most commonly used definition of software life cycle is the one given in the IEEE Standard Glossary for Software Engineering[4,16], where the software life cycle is defined as "The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase". This section discusses the imperative software life cycle phases, which are as follows:

2.1 Requirements Analysis and Specification Phase

This phase describes the requirements of a software system. The requirements describe "what" of a system, not the "how". Once the requirements are gathered, they are analyzed for their validity. The possibility of incorporating them into the system to be developed is also studied. Finally the resultant document known as Software Requirement Specification (SRS) is developed, which may act as a legal document between the developer and customer. This process of gathering requirements from user and to document them into SRS is known as requirement engineering.

The SRS should be correct, complete, consistent, unambiguous, verifiable, modifiable, and traceable. Different approaches such as viewpoints-oriented, goal-driven, and scenario based approaches, or their combinations are used for the purpose of requirement engineering[17]. Modeling approaches such as UML Use Cases and class diagrams are recommended for this phase. Formal approaches such as Petri nets[18] are also recommended by some researchers.

2.2 Design Phase

This phase transforms the SRS into a structure, which can be implemented in some programming language. Here detail designs are produced for application domain, architecture, software components, interfaces and data. UML modeling approach is used for design purpose that results into a document known as Software Design Document (SDD). Object Constraint Language (OCL), which is a textual, declarative language for specifying (more formal) constraints and queries further supplements UML[19].

The more formal the designs are, the better the chance of automatic implementation. That's why Software Engineering community is putting a lot of effort to the discipline called Model-Driven Engineering (MDE) to empower the Model-Driven development of software products[20].

2.3 Implementation Phase

This phase implements the SDD into some programming language that develops a software product. During this phase, various components are generated, which are later integrated into a final product.

2.4 Testing Phase

This phase tests the various components generated during implementation phase. It includes various levels of testing such as unit testing, integration testing, and system testing. It also includes regression testing as well as acceptance testing before the product is finally delivered to the customer. This phase is partially parallel with implementation phase. Model based testing[21] is used to check whether implementation fully adhere with respect to models.

2.5 Operation and Maintenance Phase

This phase starts when the software is finally delivered to the customer and the users start using it. "Software Maintenance is a broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities and optimization"[22]. This phase highly depends on the quality of documentation in order to completely understand the software system for maintenance purpose. That's why it is not surprising to see that 40-60% of software maintainer's time is just spent in order to understand the software system being maintained[23]. Any change in the software system should also lead to the change in the documentation manual for easy and better maintenance[24].

3. Ontologies: A Software Life-Cycle Support Technology

Pertinent literature addresses the potential advantages of applying Semantic Web techniques to enable new Software Engineering capabilities. While going through various studies, we identified twenty ontologies that have shown potential in the field of Software Engineering. The following sub-sections expound these ontologies. In each sub-section, we discuss a software activity and the related ontologies focusing on their definition and state-of-the art. Here we would also like to mention that some issues which are resolved using multiple ontologies are discussed within the scope of last ontology dealing with it. Thus the state of art for previous ontologies has been limited to avoid repetition and missing links.

3.1 Upper Ontology

Definition: Upper ontology defines very general concepts that are identical across all the knowledge domains. These knowledge domains share common concepts form upper ontology. This ontology provides the semantic interoperability to all other ontologies that lies beneath it[25,26].

State of the art: There are several ontologies available that adheres to the properties of upper ontology. Example are Bunge-Wand-Weber (BWW) ontology, Business Objects Reference Ontology (BORO), Upper Mapping and Binding Exchange Layer (UMBEL) ontology, Unified Foundational Ontology (UFO), IDEAS, WordNet, CIDOC object-oriented Conceptual Reference Model (CRM) ontology, COmmon Semantic MOdel (COSMO) ontology, Object-Centered High-level Reference (OCHRE) ontology, Gist, MarineTLO, Yet Another More Advanced Top Ontology (YAMATO), Basic Formal Ontology (BFO), Cyc, Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE), General Formal Ontology (GFO), PROTo ONtology (PROTON), Sowa's ontology, and Suggested Upper Merged Ontology (SUMO)[27]. A comparative study of last seven ontologies has been examined by[28].

3.2 Software Process Ontology

This sub-section discusses software process ontology, which describes the various processes that can be used for software development.

3.2.1 Software Process Ontology

Definition: Software process ontology defines the various software lifecycle models and their phases that are used to develop the software. Each phase can be defined by the sequence of activities. Waterfall Model, Prototype Model, Incremental Process Model, Iterative Enhancement Model, V Model, Rapid Application Development (RAD) Model, Evolutionary Development Model, Spiral Model, Agile Model, Unified Model, etc., are few examples of software life cycles models that we generally study[29-31].

State of the Art: While going through several process ontologies proposed by different authors, we found that each one has used different terminology by defining a

variable set of terms. Also, sometime the same term is used by different authors, but, has a different meaning, while different terms used by diverse authors, have same meaning. Such variation cannot be covered by one process ontology. Hence it is proposed to develop a general process ontology, which can be further extended to cover different existing process models[32]. International standards such as CMMI and ISO/ IEC 15504 can be referenced for developing such process models[29,33].

While[29,30, 31] worked upon implementing the process ontology on the software life cycle models; [34] implemented the same ontology on software project planning activities. A review on ontologies for software project management process has been done by authors[35]. Our study in this paper is limited to the life cycle models and their phases only.

3.3 Domain Ontologies

This sub-section illustrates application domain and application domain feature ontologies that can be used for specifying domain information of the software system being developed[36-39].

3.3.1 Application Domain Ontology

Definition: Application domain ontology represents the knowledge of a particular domain (for example, automobile sector domain ontology will represent the knowledge about automobile sector only) and the business information required for developing software applications in a particular domain. This ontology also defines the various relationships that exist between different concepts related to the domain. While describing relationship between different concepts, they are defined from the real world perspective[31,40-45].

State of the Art: To the best of our knowledge, we haven't seen this ontology for every domain. We found that most authors who have already built this ontology have just provided a module or only a section of it is implemented.

Literature survey further demonstrated that application domain ontology, once created, will be able to suggest elicitation questions pertaining to functional requirements, which in turn will provide help in requirement engineering.

Once all the functional requirement questions are elicitated, the application domain ontology can be used to measure the quality of requirements[46-48]. described the use of metric suite in order to measure and improve the quality of requirements. They used ontology reasoning with closed world assumption in detecting and correcting incomplete and inconsistent requirements.

According to[49] and[50], a transformation approach can be established in order to automatically generate source code from application domain ontology. Then any change in this ontology will also help in predicting the changes required in requirements specification. Hence, this ontology can serve as a base for change prediction in requirements specification[51].

