ISSN 2007-9737

Information Retrieval from Software Bug Ontology Exploiting Formal Concept Analysis

Shubhra Goyal Jindal, Arvinder Kaur

University School of Information and Communication Technology,
Guru Gobind Singh Indraprastha University,
India

{shubhra.phd, arvinder} @ipu.ac.in

Abstract. Knowledge extraction and structuring is attaining importance in real world applications such as e-commerce, decision support, problem solving and semantic web. Extraction of knowledge from collection of text documents is based upon identification of semantic content. Ontology plays an important role in accessing and structuring information. Developing an ontology are at the core of new strategies which requires accurate domain knowledge. Identification of structural and logical concepts is a time-consuming process. This work presents an ontology-based retrieval approach, that visualizes and structure the data of software bug reports domain. It exploits formal concept analysis (FCA) to elicit conceptualizations from bug reports datasets and a hierarchical taxonomy is generated of extracted knowledge. A lattice diagram of concepts and relationships is constructed from concept-relationship matrix created by FCA. Ontology is constructed on fluent editor tool and knowledge is extracted with the help of small queries executed on a reasoner window. The proposed approach is evaluated on 21 bug reports of apache projects of jira repository. It can be concluded that information can be retrieved easily from ontology as compared to manual extraction of data.

Keywords. Knowledge extraction, formal concept analysis, ontology, software bug reports, concept-lattice.

1 Introduction

In recent years, semantic web relies on ontologies as a means for data sharing and communication. Semantic web was introduced by Gruber [1], which converts the unstructured data into structured data that is easily processed by machines without human intervention. It enables the users to search questions, retrieval of information and knowledge extraction with minimum effort.

Conceptualizations of underlying knowledge and shared understanding of domains is provided by ontologies. Ontology is considered as a tool for modelling an abstract view of contextual semantic analysis of documents.

Ontology is defined as formal representation of knowledge within a domain by a set of concepts and relationship between these concepts [2]. Ontology is a feasible solution for accessing data, modelling complex domain and interoperability through standard languages Ontology Web Language (OWL) and Resource Description Framework (RDF).

Domain knowledge can be restored in ontology [3], but identification of concepts and their relationship is still an obstacle in many domains [4, 5]. Ontology is an instrument for knowledge structuring and yielding controlled vocabulary for content classification is an information domain [6].

In software engineering, software bug reports are the most valuable aspect in software development and maintenance process.

They depict numerous information of software bug in the form of metadata (containing BugId, project name, component name, release version, one-line description and others), long description and comments made by various contributors made to resolve a bug.

Several researchers have worked in literature on unstructured data of bug reports in various fields such as bug triaging [7, 8], severity prediction of bug reports [9–12], defect prediction [13], classification of bug reports [14, 15] and many others. Among various applications of bug reports, complete semantic knowledge about bug has not been addressed so far.

For example, if we want to retrieve all the contributors of a bug Id# Hdfs-7707.It should fetch all the contributors who were involved in the resolution process of #Hdfs-7707 as compared to retrieving only the name of the developer to whom it is assigned. The semantic confides to formal ontology's that structures the data extensively for machine understanding. To extract knowledge from a large corpus of bug reports, bug ontology is needed. Although developing an ontology from scrape is a labor intensive, time-consuming tasks and requires precise and detailed understanding of any domain.

Few researchers had worked on ontology of software bugs, defects [16–18]. As per the literature, none of the research work focused on extracting knowledge using ontology automatically form the content analysis of bug reports. The proposed approach exploits Formal Concept analysis (FCA) as a mathematical model to construct an ontology. This work extracts important and beneficial information from the metadata, long description and comments of software bug reports of Apache projects of jira repository. Text mining is performed on unstructured data and features are extracted. Unigrams, bi-grams and trigrams are extracted as features. Based on analysis, tri-grams were discarded and it does not significantly contribute in information extraction.

Concepts and attributes are identified based on unigrams and bi-grams and the relationship between them are mapped into concept lattice structure using Formal Concept Analysis (FCA). FCA is used for information and knowledge representation and analysis of data [19]. It visualizes all concepts and their relationships in tabular form resulting in a concept lattice.

For ontology construction, to link and connect concepts, relationships such as 'is-a', 'has-a' and user defined relationships such as 'contributors-in', 'is-not', 'has-critical' etc. is considered, An ontology of 21 bug reports of apache projects namely: Hadoop-HDFS, Hadoop-Common, Hive, Hbase and Groovy. Based on ontology, the knowledge is extracted using several small queries written in common natural language (CNL) using fluent editor tool. Fluent editor tool is used for developing and reasoning ontology. It is used in this work as compared to other tools such as Protégé as it used controlled natural language (cnl) which is easy to understand by any user [20].

Ontology is published on web using ontology web language(owl). Thus, domain ontology of software bug reports is valuable for developers and researchers as it contains the knowledge of all bug reports, which can be extracted using ontology supporting languages.

The rest of the paper is organized as follows: section 2 reviews the previous work done in the area of ontology and ontology of software bug reports. The research methodology is explained in section 3 followed by Implementation and results in section 4.

2 Related Work

This section reviews the work done in construction of ontology based on learning from text and in the field of software engineering and software bug reports. It also includes the work of authors such as Philipp Cimiano, Alexander Maedche and Stefan Staab.

2.1 Ontology from Texts and Software Engineering

Ontology is a crucial part while developing semantic applications. Several researchers have worked in the area of semantic web and ontology construction. V. Uren et al. [21] presents a survey of general-purpose tools that are used for annotation of semantic web, they have further examined these tools based on manual annotation, automatic annotation, integrated annotation or on-demand annotation. Manual annotation tools are compared based on different requirements such as standard formats, user-centered design, ontology support, document formats, document evolution and annotation storage.

The survey indicates that WickOffice and ActiveDoc are two examples of integrated authoring environment tools but still are limited to their degree of automation and variety of covered documents. There is a lot of scope of improvement in other existing automation system to provide full automatic annotation on wide range of documents and platforms.

G. Stumme et al. proposed a novel method for merging ontologies with bottom up approaches named as FCA-merge.

The approach consists of three steps: extraction of instances and computing two formal contexts namely K_1 and K_2; application of FCA-merge algorithm to derive common context and generate a concept lattice; based on concept lattice, fine merged ontology is generated. The approach is evaluated on tourism domain [22]. A Hotho et. al. employed a simple, core ontology to generate disparate representations of a particular document set which are a result of multiple clustering algorithms like k-means.