3.3.2 Application Domain Feature Model Ontology

Definition: A feature is defined as a characteristic of a software system. Therefore, a software system can have lots of features. We can develop various products that share common features along with some variations. It means that all such products will have some common features along with some variations and they belong to the family of related programs. Hence, these products share common architecture with addition of few different features. This addition makes them a distinct product. Therefore a feature can be seen as an additional functionality to a given program. Feature Model is used to represent the commonalities and differences between software features in order to represent the variability within the software[52,53]. Application domain feature model ontology models the features of software in the same application domain[32].

State of the Art: According to[54], this ontology can be used in Semantic Web based approach to provide the design guidance[32]. This ontology also maintains the information features that shows which one is *mandatory, optional, alternate, or, requires, and excludes*.

Due to the large no. of features owned by software system (which could be up to a few thousand), and increased complexity of dependencies and relations between these features, the process of expeditiously insuring the correctness of the feature models representing the software system becomes difficult. Moreover the growth in size calls for creation of distributed feature models. In this regard, [55]provided a framework that represents, integrates distributed feature models and validates them using OWL and Semantic Web Rule Language (SWRL).

3.4 Requirement Ontology

This sub-section expounds system behaviour ontology, which describes the requirements specified by the

customer. Requirements are of two types: functional and non-functional. Functional requirements can be described as the sequence of actions that a system performs. Non-functional requirements represent the quality related aspects of software system. They stipulate how well the software does, what it has to do.

3.4.1 System Behaviour Ontology

Definition: System behaviour ontology models the behaviour of the software system i.e. what actions a system is going to perform under a particular scenario. It also defines the conditions that need to be met before an action performs and will tell the status of the system once the action is performed[42,56].

State of the Art: As we know that UML Use Cases are used to capture behavioural knowledge of the software system such as who interacts with it, for what purpose one interacts with it, and what will be the outcome after such interactions without knowing internal details. As far as Use Case is concerned, the system is treated as 'black box' i.e. all the information that is captured belongs to the outside of the system. It means Use Cases are used to define overall required behaviour of the software system. In this regard, [42] raised an issue to create a large library of Use Cases using system behaviour and application domain ontologies. While creating the library, we can store semantic information about Use Cases. Once the library is built, Semantic Web based approach can be used to retrieve them based on their semantic information.

3.5 Architecture and Design Ontologies

This sub-section examines software architecture, application logic, and object oriented design ontologies, which describe the architecture and design description of software system.

3.5.1 Software Architecture Ontology

Definition: Software architecture ontology models the architecture related concepts such as architecture styles (structure of the system), elements or components of the software system, element's properties and their relationship[57].

3.5.2 Application Logic Ontology

Definition: Application logic ontology can be used to define logic (reason) behind the application behaviour. It defines the reasons for a particular behaviour performed by the software system.

While System behaviour ontology treats the system as 'Black Box', Application Logic Ontology treats it as 'White Box', which means, this ontology deals with the reason behind a particular behaviour and knows how and why the system is behaving in that way. This ontology can define and model diagrams such as data flow diagram, flow chart diagram, activity diagram, sequence diagram, class diagram state chart diagram, object diagram, component, deployment diagram, etc.[58].

3.5.3 Object Oriented Design Ontology (OO Design Ontology)

Definition: OO design ontology can be used to define the vocabulary for describing object oriented design principles. This ontology includes the concepts such as *class, subclass, method or member function, interface, objects*, etc. It also includes relationships such as *inheritance, realization*, etc.[59,60].

3.6 Pattern Ontology

This sub-section discusses pattern ontology, which describes several kinds of software patterns. Software patterns capture the best practices of system design. There are three key characteristics of design pattern. First, it promotes design reuse and provides guidance. Second, it conforms to a literary style. Third, together, they define a vocabulary for discussing design[61]. Presently, there are several kinds of patterns such as Gang of Four (GoF) design pattern, workflow pattern, web application pattern, and usability pattern. Semantic web techniques can be used to formally describe the patterns[59,62].

3.6.1 Pattern Ontology

Definition: This ontology provides a library of patterns that are commonly used to create the software system. This ontology can consist of design pattern, workflow pattern, web application pattern, usability pattern, etc.[63-65].

State of the Art: [66]discussed about design pattern of ontologies. According to them, ontologies also have lifecycle phases like the softwares have. Ontologies are also designed, implemented and maintained like the softwares are. They considered 'Reusability' as one of the

most challenging and neglected area in ontology's design process. They classified the design pattern of ontologies into six categories namely Structural, Correspondence, Reasoning, Presentation, Lexico-Syntactic, and Content Ontology Design Pattern (ODP). They specially focus on Content ODP that describes the conceptual pattern rather than logical. Content ODP helps in transforming Use Cases into design solutions and discusses the relationship between them. They also set a web portal, which discusses about Content ODP known as ontologydesignpatterns.org as a part of their research work which provides in-depth detail.

[67]Also worked on Content Ontology Design Patterns (ODPs). They experimented with Content ODPs in order to show their advantages and it is found that Content ODPs are acknowledged beneficial by ontology developers. The quality of ontology improves in terms of task coverage, usability aspects, and fewer modeling mistakes when Content ODPs are reused. The ontology developers also perceived that modeling was made simpler when Content ODPs were used, but the hypothesis that it accelerates the development process wasn't sustained. Instead, it was felt that though the development process is slow at the beginning, but it results in better constructed ontologies.

Usage of Content ODP is found in different domains and for different tasks. For example, Content Ontology Design Pattern called 'Participation' is found in different domain ontologies covering enterprise models[68], legal norms[69], software management[70], biochemical pathway[71,72] and fishery techniques[73].

[74]Found the decent use of ODPs in a repository of biomedical ontologies. According to them, ODPs in biomedical ontologies may be due to age of ontologies in repository, deficiency of proper tooling, lack of user's knowledge and education about them, specificity of domain and a tension between highly logical expressivity of ODPs with the objective of maintaining low expressivity of biomedical ontologies.

In the reference of pattern ontologies, [63]provides description about Usability pattern, [75]defines ontology for Web Application pattern and [76]describes about ontology based automated discovery of Workflow pattern.

According to[32], once we create the well-defined pattern ontology, pattern training task can be facilitated by it and if any change occurs in the pattern, it will help in production of change adaptable code.

3.7 Implementation Ontologies

This subsection illustrates software artifact, object oriented source code, version, and configuration ontologies, which can be used for implementation purpose.

3.7.1 Software Artifact Ontology

Definition: Software artifact ontology defines the concepts of various artifacts generated during development of software system. These concepts allow us to classify them according to their type or formats such as text file, binary file, source code, image file, audio/video file, documentation manuals, etc.[40,41,43].