Then based on the results if different clustering algorithm and actual concepts, user can conclude the results. The author had proposed a new approach namely. COSA (Concept Selection and Aggregation) which has been evaluated on customer database of telecommunication domain [23]. The work done on ontology creation in the field of software engineering is focused here. E. Blomqvist [24] proposed a pattern-based ontology construction, ontocase based on case-based reasoning. An automatic approach for pattern matching and selection is proposed influenced by ontology ranking to bridge the gap between patterns and specific features extracted from texts. The approach is evaluated and the results signify that proposed approach performs significantly better on small and abstract patterns. P.E. Khoury et al. proposed an ontology-based approach to identify various security patterns needed by software developers.

The description of security properties are used for ontology [25]. An ontological mapping is proposed to map requirements from one side to other side of contexts such as threat models, security bags and security errors. In another research work, software product line engineering (SPLE) paradigm is considered which is based on reusing artefacts and knowledge from similar software products. I.R. Berger et al. proposed an automated extraction method, CoreReq.

The main task is to generate core and reusable requirements from existing product requirements that can be used by distinct members of software product line to generate requirements of a new product.

The approach is based on ontological framework with two dimensions: elements and product.

CoreReq analyzes the product requirements through natural language processing and compares those using semantic measures such as latent semantic analysis and ontological variability analysis.

Based on similarity measures, similar requirements are categorized according to the dimensions of the framework. The approach is evaluated on four software products namely: Hotel, Library, Car rental and Second hand book shop as all deal with check-in check-out operation. The core requirements are gathered and were reused to generate the requirement of fifth product i.e medical equipment rental system[26]. Apart from these several artefacts of software, work has also been done on ontology of software bug reports as discussed in section 2.2.

2.2 Ontology on Software Bug Reports

In software engineering, every reference model of each organization employs its own vocabulary to explain failures, errors and defects that occur. This leads to deficiency in understanding these concepts of different software during product interoperability and other tasks. To reduce this problem of common conceptualization, domain ontology of software defects, error and failures (OSDEF) is proposed by B. Duarte et al. The ontology is developed using Systematic Approach for Building Ontologies (SABiO) and grounded on Unified Foundation Ontology (UFO).

Main features of OSDEF are: providing conceptual analysis of nature of distinct anomalies such that notions like failure, fault, defect and error refers to different types of phenomena. It is used for developing issue trackers and configuration management tools. It establishes a common vocabulary for better communication among software engineers and stakeholders. Thus, OSDEF is domain ontology to provide a common conceptual structure to different types of anomalies and vocabulary for better communication in software engineering [18].

In [16], authors have extracted numerous bug reports from several open source bug tracking

systems such as Bugzilla, Trac, Mantis and Debian using APIs and Buglook (web crawler).

To store such large amount of data, a unified bug data model is built that captured the most important aspects of version tracking system.

To perform semantic search of bug reports on unified bug dataset, two methods are used: MVR (Multi-vector representation) and RDF (Resource Description Framework).

The MVR method was executed on semi-structured data (full text and metadata) of bug reports to search similar bugs with salient features. While RDF method used bug correlation, symptom-based classification and package dependency. The results proved that semi-structured search outperforms other consolidation of search methods. It also proved that there exists a smaller number of similar bugs in bug tracking system.

In their consecutive study [17], authors proposed to find similar bug reports using semantic bug search system on Peer to peer network. A unified bug schema is created, which stores several types of bug reports from various bug tracking systems. It also has several properties such as dependencies, packages, symptoms, categories based on concepts that help in bug classification and relationships for semantic search. Further, Gnutella P2P protocol is used that uses super peers for query routing and processing, to improve the performance of the system. The proposed system is evaluated on EMANICSLAB on three issues: feasibility, scalability, and efficiency.

P. Schueger et al. used semantic web technology to compute the quality of bug reports and in turn to enrich existing software engineering ontology. In this work, existing quality attributes are refined, and new quality attributes are proposed to intensify the quality of bug reports. New quality attributes such as certainty, focus, reproducibility, and observability are identified from keywords and key expressions embedded in description of bugs. The proposed technique is evaluated on dataset of AgroUML and quality of bug reports was used to extract knowledge from software engineering ontology [27]. Evoont is a collection of software ontology, bug and version ontologies which is used for software analysis, design and bug tracking purpose [39].

Two ontology such as Baetle (Bug and enhancement tracking language) [40] and Helios [41] are under development towards a unified ontology of software bugs.

In literature, an ontology is constructed to perform semantic search to find similar bugs in bug tracking systems or quality attributes are defined to enrich software engineering ontology, but none of the work focused on ontology construction that extract knowledge from metadata, long description and comments of bug reports.

To help software developers to extract knowledge of semantically related bugs based on several attributes, ontology is created. As most of the knowledge is enclosed in corpus of unstructured bug reports, ontology assures efficiency, accuracy and effectiveness in information retrieval process. The approach is implemented on bug reports of apache projects of jira repository. Text mining is performed on long description and comments and features (unigrams and bi-grams) are extracted using term frequency-inverse document frequency (tf-idf).

Concepts and several attributes such as type of error, major contributors, severity, and resolution are identified using extracted features. Relationship between concepts and attributes are mapped to form concept-lattice matrix using Formal concept analysis. Based on identified relationships, ontology is constructed and knowledge is extracted using queries executed on a reasoner tool. The created ontology is also published on web using ontology web language. Thus, use of ontologies in the domain of bug reports is indispensable.

3 Research Methodology

This section explains the various concepts used in this research work in section 3.1. The process flow diagram of the proposed work is illustrated and explained in section 3.2.

3.1 Preliminary Concepts

Ontology construction and visualization require preliminary activities, which forms the part of execution process of the system. It includes text

pre-processing, formal concept analysis and fluent editor tool.

3.1.1 Text Pre-Processing

Text pre-processing includes standard pre-processing steps i.e. tokenization of text data, removal of punctuations and numbers, stop word removal and stemming. Tokenization is segmentation of text into substantial elements such as words, symbols, phrases called as tokens. Tokenization is followed by removal of punctuations and numbers. In next step, stop words are removed.

Stop words are common words such as *the, and, this, these* etc., which do no convey any significant information. Stemming is performed at the last, which reduces the words to their root forms. For example, *likes, liked, likely* are reduced to word '*like*' as their base form. After pre-processing the textual data, document term matrix (DTM) is created [12].

3.1.2 Formal Concept Analysis

Formal concept analysis (FCA) is a conceptual clustering technique with mathematical foundation, which formalizes concepts as basic unit and analyses data in object-attribute form. It is used as a tool for knowledge representation, information retrieval and analysis of data. It is a contrasting approach as compared to traditional and statistical means of knowledge retrieval and data analysis techniques as it focusses on human centered approaches [28]. It has been used in several applications such as psychology, medicine, ecology and software engineering [29]. FCA is used over other methods like pattern-based approach and frame-based system due to several issues.

These issues are described as: objects that are used to create hierarchies are not clearly distinct in terms of attributes, which raises a concern in knowledge sharing; change of classes and its attributes is not easy once they are defined. To overcome these difficulties of several existing methods FCA is used as: it identifies concepts that are distinctly described by properties; hierarchy is built based on the identified concepts not by explicit designers [28]. Other main advantages of exploiting FCA is that i) it is suitable for collaborative environment with distinct designers working on a single ontology. ii) sets of formal concepts and Galois connections can be graphically visualized. FCA identifies conceptual structures, which are graphically represented as concept lattice. Concept lattice is represented by a binary matrix called as formal context.

Table 1. formal concept matrix

Concepts/attributes	Attribute 1	Attribute 2
Concept 1	×	
Concept 2	×	×
Concept 3		×

Formal context is a triple of O, A, I where O is a set of objects, A is a set of attributes and I is the relation between O and A such that $o, a \in I$ means that object o has attribute a. The context is represented as a cross table where rows represents the formal objects or concepts and column represents formal attributes and crosses represents the relation between them as depicted in table 1 [6-19].

Formal concept is based on two properties: extensions and Intentions. Formal concept is a pair of (E, I) where, E is a set of objects called the extent of concept and I is set of attributes called as intent of concept, such that $E' = I \ \& \ I' = A$

Formal concept analysis is used in numerous applications in various fields of software engineering [30, 31], data mining [19] and development of ontology [32, 33].

3.1.3 Fluent Editor Tool

Fluent editor tool is used for construction, editing, manipulating and reasoning complex ontology. It is developed by cognitum and uses controlled natural language (cnl) for developing and reasoning ontologies as compared to other ontology tools such as Protégé [20]. It has several tools that are used to manage complex ontologies such as: a reasoned window, SPARQL window, xml preview window and taxonomy. Through a reasoned window, knowledge can be extracted by asking queries, SPARQL window executes SPARQL queries and xml wile is generated using xml preview window, similar to ontology web language (owl) for any domain. Fluent editor tool is used in several domains for ontology construction.

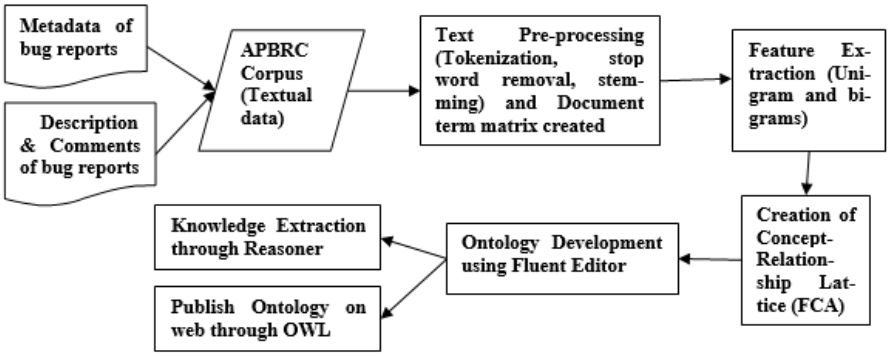

Fig. 1. Process flow diagram of proposed approach

3.2 Process Flow Diagram

Figure 1 depicts the general idea of process flow of the proposed approach through main phases.

3.2.1 Corpus Creation and Text Pre- Processing

A corpus of Apache projects Bug Report Corpus (APBRC) is established, which contains the metadata (BugId, Priority, Severity, Project Name, Component, Name, Release Version, One-line Description etc.), long description and comments made by various contributors.

The textual data is then processed using standard pre-processing steps namely, Tokenization, removal of numbers and punctuations, removal of stop words and stemming. After text pre-processing, document term matrix is created. Sparsity is removed to create a dense vector. Text pre-processing is performed using R language including various packages such as 'tm' and 'NLP'.

3.2.2 Feature Extraction

Humans can understand the linguistic structures and its meaning from texts. But to train machines on linguistic structure and their meaning, there is a need to extract features from the text data. Text consists of sentences and sentence consists of words. In context of text-corpus, sequence of words is called n-grams.

N-grams are set of co-occurring words within a given window (N). If N=1, it is called as unigram, if N-2, it is known as bi-grams, N=3 is called as tri-grams and so on.

To construct concept-relationship matrix, unigrams and bi-grams are extracted from document term matrix(dtm) using NGramTokenizer() and BiGramTokenizer() function.

Tri-grams were also extracted, but it made no significant contribution in identification of concepts and relationships, so was discarded.

After extracting unigrams and bi-grams for each bug report, the relationship among concepts and attributes are identified.

3.2.3 Creation of Concept-Relationship Lattice

In this step, bugId's are considered as concepts and several attributes are identifies based on metadata of bug reports and extracted features (unigram and bi-grams). After the identification of concepts and attributes, relationship between them is identified. In this work, 21 concepts are identified as BugId and several attributes such as Resolved, Severe, Non-severe, Major contributors, network error, Edit log error, Namenode Error, Authentication error, File Permission error, Input/Output error, Application-consistency error, Bytebuffer error, configuration error and Java error were identified and relationship between each concept and attribute is found using FindAssoc() function and are mapped.