This ontology can also define concepts, which contain metadata information such as during which phase a particular artifact is produced, by whom (person) it is created and in which project it is created.

State of the Art: According to[32], once we have all the metadata information about the artifacts produced, this ontology along with application domain ontology can be used in automatic generation of documentation (up to some extent).

Metadata standards such as Dublin Core Metadata from Dublin Core Metadata Initiative[77] and IEEE learning Object Metadata[78] can be referenced for concepts containing information about metadata.

3.7.2 Object Oriented Source Code Ontology (OO Source Code Ontology)

Definition: OO source code ontology defines all the concepts related to Object Oriented Programming (OOP). OOP major concepts such as Abstraction, Inheritance, Polymorphism, and encapsulation will be defined by this ontology. It will also include concepts such as *package, class, subclass, variable or data member, member function, object*, etc.[79,80,81].

3.7.3 Version Ontology

Definition: Version ontology defines the various versions of software, which are generated till date. It establishes the relationship between various generated artifacts to a particular version. It includes the concepts such as *file, release, and revision*[79].

3.7.4 System Configuration Ontology

Definition: System configuration ontology models the characteristics of various components belonging to different versions. The Ontology prescribes the kind of software/ hardware configuration required to install the

software or which version is supported by the available configuration[81].

3.8 Documentation Ontologies

This sub-section expounds documentation and document ontologies, which can be used for software's documentation purpose.

3.8.1 Documentation Ontology

Definition: Documentation ontology defines all the documents related to software system. It contains various documents generated during various phases such as formal specification, context diagram, and data flow diagram generated during requirement analysis and specification phase; flow chart and entity relationship diagram generated during design phase; source code listing and cross-reference listing generated during implementation phase; test data and test result generated during testing phase[80-83].

State of the Art: In order to provide better documentation, Graaf[84] and [85]developed an annotating semantic wiki page with the help of light weight (with limited classes/ properties) Software Engineering ontology, which contains Dublin core data properties to allow specification of metadata. They developed the wiki page with the help of ArchiMind and OntoWiki Semantic Wiki in which the annotated text can be searched. Both Archimind and Ontowiki semantic wiki provide advantages in versioning, responsibility, accessibility, and in traceability. However, issues such as synonym and homonym remain along with spelling error, abbreviation, ambiguity, and context dependent interpretation. The only difference between their work is that author [85]provides the documentation on software requirements and architecture design part only. An exploratory study on ontology engineering approach for software architecture documentation has been conducted by De Graaf et al.[86,87].

According to [80] and [81], this documentation ontology can be used to facilitate the task of re-establishing the lost information i.e. it can provide help in traceability. In a survey done by[88], it was found that many practitioners don't document very frequently; reasons are "no time", "no budget", "no standards" and no suitable tool. As per[89], it is also due to the unwelcoming processes that are not fully integrated into the software industry. To solve such problems we can also automate the documentation process, up to some extent, by using software artifact, domain, and software architecture ontologies.

As far as automatic documentation is concerned, [90]developed a Toeska Rationale Extraction (TREx) tool that is used for software architecture documentation part only and is based on the software architecture and design ontologies. According to them, due to lack of standard and guidelines, practitioners don't know what, why, when and how to document. It is also due to schedule and budget constraint that they don't document. They implemented the TREx on a case study and compared its outcome vs. plain text documents and it was found that humans are poorer than TREx in identifying rationale, but are better at dealing with "generic" components.

3.8.2 Document Ontology

Definition: Document ontology defines the relationship between various documents. It relates an updated document to its source document. It also defines that by what relationship the two documents are linked with each other[82].

Document ontology differs from documentation ontology as the document ontology attempts to organize documents, whereas documentation ontology attempts to organize information recorded by documents. They also differ in the way the concepts and restriction are defined in both ontologies.

3.9 Quality Ontologies

This sub-section examines quality, testing, and defect ontologies, which can be used for describing various quality attributes and assessing those quality attributes that in turn enhances the quality of software system being developed. Presently, there are several tools that are widely used for software quality measurement such as Ishikawa's seven basic tools, scatter diagram, pare to diagram, histogram, control chart, run chart, and cause-effect diagram[32]. Several quality models and standards have been developed such as Capability Maturity Model (CMM), ISO 9001, and ISO/IEC 9126, which are used to improve the quality of software system being developed[33].

3.9.1 Quality Ontology

Definition: Quality ontology defines various quality attributes related to software system such as correctness, consistency and precision, robustness, simplicity,

traceability, accuracy, completeness, efficiency, testability, modularity, readability, simplicity, modifiability, expandability, portability, etc.[91].

State of the Art: According to [91],this ontology once created will be able to suggest elicitation questions related to non-functional requirements.

3.9.2 Testing Ontology

Definition: Testing ontology defines the concepts of testing such as who is the tester, in what kind of environment testing is performed, what are the various testing mechanism available, on what artifacts testing is performed, etc.[92].

This ontology can further explain testing mechanism, for example, the various testing techniques that are available such as black box testing (functional testing) and white box testing (structural testing) techniques; levels of testing such as unit testing, integration testing, and system testing; kinds of testing such as validation and verification testing.

State of the Art: According to Zhu et al.[92], this ontology can provide help in automation of testing process. As far as automation is concerned, Nasser et al.[93] presented a framework to automatically generate the executable test cases based on the ontologies on dissimilar domains with custom defined coverage rules. The framework uses various ontologies such as behavioural model ontology, domain ontology, and implementation ontology to generate the custom defined coverage criteria in order to generate the executable test suite. They also explained the limitations for practical implementation of this framework such as non-optimal redundancy checking constraint, time efficiency constraint, modeling complexity constraint, and the logic programming limitation constraint.

Paydar et al.[94] presented a theoretical roadmap for ontology based web application testing. Here they discuss that first they require to capture all the knowledge related to testing process, second they require application domain ontology in order to have all knowledge about application under test, and third and last is to develop procedures, which are required in order to automate the testing process, with the help of previously gained knowledge. While creating the roadmap, they discuss the Software Engineering Body of Knowledge (SWEBOK) that keeps knowledge such as relationship, concepts, facts, and principles about various Software Engineering topics (known as knowledge area).

Li et al.[95] used the testing ontology to automatically generate the test cases. They describe an ontology based approach for Graphical User Interface (GUI) testing in which ontology is built for GUI system. Using reverse engineering techniques based on the design of GUI ontology, all elements from source code are extracted and stored in GUI ontology. Then using the knowledge stored in GUI ontology, test case generation rules are extracted by analysing tester's experience that is based on the GUI components sequences. At last, based on test case generated rules and GUI ontology, test cases are generated. Then Li et al.[96] described the ontology based approach for reusing the already built test cases. It is found that the test case design effort is reduced and efficiency is increased as a result of their work. Dalal et al.[97] also described an ontology based approach for test case reuse, which enhances flexibility and reusability during test case generation and handles users' queries better.