Formal concept analysis is used to represent binary relationship among two concepts. For illustration, to extract which bug has severity as severe and edit log error from concept-relationship matrix, it is easily possible to extract data as bugId HDFS-7707 and HDFS-13112 as depicted in table 2. After mapping the concept and relationships, a lattice diagram is constructed to depict the extent and intent among different concepts.

3.2.4 Ontology Development and Evaluation

After construction of concept-relationship matrix, the concept –relationship data is fed in the form of Controlled Natural Language (CNL) in Fluent editor tool. The ontology of software bug reports is developed to discover different aspects of bug reports. The knowledge is extracted with the help of queries through reasoned. To extract knowledge from bug report ontology, several questions are formed such as:

- What is a bug?
- What are different bugs?
- What is different bug-id numbers?
- Who contributes in which type of bug?
- What are different types of bugs?
- Who contributes-in Hdfs-7707?
- What is blocker-severity bugs?
- What are critical severity bugs?
- Who has-major-severity?
- Who has a network-error?
- Who has authentication error?

Similarly, more knowledge can be extracted through several more queries illustrated in section 4 on a reasoner. The ontology can also be published on web through Ontology Web Language (OWL). The ontology is evaluated on the basis of two perspective: - Ontology Quality and Ontology Correctness [34]. In order to evaluate ontology, ontology is classified into three categories. First category is based on the definition of ontology evaluation , second is based on layered approach to ontology evaluation and last is based on comparison against a gold standard/ application or task-based/user-based or data driven evaluation [35]. In our work, the proposed approach is an example of application-based ontology evaluation. In this type of evaluation, ontology is just plugged in into an application and results are evaluated which is relatively straightforward and unambiguous. It also describes with relations (is-a, has-a and others) are used to induce the clones among two concepts in terms of their meaning.

4 Implementation and Results

In this section, the process of data collection and several steps such as feature extraction, generation of concept-lattice and ontology development is illustrated.

4.1 Data Collection

To train the model and build ontology, a corpus of bug reports is required. Bug reports of five projects of Apache software foundations are extracted in time duration of 2015-2018. The bug reports are extracted using the tool named Bug Report Collection System (BRCS) [36] for five projects namely, Hadoop-common, Hadoop-hdfs, Hive, Hbase and Groovy. Long description, comments made by several contributors and metadata information (bugId, Resolution, Priority, status, one-line description, assigned to, component, date, version etc.) of about twenty thousand bug reports are extracted. From this large pool of data, 21 bug reports are selected to construct ontology of bugs' domain and knowledge extraction.

4.2 Process Illustration

Step 1: Feature Extraction

To train machines on linguistic structures and predict natural language accurately, features are extracted. Bug reports are pre-processed using text mining techniques and document term matrix(dtm) is created. Unigrams are extracted using findfrequentterm() function. NGramTokenizer function is applied on dtm and bigrams are extracted. Unigrams and bi-grams are extracted as features. Significant unigrams and bi-grams with frequency above than 10 for each bug report are selected and depicted in table 2.

Step 2: Creation of Concept-Lattice

A conceptual framework is created using formal concept analysis to structure, analyze and visualize data to make it more understandable.

Table 2. Feature set of bugs

Bug Id#	Unigrams	Bi-grams
Hdfs-7707	Block, check, delayed, edit, log, hdfs, new, removal, file, kihwal, deleted, test	Edit log, block removal, delayed block, applied patch, due delayed, log corruption, total number, corruption due, creation time
Hdfs-8312	Branch, check, delete, permission, files, fix, trash, rename	Ecn flags, permission error, separate ecn, status datatransferpipelineack, datatransferpipelineack contributed
Hdfs-13112	Edit, edits, expiration, expired, lock, secret, threads, token, tokens	Expired tokens, edit logging, secret keys, token expiration, write lock, edit log, old tokens, read lock, secondary namenode, namenode may, retain node, secret manager
Hadoop-13155	Added, config, delegation, hadoop, token, new, renewer	Delegation token, config key, request header, accessing required, create new, delegation tokens, dt request, hdfs encryption, looks like
Hadoop-13890	Can, code, failed, fix, kerberosname, server, spn, spnego, unit, like, principal, realm	Kerberosname parsing, like httphost, spnego spn, unit tests, local realm
Hadoop-14146	branch, hadoop, http, keytab, realm, principal, spn, spnego, server, support, work	Internal spnego, support multiple, local realm, host header, service principal, conf key, hadoop spnego, principals specified
Hadoop-11802	Add, applied, block, datanode, deadlock, domainsocketwatcher, exception, failure, fail, fix, issue, log, logging, slot, socket	Domainsocketwatcher thread, deadlock domainsocketwatcher, fix deadlock, domainsocketwatcher notification, pipe full
Hadoop-15273	Checksum, blocksizes, datacorruption, remote	Different checksum, stores different, blocksizes copy, checksum mismatch, checksumchecks altogether, differ blocksize, handle remote, hdfs webhdfs, masking datacorruption, preserve blocksizes, remote stores, risk masking
Hbase-13229	Cell, column, grant, groups, hbase, level, specific	Work groups, grant specific, specific column, column level, hbase grant, level work, cell level, grant cell
Hbase-13732	Does, fails, intermittently, number, tests, testhbasefsckparallelwithretrieshbck,	Does not, fails intermittently, testhbasefsckparallelwithretrieshbck fails, hbase13732 testhbasefsckparallelwithretrieshbck
Hbase-15638	Can, classes, hbase, hbaseprotocol, module, protobuf, references, shaded, version	Hbaseprotocol module, pb references, shade plugin, shaded protobuf, module relocated, coprocessor endpoints, use protobuf, protobuf version, final artefact
Hbase-19496	Bbpool, bytebuffer, branch, code, data, clone, layer, methods, rpc, server, serverload, regionload, pool	Pb object, rpc layer, min size, cloneing request, data rpc, call done, build hbasetrunkmatrix
Hbase-20004	Api, browser, client, rest, security, server, release	Rest server, rest api, rest queries, allow options, based query, browser based, clients expect, config knob, disallow default, firefox browser
Hbase-20201	Can, commonscli, hadoop, hbasethirdparty, roll	Hadoop jar, new hbasethirdparty, looks like, back commonscli, broke hadoop, commonscli dependencies, due jobs, depends commonscli
Hive-12538	Conf, hive, issparkconfigupdated, multiple, operation, queries, operation, session, spark	Session level, level conf, spark session, operation level, issparkconfigupdated false, concurrent queries, conf object, hs session, multiple concurrent, one session
Hive-10428	Executed, failed, least,noformat	Due failederrored, failed tests
Hive-13369	Automatically, failed, executed, noformat, open, modified	Precommithivemasterbuild, open txns, failed tests
Hive-11112	Case, fix, hive, method, noformat, test, texts	Test case, hive duplicate
Hive-15827	Exception, instances, hive, llap, mode, registry, root, running, service, watch	Root path, watch mode, live instances, registry service, llap registry, connects llap, id precommithivebuild
Groovy-7535	Atomicinteger, categoryinuse, fix, issue, category, thread	Race condition, another thread, github user, int counter, read hascategoryincurrentthread, using atomicinteger, fgit closed, closed pull, fix issue, int field
Groovy-8483	Gradle, groovy, issue, jar, missing, pom, time	Artefact now, fat jar, packaging looks, release process, also synced

Table 3 Concept-relationship matrix

Attributes / Concepts	Res	S	NS	MC	NNE	ELE	NE	AE	FPE	I/O E	ACE	BBE	CE	JE
Hdfs-7707	×	×		×		×	×		×					
Hdfs-8312	×	×		×				×	×					
Hdfs-13112	×	×		×	×	×		×						
Hadoop-13155	×	×		×	×			×						
Hadoop-13890	×	×		×	×			×						
Hadoop-14146	×	×		×	×			×						
Hadoop-11802	×	×		×						×				
Hadoop-15273	×	×		×	×									
Hbase-13229	×		×	×				×	×					
Hbase-13732	×		×	×					×		×			
Hbase-15638	×	×		×				×				×		
Hbase-19496	×	×		×							×	×		
Hbase-20004	×		×	×	×			×						
Hbase-20201	×	×		×	×				×					
Hive-12538	×	×		×							×		×	
Hive-10428	×	×		×							×			
Hive-13369	×	×		×					×		×			×
Hive-11112	×	×		×	×					×				×
Hive-15827	×	×		×	×			×			×			×
Groovy-7535	×	×		×					×		×			×
Groovy-8483	×	×		×						×	×			×

Concepts and attributes are identified and the relationship among them are mapped in the cross-table form depicted in table 3. Figure 2 depicts the snapshot of concept-relationship matrix created through concept explorer tool. The extensions and intentions of concepts are depicted in the lattice diagram in figure 3. Here, only subconcepts is depicted and to extract more relationship between concepts and attributes, an ontology is constructed using Fluent Editor tool as explained in step 3.

Step 3: Development of Ontology

After the construction of lattice diagram, the concepts and its relationship are fed as an input to fluent editor tool in controlled natural language.

To build an ontology many relationships are considered such as ‘is-a’, ‘has-a’, ‘is-not a’ relationship.

Along with these predefined relations, several user-defined relations are also considered such as ‘contributors-in’, ‘has-major’, ‘has-critical’, ‘has-blocker’ are used.

As per the literature, the proposed approach and constructed ontology is application-based ontology [34, 35] That is, it is constructed form the datasets of a particular domain. It is not compared with any gold standard ontology. thus, it is not possible to evaluate the ontology through metrics such as precision, recall or f-measure [37].

Based on further research, the approach can be evaluated using data quality dimensions. Data