To the best of our knowledge no work has been done on ontology based automatic testing, which gives researchers ample opportunity in this direction.

3.9.3 Defect Ontology

Definition: Defect ontology defines the concepts that describe the defects detected during the testing phase. It includes the concepts such as *Issue*, which defines the defect; *Person*, who detects and rectifies it; *Comment* to provide extra information; *Activity*, which is required to rectify it; *Computer System, Product, Component* to which a defect belongs[79].

This ontology can also be used for analysing detected defects. We can establish suitable metrics for the analysis purpose. With suitable metrics, severity level of the defect can be defined. Based on the above information, prioritization can be done. Higher priority means higher degree of risk to the software system, which means such a defect needs to be resolved first before mitigating others. Once the defect is detected, we can have information such as to which phase it belongs and by what method it is detected. Based on analysed information, we can also estimate the cost required to repair the detected defects.

State of the Art: As far as defect severity is concerned, Iliev et al.[98] used ontology based approach for automatic prediction (classification) of defects severity that depends on the requirements, design, coding, and configuration. They focused on the classification of defects from User Point of View rather than developer's and tester's. To do this, they picked a case study from Industry Logic a (Organization Name) and collected defects

and classification as provided by them. Then using ontology based approach, they automatically classified the defects into three levels namely Major, Medium and Minor. Further they compared their work with original classification (as provided by the organization) and found that 58% defects matched with the same severity level.

According to Kiefer et al.[79] also, this ontology can be used to detect the potential areas from the source code where defects can be found. The task of measuring defect density can also facilitated by this ontology in order to measure the quality of the software system being developed.

3.10 Maintenance Ontology

This sub-section discusses maintenance ontology, which describes various maintenance activities.

3.10.1 Software Maintenance Ontology

Definition: Software Maintenance Ontology defines the maintenance related concepts and their relationship[99]. It can define the concepts describing maintenance type such as perfective maintenance, adaptive maintenance, corrective maintenance, and any other type of maintenance; what procedures can be used to maintain the software i.e. maintenance activities; who does the maintenance. It includes the concepts like *activity* such as management activity and modification activity; *person* such as maintenance engineer and maintenance manager; *procedure* such as method, technique, and paradigm; *resource* such as maintenance human resource, client human resource, etc. It includes the relationship such as *approves, performs, trains, uses*, etc. Maintenance ontology can also be used to model various maintenance models such as Iterative enhancement Model, Reuse Oriented Model, Boehm's Model, Taute Maintenance Model, etc.[100].
State of the Art: Reverse Engineering is also done for maintenance purpose that includes activities such as discovering unknown and hidden information of a software system. Pattern ontology along with software architecture and object oriented design ontologies can be used to facilitate the task of discovering such unknown and hidden information as patterns tends to reveal the style that was used to create the software system[59].

Semantic web approaches can also be facilitated for estimation of maintenance cost. We can use models such as 'Belady and Lehman Model', Boehm Model, etc. to estimate the maintenance cost.

April et al.[99] discussed that a Semantic Web based software maintenance tool can be built. This tool will use software maintenance ontology. Hyland-Wood et al.[60] also proposed an approach for maintaining the software system using Semantic Web techniques. They built an ontology that captures metadata information of software components such as requirements documentation (both functional and non-functional), metrics, test suite and the means by which various components interact with each other. If any change is done to any software component, it will trigger all the metadata information related to that changed software component in order to understand and maintain that component. For this purpose, they encoded the metadata information in RDF graph and SPARQL queries were used over them in order to facilitate the understanding and maintenance of the software component. SPARQL Queries were built to check whether re-validation of changed software component is required in order to know that which test case has failed and to which requirement that failed test case belongs[101].

3.11 Technology Ontology

This sub-section illustrates technology ontology, which describes the various existing tool and techniques that can be used for software development.

3.11.1 Technology Ontology

Definition: Technology ontology is repository of various software development tools that can be used to develop a particular software[43,58].
State of the Art: Dinger et al.[58] illustrates that using Semantic Web environment, a tool can be generated to select a particular technology for developing the software system. Metadata information about the tool can be provided using Semantic Web techniques such as information about its application scope, information about its required hardware-software combinations, and information about properties of particular tool. Metadata information can also include that what kind of information can be generated by a particular tool[102].

4. Categorization based on Application Scope of Ontologies

The aim of this section is to classify various ontologies

based on software life cycle phases (as defined earlier) and their application scope. The categorization of ontologies based on software life cycle phases is illustrated in Figure 1.

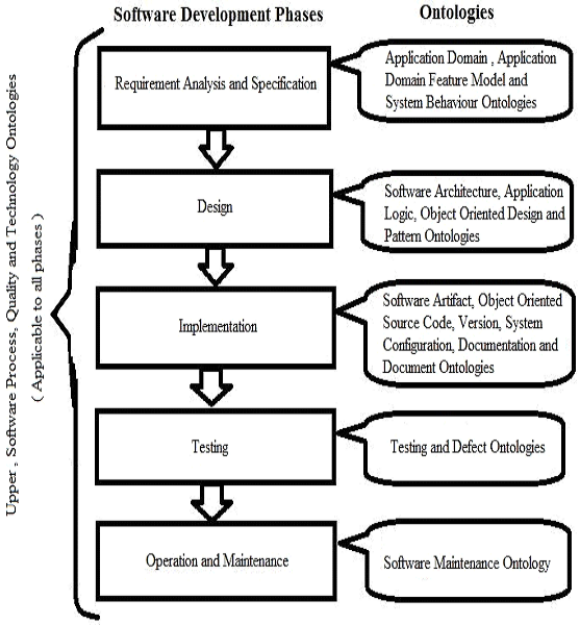

Figure 1. Categorization of ontologies based on software life cycle phases.

In accordance to the categorization based on software life cycle phases, we identified that upper, process, quality, and technology ontologies can be exploited in all the life cycle phase; application domain, application domain feature model, and system behaviour ontologies have been utilized in requirement analysis and specification phase; software architecture, application logic, object oriented design, and pattern ontologies have been utilized in the design phase; software artifact, object oriented source code, version, system configuration, documentation, and document ontologies have been used in implementation phase; testing and defect ontologies have been utilized in testing phase and software maintenance ontology exploited in operation and maintenance phase.