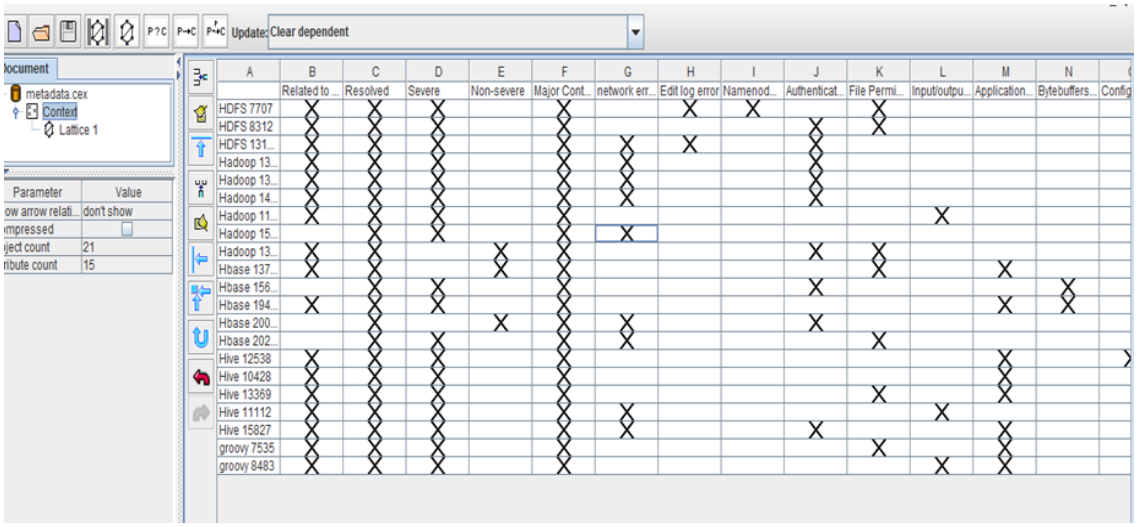

Fig. 2. Output of concept-relationship matrix

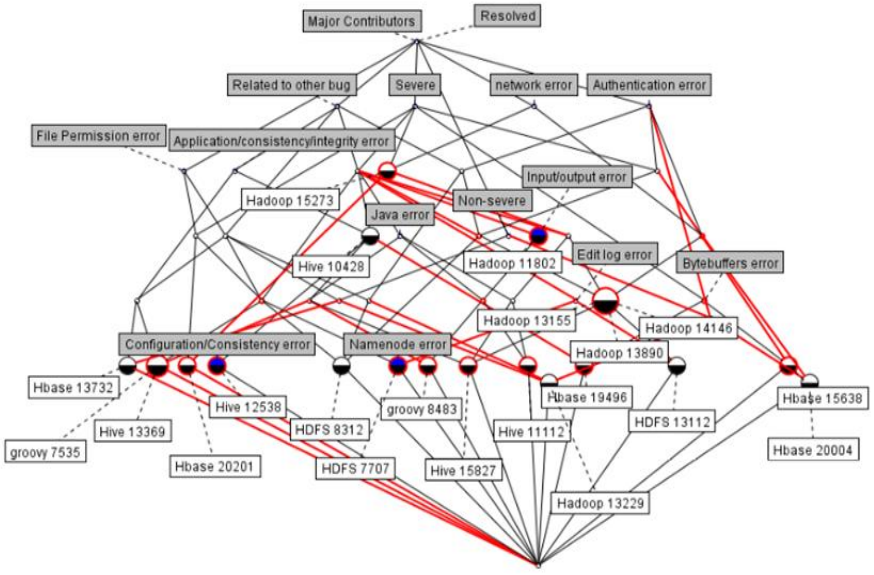

Fig. 3. Concept-relationship lattice

quality dimensions are accepted universally and are used to measure the quality of data [38]. The ontology of bug reports is evaluated using several dimensions such as completeness, consistency, uniqueness, validity and accuracy. Various metrics are defined in table 4. Thus, use of ontologies in the domain of bug reports is indispensable.

Res-Resolution, S-severe, NS-Non-severe, MC-Major contributors, NNE- NameNode error, ELE-Edit log error, NE- Network error, AE-Authentication error, FPE-File Permission error, I/OE-Input/Output Error, ACE-Application-consistency error, BBE- Bytebuffer error, CE-Configuration error, JE-Java error.

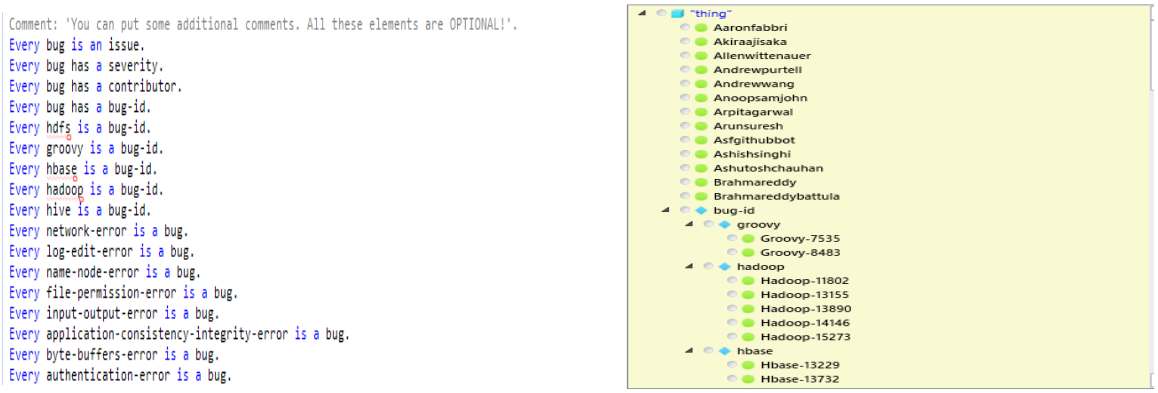

Fig.4. Ontology creation and taxonomy tree

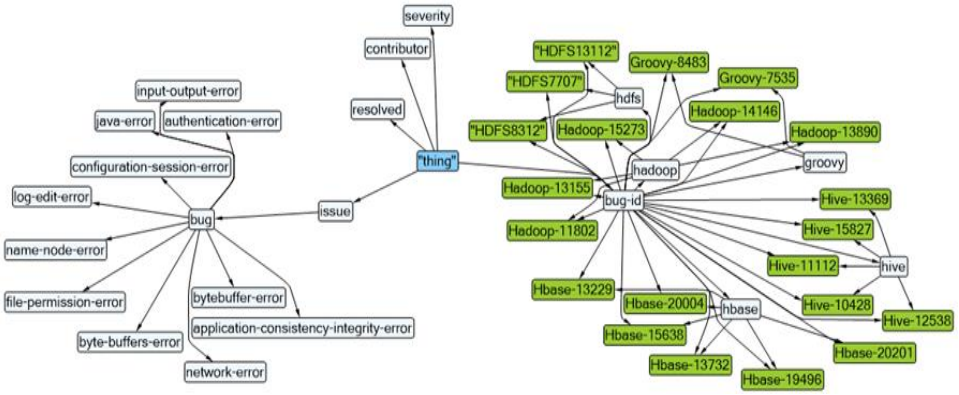

Fig. 5. Ontology on software bug reports

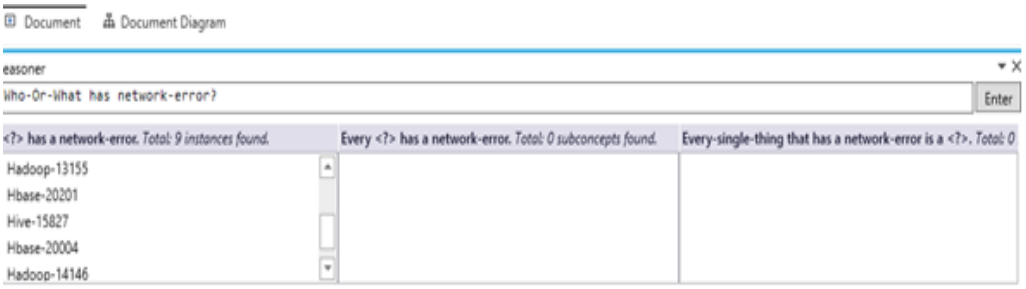

Fig. 6. Knowledge extraction using Reasoner

To evaluate our constructed ontology, the metrics defined in table are evaluated using 3 annotators working in the same domain of software bug reports. The results of three annotators are then averaged to achieve final results in table 5.

A taxonomy tree and ontology are developed. To extract knowledge from an ontology, several queries are formed which are executed on a reasoner. The queries generated and knowledge extracted is tabulated in table 6.

The output of fluent editor tool and taxonomy tree is depicted in figure 4. The complete generated ontology has been illustrated in figure 5.

The knowledge extracted through a reasoner is depicted in figure 6.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Ontology ontologyIRI="http://ontorion.com/namespace#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xml:base="http://ontorion.com/namespace#" xmlns="http://www.w3.org/2002/07/owl#">
<!--Generated by Cognitum FluentEditor2015, a part of Ontorion(TM) Knowledge Management Framework, (with support of OwlApi)-->
    <Prefix IRI="http://ontorion.com/namespace#" name=""/>
    <Prefix IRI="http://www.w3.org/2002/07/owl#" name="owl"/>
    <Prefix IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" name="rdf"/>
    <Prefix IRI="http://www.w3.org/2001/XMLSchema#" name="xsd"/>
    <Prefix IRI="http://www.w3.org/2000/01/rdf-schema#" name="rdfs"/>
  - <SubClassOf>
        <Class IRI="Bug"/>
        <Class IRI="Issue"/>
    </SubClassOf>
  - <SubClassOf>
        <Class IRI="Bug"/>
      - <ObjectSomeValuesFrom>
            <ObjectProperty IRI="has"/>
            <Class IRI="Severity"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
  - <SubClassOf>
        <Class IRI="Bug"/>
      - <ObjectSomeValuesFrom>
            <ObjectProperty IRI="has"/>
            <Class IRI="Contributor"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
  - <SubClassOf>
        <Class IRI="Bug"/>
      - <ObjectSomeValuesFrom>
            <ObjectProperty IRI="has"/>
            <Class IRI="BugId"/>
        </ObjectSomeValuesFrom>
```

Fig. 7. Published ontology on web using OWL

Table 4. Definition of metrics

Metrics	Definition
Completeness	It is defined as the proportion of data stored in contrast to “100% complete” and focusses on measuring critical data.
Uniqueness	It is defined as everything that is identified is recorded only once.
Validity	It is defined as data is valid if it conforms to the syntax.
Accuracy	It is defined as degree to which data describes the “real world” object correctly.
Consistency	It is defined as how consistent the data is when compared with two or more representations.

Table 5. Evaluation of ontology by annotators

Metrics	Annotator 1	Annotator 2	Annotator 3	Final Results
Completeness	90%	85%	95%	90%
Uniqueness	95%	80%	90%	88.3%
Validity	90%	85%	80%	85%
Accuracy	85%	90%	85%	86.6%
Consistency	90%	90%	90%	90%

Table 6. Knowledge extraction through query retrieval

Queries executed	Information Extracted	Additional results
What is a bug-id?	Hdfs-7707, Hdfs-8312, Hdfs-13112, hadoop-13155, hadoop-13890, hadoop-14146, hadoop-11802, hadoop-15273, hbase-13229, hbase-13732, hbase-15638, hbase-19496, hbase-20004, hbase-20201, hive-12538, hive-10428, hive-13369, hive-11112, hive-15827, groovy-7535, groovy-8483	21 instances, 5 subconcepts
What is a bug?	Byte-buffer-error, name-node-error, java-error, log-edit-error, file-permission-error, application-consistency-error, configuration-error, input-output-error, network-error, authentication-error	11 subconcepts, 1 superconcept
Who contributors –in?	Yongzhichen, andrewpurtell, stack, ericyang, Hudson,khilwallee, seanbusbey, henrik, siddharthseth, many others	78 instances
Who contributors-in Hdfs-7707?	Hudson, brahmareddy, haddopqa, yongjunzhang, vinodkumarvavillapalli, kilwallee	6 instances
Who has-blocker-severity?	Hive-13369, hbase-20201, hbase-19496, grrovy-8483	4 instances
Who has-critical-severity?	Hbase-15638, Hadoop-15273, Hdfs-8312, hdfs-13112	4 instances
Who has-major-severity?	Hive-10428, hadoop-13890, Hive-11112, Hadoop-13155, Hdfs-7707, Hive-12538, hadoop-14146, groovy-7535	8 instances
Who has a network-error?	Hdfs-13112, Hadoop-13890, Hive-11112, Hadoop-15273, Hadoop-13155, Hbase-20201, Hbase-20004, Hive-15827, hadoop-14146	9 instances
Who has authentication-error?	Hdfs-8312, Hdfs-13112, hbase-15638, hadoop-13890, Hadoop-13155, Hbase-20004, Hive-15827, hbase-13229, hadoop-14146	9 instances
Who has a log-edit-error?	Hdfs-13112, Hdfs-7707	2 instances
Who has a file-permission-error?	Hdfs-8312, Hbase-13732, Hive-13369, Hdfs-7707, Hbase-20201, hbase-13229, Groovy-7535	7 instances
Who has a name-node-error?	Hdfs-7707	1 instance
Who has application-consistency-error?	Hive-10428, Hbase-13732, Hive-13369, Hbase-19496, Hive-15827, groovy-8483, Hive-12538, groovy-7535	8 instances
Who has a byte-buffers-error?	Hbase-15638, hbase-19496	2 instances
Who has a severity?	Bug, Byte-buffer-error, name-node-error, java-error, log-edit-error, file-permission-error, application-consistency-error, configuration-error, input-output-error, network-error, authentication-error	12 subconcepts
Who has a contributor?	Bug, Byte-buffer-error, name-node-error, java-error, log-edit-error, file-permission-error, application-consistency-error, configuration-error, input-output-error, network-error, authentication-error	12 subconcepts
What is a network-error?	Bug, issue	2 superconcepts

The ontology is published on web using Ontology Web Language (OWL) as depicted in figure 7.

5 Threats to Validity

Threats to internal validity refer to whether conducted experiment deals with the causes and effects in specific domain ontology. Different concepts were identified by manual study and then compared with the concepts identified by the proposed techniques. It was reported that there was no significant difference in concepts identification between manual and proposed techniques.

Threats to external validity refer to the capability to generalize the results. The main threat is due to

the specific domain and tasks used in this work. As there is no ontology on software bugs, the data is collected is very small and concrete (about 21 bug reports of 5 different apache projects).

However, it includes various types of bugs based on classification of error such as network error, file permission error etc. In future, the proposed approach will be implemented on large number of bug reports of several distinct projects of apache software and with other techniques.

6 Conclusion and Future Work

In this research, an approach is proposed that uses Formal Concept Analysis to extract an ontology from the content analysis of bug reports. The main focus has been with software bugs domain and extracting knowledge from the bug reports reported in bug tracking systems.