The categorization of ontologies based on their application scope such as generic, domain specific, object-oriented system specific, and project specific is illustrated here. Generic scope means the concerned ontologies can be used in any Software Engineering process. Domain specific scope means the concerned ontologies can only be used in the domain being modelled. Object-oriented system specific scope means the concerned ontologies can only be used in the applications where OO programming is used. Project specific scope means the concerned ontologies can only be used in their concerned projects.

Table 1 shows ontologies and their respective

Table 1. Ontologies and their application scope

Ontologies \ Application Scope	Generic	Domain Specific	Object- Oriented System Specific	Project Specific	Related Work (References)
Upper	✓				(24-26)
Software Process	✓				(27-29)
Application Domain		✓			(29, 34-45)
Feature Model		✓			(46-49)
System Behaviour	✓				(36)
Software Architecture	✓				(50)
Application Logic	✓				(51)
OO Design			✓		(52,53)
Pattern	✓				(56-60,66,68)
Software Artifact	✓				(34,35,37)
OO Source Code			✓		(71-73)
Version	✓				(71)
System Configuration				✓	(73)
Documentation				✓	(72-78,81)
Document	✓				(74)
Quality	✓				(82)
Testing	✓				(83-88)
Defect	✓				(71,89)
Maintenance	✓				(53,90,91)
Technology	✓				(37,51)

application scope to which they belong. A ✓ (Tick) mark against the ontology shows its corresponding application scope.

5. Application of Ontologies in Various Software Engineering Areas

This section illustrates the applications of ontologies in various Software Engineering areas. Through this literature survey, we found applications of ontologies in various Software Engineering areas such as requirement engineering, software design, implementation, testing, maintenance, documentation, traceability, change control, quality, reuse, and technology selection and process support along with 'Reverse Engineering' where very little work has been done limited only to web applications.

Table 2 shows ontologies and Software Engineering are as which they address. A ✓ (Tick) mark against the ontology shows that its application is found in corresponding Software Engineering area. However, a ? (Question) mark against the ontology shows that its application is not found in existing literature, but can be used in the future.

At last, it can be summarised that if we traverse horizontally in the table, we will find that which ontology has addressed and can also address what Software Engineering areas. Further if we traverse vertically in the table, we will find that which Software Engineering area is addressed and can also be addressed by what combinations of ontologies.

6. Future Work (Open Issues)

The following open issues have been identified across

Table 2. Application of ontologies in various Software Engineering areas

Ontologies \ Software Engineering Area	Requirement Engineering	Software Design	Implementation	Testing	Maintenance	Documentation	Traceability	Change Control	Quality	Reuse	Process Support	Technology Selection and	Reverse Engineering
Software Process									✓		✓		
Application Domain	✓		✓	?	✓	✓		✓	✓	✓			✓
Feature Model		✓							?				
System Behaviour	✓		✓	?									
Software Architecture		?				✓				✓			?
Application Logic		?									✓		
OO Design		?			✓								?
Pattern		✓			✓			✓	✓	?			?
Software Artifact					?	✓			?	✓	✓		
OO Source Code			✓	✓	✓		✓	✓	✓				
Version					✓			✓	✓				
System Configuration			✓										
Documentation					✓	?	✓			?	✓		
Document										?			
Quality	?				✓				?				
Testing	✓			✓	✓				✓	✓			
Defect					✓				✓				
Maintenance					✓								
Technology											✓		

literature and clearly emerged as potential opportunities for future work.

1. Potential opportunities in Requirement Engineering: Using domain and quality ontologies to improve overall requirement engineering process (e.g. removing ambiguities) and assessing it with well-defined metrics can be a future work.

2. Potential opportunities in Software Design: Using software architecture, application logic, object oriented design and pattern ontologies to publicize architectural styles and information related to existing modeling languages. This may improve spreading architecture and design knowledge.

3. Potential opportunities in Software Testing: Work exhibiting semi or fully automatic testing of software using ontologies has though been presented[93-95], but much of the work is just theoretical or framework based, where real implementation is yet to be done. Also, another possible work in software testing is the use of domain, system behaviour and implementation ontologies can be explored to generate executable test cases with well-defined domain/system and custom specific coverage criteria rules.

4. Potential opportunities in Software Maintenance: The potential use of ontologies can be explored diversely to enrich the maintenance process as follows:

- Using application domain, software artifact, version, quality, testing and maintenance ontologies can be used to represent software system components and information about them (information such as functional and non-functional requirements, metrics, test cases and how various components interact with each other) to facilitate understanding and maintenance of software system.
- Improving semantic annotation process of software models and source code, integration of ontologies and meta-modeling architectures, and all-inclusive traceability model of software artifacts can be used to better understand software knowledge artifacts.
- Using application domain, object oriented design and object oriented source code ontologies to automatically detect and update out-dated requirements during maintenance phase can be a potential future work. A framework has been proposed for the above issue by Bhatia et al.[103] , but real implementation is still to be done.

5. Potential opportunities in Software Documentation: Automation of documentation process using Application Domain (AD), Software Architecture (SArch), Software Artifact (SArt), and Documentation ontologies can be a possible future work. AD, SArch, and SArt ontologies can be merged together to develop documentation ontology. Any missing information in documentation ontology can be extracted from data corpus, if available, so that it has all necessary details. Then a transformation approach can be established to convert the documentation ontology into text/html document. Though a framework for this has been proposed by Bhatia et al.[83], but implementation is yet to be done.

6. Potential opportunities to improve Software Quality: Using feature model and quality ontologies to detect inconsistencies in feature models along with reasons for inconsistencies from usability point can be explored. Also, detecting conflicts in feature models for accurate decision and error free configuration, creating repository of feature model scan be possibly investigated.

7. Potential opportunities to Reuse software related knowledge: Reuse of all ontology design patterns to show their usefulness in terms of quality and maintainability issues can be a possible direction of future work. Assessment of design patterns to improve their quality using pattern and quality ontologies can also be examined. Further, Reuse of documentation ontology to create documents for software that reuses source code can also be explored.

8. Potential opportunities in Reverse Engineering: Using software architecture, object oriented design and pattern ontologies to facilitate the discovery of unknown and hidden information for the purpose of reverse engineering can be a prospective direction of work. Most of the work in this area is limited to ontology based web application reverse engineering[104-107], whereas the scope of using Semantic Web in conventional software reverse engineering is not yet examined.

7. Conclusion

This research paper has discussed about the past, present and future of ontologies for Software Engineering. The existing ontologies have been defined, their state-of-art reviewed to an extent where the past and present applications have been investigated. Also, a detailed description on open challenges that keep this area dynamic and alive have been analysed. We have attempted to provide a categorization of ontologies based on software life cycle phases and their application scope.

8. References

1. W3C Semantic Web [Internet]. 2016 Jan 30; Available from: http://www.w3.org/2001/sw.
2. Berners-Lee T, Hendler J, Lassila O. The semantic web. Scientific american. 2001 May 17; 284(5):28-37.
3. Ding L, Kolari P, Ding Z, Avancha S. Using ontologies in the semantic web: A survey. In Ontologies. Springer US; 2007 Jan 1.p. 79–113.
4. September A. IEEE standard glossary of software engineering terminology. Office. 1990; 121990(1):1.
5. Pan JZ, Zhao Y. Semantic Web Enabled Software Engineering. IOS Press; 2014 Jul 16.
6. Gröner G, Pan JZ, Zhao Y, Kendall EF, Stojanovic L. Introduction to the Proceedings of the 9th International Workshop on Semantic Web Enabled Software Engineering (SWESE) 2013. InService-Oriented Computing–ICSOC 2013 Workshops. Springer International Publishing; 2014 Jan 1.p. 223–4.
7. Ilyas QM. Ontology Augmented Software Engineering. Software Development Techniques for Constructive Information Systems Design. 2013 Mar 31:406.
8. Aßmann U, Zivkovic S, Miksa K, Siegemund K, Bartho A, Rahmani T, Thomas E, Pan JZ. Ontology-Guided Software Engineering in the MOST Workbench. InOntology-Driven Software Development. Springer Berlin Heidelberg; 2013 Jan 1.p. 293–318.
9. Schügerl P. Semantic Web Enabled Software Engineering (Doctoral dissertation, Concordia University).
10. Isotani S, Ibert Bittencourt I, Francine Barbosa E, Dermeval D, Oscar Araujo Paiva R. Ontology Driven Software Engineering: A Review of Challenges and Opportunities. Latin America Transactions, IEEE (Revista IEEE America Latina). 2015 Mar; 13(3):863-9.
11. Pan JZ, Staab S, Aßmann U, Ebert J, Zhao Y. Ontology-driven software development. Springer Science & Business Media; 2012 Dec 22.
12. Tetlow P, Pan JZ, Oberle D, Wallace E, Uschold M, Kendall E. Ontology driven architectures and potential uses of the semantic web in systems and software engineering. W3C Working Draft. 2005 Oct.
13. OMG ODM [Internet]. Available from: http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf. 2016 Jan 30.
14. Berardi D, Calvanese D, De Giacomo G. Reasoning on UML class diagrams. Artificial Intelligence. 2005 Oct 31; 168(1):70-118.
15. Kumar A, Sebastian TM. Sentiment analysis: A perspective on its past, present and future. International Journal of Intelligent Systems and Applications (IJISA). 2012 Sep 1;4(10):1.
16. Gašević D, Kaviani N, Milanović M. Ontologies and software engineering. InHandbook on Ontologies. Springer Berlin Heidelberg; 2009 Jan 1. p. 593–615.
17. Lee SW, Gandhi R. Ontology-based active requirements engineering framework. InSoftware Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific. IEEE; 2005 Dec 15 .p. 8.
18. Jørgensen JB, Bossen C. Executable use cases: requirements for a pervasive health care system. Software, IEEE. 2004 Mar; 21(2):34-41.
19. Rumbaugh J, Jacobson I, Booch G. Unified Modeling Language Reference Manual, The. Pearson Higher Education; 2004 Jul 1.
20. Favre JM. Foundations of Meta-Pyramids: Languages vs. Metamodels-Episode II: Story of Thotus the Baboon1. Language Engineering for Model-Driven Software Development. 2004 Feb; 4101.
21. Apfelbaum L, Doyle J. Model based testing. InSoftware Quality Week Conference 1997 May. p. 296–300.
22. Lamb DA. Software engineering: planning for change. Prentice-Hall, Inc.; 1988 Jan 1.
23. Pfleeger SL, Atlee JM. Software engineering: theory and practice. Pearson Education India; 1998 Mar.
24. Chitra S, Kalpana B. Optimum session interval based on particle swarm optimization for generating personalized ontology. Indian Journal of Science and Technology. 2014 Aug 25; 7(8):1137-43.
25. Upper Ontology [Internet]. 2016 Jan 30; Available from: http://en.wikipedia.org/wiki/Upper_ontology.
26. Chauhan A, Vijayakumar V, Ragala R. Towards a Multi-level Upper Ontology/foundation Ontology Framework as Background Knowledge for Ontology Matching Problem. Procedia Computer Science. 2015 Dec 31; 50:631-4.
27. Priya M, Kumar CA. A Survey of State of the Art of Ontology Construction and Merging using Formal Concept Analysis. Indian Journal of Science and Technology. 2015 Sep 14; 8(24).
28. Mascardi V, Cordì V, Rosso P. A Comparison of Upper Ontologies. InWOA 2007 Sep 24. p. 55–64.
29. Hawker JS, Ma H, Smith RK. A Web-based process and process models to find and deliver information to improve the quality of flight software. The 22nd In Digital Avionics Systems Conference DASC'03. IEEE; 2003 Oct 12; 1:3-B.
30. Liao L, Qu Y, Leung HK. A software process ontology and its application. 2003.
31. Thaddeus S, Raja SK. Ontology-driven Model for Knowledge-Based Software Engineering. In SEKE 2006 Jul. p. 337–42).
32. Zhao Y, Dong J, Peng T. Ontology classification for semantic-web-based software engineering. Services Computing, IEEE Transactions on. 2009 Oct; 2(4):303-17.
33. Henderson-Sellers B, Gonzalez-Perez C, McBride T, Low G. An ontology for ISO software engineering standards: 1) Creating the infrastructure. Computer Standards & Interfaces. 2014 Mar 31; 36(3):563-76.
34. de Oliveira Bringuente AC, de Almeida Falbo R, Guizzardi G. Using a foundational ontology for reengineering a software process ontology. Journal of Information and Data Management. 2011 Sep 13; 2(3):511.
35. Fitsilis P, Gerogiannis V, Anthopoulos L. Ontologies for Software Project Management: A Review. Journal of Software Engineering and Applications. 2014 Dec 18; 7(13):1096.

36. Chung H, Lee S, Kim J. Learning Concept Sequencing through Semantic-based Syllabus Design and Integration. Indian Journal of Science and Technology. 2015 Aug 18; 8(18).
37. Viniba V, Sairam N. A Hybrid Layered Approach for Ontology Matching. Indian Journal of Science and Technology. 2015 Aug 1; 8(17):1.
38. Oh MS, Son YH, Lee KC. An Ontology System for Interoperation between DDS and HLA. Indian Journal of Science and Technology. 2015 Oct 1; 8(27):1.
39. Razak SH, Eri ZD, Abdullah R, Murad MA. Ontological Model of Virtual Community of Practice (VCoP) Participation: a Case of Research Group Community in Higher Learning Institution. Indian Journal of Science and Technology. 2013; 6(10):5307-15.
40. Ambrosio AP, De Santos DC, De Lucena FN, Silva JC. Software engineering documentation: an ontology-based approach. 2004. Proceedings in Web Media and LA-Web. IEEE; 2004 Oct 12. p. 38–40.
41. Antunes B, Gomes P, Seco N. SRS: a software reuse system based on the semantic web. In 3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE) 2007 Jun.
42. Caralt JC, Kim JW. Ontology driven requirements query. In System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on 2007 Jan (pp. 197c-197c). IEEE.
43. Happel HJ, Korthaus A, Seedorf S, Tomczyk P. KOntoR: an ontology-enabled approach to software reuse. Proceedings of the 18th International Conference On Software Engineering And Knowledge Engineering; 2006.
44. Henderson-Sellers B. Bridging metamodels and ontologies in software engineering. Journal of Systems and Software. 2011 Feb 28; 84(2):301-13.
45. Rastgoo V, Hosseini MS, Kheirkhah E. Semantic Web-Based Software Engineering By Automated Requirements Ontology Generation In Soa. International Journal of Web and Semantic Technology. 2014 Apr 1; 5(2):1.
46. Zhu X, Jin Z. Ontology-based inconsistency management of software requirements specifications. In SOFSEM 2005: Theory and Practice of Computer Science. Springer Berlin Heidelberg; 2005 Jan 1. p. 340–9.
47. Zhu X, Jin Z. Inconsistency measurement of software requirements specifications: an ontology-based approach. Proceedings of 10th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS. IEEE; 2005 Jun 16. p. 402–10.
48. Siegemund K, Zhao Y, Pan JZ, Aßmann U. Measure software requirement specifications by ontology reasoning. In8th International Workshop on Semantic Web Enabled Software Engineering (SWESE'2012) 2012.
49. Athanasiadis IN, Villa F, Rizzoli AE. Enabling knowledge-based software engineering through semantic-object-relational mappings. Proceedings of the 3rd International Workshop on Semantic Web Enabled Software Engineering; 2007 Jun.
50. Bossche MV, Ross P, MacLarty I, Van Nuffelen B, Pelov N. Ontology driven software engineering for real life applications. Proceedings of 3rd International Workshop on Semantic Web Enabled Software Engineering; 2007 Jun.
51. Kaiya H, Saeki M. Ontology based requirements analysis: lightweight semantic processing approach. Fifth International Conference on Quality Software, (QSIC 2005). IEEE; 2005 Sep 19. p. 223–30).
52. Bosch J. Design and use of software architectures: adopting and evolving a product-line approach. Pearson Education; 2000.
53. Štuikys V, Damaševičius R. Measuring complexity of domain models represented by feature diagrams. Information Technology and Control. 2015 Apr 26; 38(3).
54. Wang HH, Li YF, Sun J, Zhang H, Pan J. Verifying feature models using OWL. Web Semantics: Science, Services and Agents on the World Wide Web. 2007 Jun 30; 5(2):117-29.
55. Zaid LA, Kleinermann F, De Troyer O. Applying semantic web technology to feature modeling. InProceedings of the 2009 ACM symposium on Applied Computing 2009 Mar 8 (pp. 1252-1256). ACM.
56. Murugesh S, Jaya A. Construction of Ontology for Software Requirements Elicitation. Indian Journal of Science and Technology. 2015 Nov 17;8(29).
57. Inostroza P, Astudillo H. Emergent architectural component characterization using semantic web technologies. In-Proc. Second Int'l Workshop Semantic Web Enabled Software Eng 2006 Nov.
58. Dinger U, Oberhauser R, Reichel C. SWS-ASE: Leveraging Web Service-based Software Engineering. InSoftware Engineering Advances, International Conference on 2006 Oct (pp. 26-26). IEEE.
59. Dietrich J, Elgar C. A formal description of design patterns using OWL. InSoftware Engineering Conference, 2005. Proceedings. 2005 Australian 2005 Mar 29 (pp. 243-250). IEEE.
60. 60. Hyland-Wood D, Carrington D, Kaplan S. Toward a software maintenance methodology using semantic web techniques. InSoftware Evolvability, 2006. SE'06. Second International IEEE Workshop on 2006 Sep 24 (pp. 23-30). IEEE.
61. Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Pearson Education; 1994 Oct 31.
62. Dietrich J, Elgar C. Towards a web of patterns. Web Semantics: Science, Services and Agents on the World Wide Web. 2007 Jun 30;5(2):108-16.
63. Henninger S, Ashokkumar P. An Ontology-Based Infrastructure for Usability Design Patterns. Proc. Semantic Web Enabled Software Engineering (SWESE), Galway, Ireland. 2005 Nov 6:41-55.
64. Henninger S, Ashokkumar P. An ontology-based metamodel for software patterns. 2006.
65. Gangemi A, Peroni S, Shotton D, Vitali F. A pattern-based ontology for describing publishing workflows. InProceedings of the 5th International Workshop on Ontology and Semantic Web Patterns, CEUR Workshop Proceedings 2014 (Vol. 1302).

66. Gangemi A, Presutti V. Ontology design patterns. InHandbook on ontologies 2009 Jan 1 (pp. 221-243). Springer Berlin Heidelberg.
67. Blomqvist E, Gangemi A, Presutti V. Experiments on pattern-based ontology design. InProceedings of the fifth international conference on Knowledge capture 2009 Sep 1 (pp. 41-48). ACM.
68. Grüninger M, Fox MS. The role of competency questions in enterprise engineering. InBenchmarking—Theory and Practice 1995 Jan 1 (pp. 22-31). Springer US.
69. Gangemi A, Pisanelli DM, Steve G. An ontological framework to represent norm dynamics. InProceedings of the 2001 Jurix Conference, Workshop on Legal Ontologies, University of Amsterdam 2001.
70. Mika P, Oberle D, Gangemi A, Sabou M. Foundations for service ontologies: aligning OWL-S to dolce. InProceedings of the 13th international conference on World Wide Web 2004 May 17 (pp. 563-572). ACM.
71. Gangemi A, Catenacci C, Battaglia M. Inflammation ontology design pattern: an exercise in building a core biomedical ontology with descriptions and situations. Studies in health technology and informatics. 2004:64-80.
72. Mohan K, Aramudhan M. Ontology based Access Control Model for Healthcare System in Cloud Computing. Indian Journal of Science and Technology. 2015 May 4;8(S9):218-22.
73. Gangemi A, Fisseha F, Keizer J, Lehmann J, Liang A, Pettman I, Sini M, Taconet M. A Core Ontology of Fishery and its Use in the FOS Project. InProceedings of the EKAW 2004 (Vol. 4).
74. Mortensen J, Horridge M, Musen MA, Noy NF. Modest Use of Ontology Design Patterns in a Repository of Biomedical Ontologies. InWOP 2012.
75. Kamthan P, Pai HI. An Experience in Ontological Representation of Web Application Patterns for the Semantic Web. InProc. First Int'l Workshop Semantic Web Enabled Software Eng 2005 Nov.
76. Ferreira DR, Alves S, Thom LH. Ontology-based discovery of workflow activity patterns. InBusiness Process Management Workshops 2012 Jan 1 (pp. 314-325). Springer Berlin Heidelberg.
77. Dublin Core Metadata Initiative. Dublin core metadata element set, version 1.1: Reference description. http://dublincore. org/documents/dces/. 2004.
78. IEEE Learning Technology Standards Committee. Draft standard for learning object metadata. Accessed July. 2002; 14:2002.
79. Kiefer C, Bernstein A, Tappolet J. Analyzing software with iSPARQL. InProc. rd International Workshop on Semantic Web Enabled So ware Engineering (SWESE')..(Cit. on p.) 2007 Jun.
80. Witte R, Zhang Y, Rilling J. Empowering software maintainers with semantic web technologies. InThe Semantic Web: Research and Applications 2007 Jan 1 (pp. 37-52). Springer Berlin Heidelberg.
81. Zhang Y, Witte R, Rilling J, Haarslev V. An ontology-based approach for traceability recovery. In3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), Genoa 2006 Oct 1 (pp. 36-43).
82. Decker B, Ras E, Rech J, Klein B, Hoecht C. Self-organized reuse of software engineering knowledge supported by semantic wikis. InProceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE) 2005 Nov 6.
83. Bhatia MPS, Kumar A, Beniwal R, editors. Ontology based framework for automatic software's documentation. Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on; 2015 March, 11-13.
84. de Graaf KA. Annotating software documentation in semantic wikis. InProceedings of the fourth workshop on Exploiting semantic annotations in information retrieval 2011 Oct 28 (pp. 5-6). ACM.
85. Tang A, Liang P, Van Vliet H. Software architecture documentation: The road ahead. In2011 Ninth Working IEEE/IFIP Conference on Software Architecture 2011 Jun 20 (pp. 252-255). IEEE.
86. De Graaf KA, Liang P, Tang A, Van Hage WR, Van Vliet H. An exploratory study on ontology engineering for software architecture documentation. Computers in Industry. 2014 Sep 30;65(7):1053-64.
87. Vigneshwari S, Aramudhan M. Social Information Retrieval Based on Semantic Annotation and Hashing upon the Multiple Ontologies. Indian Journal of Science and Technology. 2015 Jan 1;8(2):103-7.
88. Tang A, Babar MA, Gorton I, Han J. A survey of architecture design rationale. Journal of systems and software. 2006 Dec 31;79(12):1792-804.
89. Kruchten P, Capilla R, Dueas JC. The decision view's role in software architecture practice. Software, IEEE. 2009 Mar;26(2):36-42.
90. López C, Codocedo V, Astudillo H, Cysneiros LM. Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. Science of Computer Programming. 2012 Jan 1;77(1):66-80.
91. Al Balushi TH, Sampaio PR, Dabhi D, Loucopoulos P. Performing Requirements Elicitation Activities Supported by Quality Ontologies. InSEKE 2006 Jul (pp. 343-348).
92. Zhu H, Huo Q. Developing a software testing ontology in UML for a software growth environment of web-based applications. Software Evolution with UML and. 2005; 1060:263-95.
93. Nasser VH, Du W, MacIsaac D. An Ontology-based Software Test Generation Framework. InSEKE 2010 (pp. 192-197).
94. Paydar S, Kahani M. Ontology-based web application testing. InNovel Algorithms and Techniques in Telecommunications and Networking 2010 Jan 1 (pp. 23-27). Springer Netherlands.
95. Li H, Chen F, Yang H, Guo H, Chu WC, Yang Y. An ontol-

ogy-based approach for GUI testing. InComputer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International 2009 Jul 20 (Vol. 1, pp. 632-633). IEEE.
96. Li X, Zhang W. Ontology-based testing platform for reusing. InInternet Computing for Science and Engineering (ICICSE), 2012 Sixth International Conference on 2012 Apr 21 (pp. 86-89). IEEE.
97. Dalal S, Kumar S, Baliyan N. An Ontology-Based Approach for Test Case Reuse. InIntelligent Computing, Communication and Devices 2015 Jan 1 (pp. 361-366). Springer India.
98. Iliev M, Karasneh B, Chaudron MR, Essenius E. Automated prediction of defect severity based on codifying design knowledge using ontologies. InRealizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 First International Workshop on 2012 Jun 5 (pp. 7-11). IEEE.
99. April A, Desharnais JM, Dumke RR. A Formalism of Ontology to Support a Software Maintenance Knowledge-based System. InSEKE 2006 (pp. 331-336).
100. Rilling J, Zhang Y, Meng WJ, Witte R, Haarslev V, Charland P. A unified ontology-based process model for software maintenance and comprehension. InModels in Software Engineering. Springer Berlin Heidelberg; 2007 Jan 1. p. 56-65.
101. Khamparia A. Performance analysis of SPARQL and DL-QUERY on electromyography ontology. Indian Journal of Science and Technology. 2015 Aug 13; 8(17).
102. Slimani T. Ontology Development: A Comparing Study on Tools, Languages and Formalisms. Indian Journal of Science and Technology. 2015 Sep 15;8(1).
103. Bhatia MPS, Beniwal R, Kumar A. An ontology based framework for automatic detection and updation of requirement specifications. International Conference on Contemporary Computing and Informatics (IC3I). IEEE; 2014 Nov 27. p. 238–42.
104. Bouchiha D, Malki M, Benslimane SM. Ontology based web application reverse-engineering approach. INFOCOMP journal of Computer Science. 2007 Mar 1; 6(1):37-46.
105. Mohamed Benslimane S, Malki M, Bouchiha D. Maintaining web application: an ontology-based reverse engineering approach. International Journal of Web Information Systems. 2009 Nov 20; 5(4):495-517.
106. Di Lucca GA, Di Penta M, Antoniol G, Casazza G. An approach for reverse engineering of web-based applications. Proceedings of Eighth Working Conference on Reverse Engineering. IEEE; 2001. p. 231–40.
107. Antoniol G, Di Penta M, Zazzara M. Understanding web applications through dynamic analysis. Proceedings. 12th IEEE International Workshop on Program Comprehension. IEEE; 2004 Jun 24. p. 120–9.