The proposed approach extracts features (unigrams and bi-grams) and identifies various concepts and attributes. The concept lattice is created using formal concept analysis, which maps the relationship between concepts and attributes. This is further converted to ontology of bug reports through which knowledge is extracted using several small queries.

Several queries have been executed for information extraction and found 100% results on all the queries retrieval. It is concluded that proposed approach is feasible and useful in software bug domain.

It is due to the fact that frameworks with diagrams can be used for formal reasoning and yields in improved visualizations through concept-lattice. In software bug domain, the information is reported in the form of bug reports, which are large textual documents that are difficult to read and comprehend. With exploitation of FCA, large textual documents have been converted to a mathematical theory and are visualized through diagrams such as concept lattice. Further, concepts are arranged in hierarchical form, which are easy to interpret and retrieval of knowledge. Thus, proposed approach is feasible and useful in software bug domain [28].Thus, use of ontologies in the domain of bug reports is indispensable.

The work can be extended in future in many directions. First, several bug reports of distinct projects of various bug tracking systems can be used to construct a global ontology of software bugs. Second, along with formal concept analysis, fuzzy techniques can be integrated to create more complex ontology, which can be used to extract more knowledge through complex queries.

References

1. **Gruber, T.R. (1993).** A translation approach to portable ontology specifications by a translation approach to portable ontology specifications. *Knowledge Acquisition,* Vol. 5, No. 2, pp. 199–220.
2. **Obitko, M. (2001).** Ontologies description and applications. *Gerstner Laboratory for Intelligent Decision Making and Control*, pp. 1–35.
3. **Spear, A.D. (2006).** Ontology for the twenty first century : An introduction with recommendations. *Institute for Formal Ontology and Medical Information Science,* pp. 1–37.
4. **Garla, V.N. & Brandt, C. (2012).** Ontology-guided feature engineering for clinical text classification. *Journal of Biomedical Informatics*, Vol. 45, No. 5, pp. 992–998. DOI: 10.1016/j.jbi.2012.04.010.
5. **Juckett, D.A., Kasten, E.P., Davis, F.N., & Gostine, M. (2019).** Concept detection using text exemplars aligned with a specialized ontology. *Data & Knowledge Engineering Data*, Vol. 119, pp. 22–35. DOI: 10.1016/j.datak.2018.11.002.
6. **De Maio, C., Fenza, G., Loia, V., & Senatore, S. (2012).** Hierarchical web resources retrieval by exploiting fuzzy formal concept analysis. *Information Processing & Management,* Vol. 48, No. 3, pp. 399–418. DOI: 10.1016/j.ipm.2011.04.003.
7. **Yang, G., Zhang, T., & Lee, B. (2014).** Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. *IEEE 38th Annual Compute Software and Applications Conference,* pp. 97–106. DOI: 10.1109/COMPSAC.2014.16.
8. **Kanwal, J. & Maqbool, O. (2012).** Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, Vol. 27, No. 2, pp. 397–412. DOI: 10.1007/s11390-012-1230-3.
9. **Jin, K., Dashbalbar, A., Yang, G., & Lee, J. (2016).** Improving predictions about bug severity by utilizing bugs classified as normal 1. *Contemporary Engineering Sciences,* Vol. 9, No. 19, pp. 933–942. DOI: /10.12988/ces.2016.6695.
10. **Sharma, G., Sharma, S., & Gujral, S. (2015).** A novel way of assessing software bug severity using dictionary of critical terms. *Proceedings of the 4th*

International Conference on Eco-friendly Computing and Communication Systems, Vol. 70, pp. 632–639. DOI: 10.1016/j.procs.2015.10.059.

11. **Lamkanfi, A., Demeyer, S., Giger, E., & Goethals, B. (2010).** Predicting the severity of a reported bug. *7th IEEE Working Conference on Mining Software Repositories,* pp. 1–10. DOI: 10.1109/ MSR.2010. 5463284.
12. **Kaur, A. & Goyal, S. (2019).** Text analytics based severity prediction of software bugs for apache projects. *International Journal of System Assurance Engineering and Management,* Vol. 10, pp. 765–782. DOI: 10.1007/s13198-019-00807-8.
13. **Jindal, R., Malhotra, R., & Jain, A. (2016).** Prediction of defect severity by mining software project reports. *International Journal of System Assurance Engineering and Management,* Vol. 8, pp. 334–351. DOI: 10.1007/s13198-016-0438-y.
14. **Malhotra, R. (2013).** Severity assessment of software defect reports using text classification. *International Journal of Computer Applications*, Vol. 83, No. 11, pp. 13–16.
15. **Lessmann, S., Member, S., Baesens, B., Mues, C., & Pietsch, S. (2008).** Benchmarking classification models for software defect prediction : a proposed framework and novel findings. *IEEE Transactions on Software Engineering,* Vol. 34, No. 4, pp. 485–496. DOI: 10.1109/TSE.2008.35.
16. **Tran, H.M., Lange, C., & Chulkov, G. (2009).** Applying semantic techniques to search and analyze bug tracking data. *Journal of Network and Systems Management,* Vol. 17, pp. 285–308. DOI: 10.1007/s10922-009-9134-4.
17. **Tran, H.M. & Le, S.T. (2014).** Software bug ontology supporting semantic bug search on peer to peer networks. *New Generation Computing,* Vol. 32, pp. 145–162. DOI: 10.1007/ s00354-014-0203-1.
18. **Duarte, B.B., Falbo, R.A., Guizzardi, G., Guizzardi, R.S.S., & Souza, V.E.S. (2018).** Towards an ontology of software defects, errors and failures. *Conceptual Modeling (ER´18), Lecture Notes in Computer Science,* Vol. 11157. DOI: 10.1007/978-3-030-00847-5_25.
19. **Škopljanac, F. & Blaškovi, B. (2014).** Formal concept analysis – overview and applications. *24th DAAAM, International Symposium on Intelligent Manufacturing and Automation,* Vol. 69, pp. 1258–1267. DOI: 10.1016/j.proeng.2014. 03.117.
20. **Seganti, A.B. & Kapla, P. (2016).** Collaborative editing of ontologies using fluent editor and ontorion. *International Experiences and Directions Workshop on OWL*, Vol. 1, pp. 45–55. DOI: 10.1007/978-3-319-33245-1_5.
21. **Uren, V., Cimiano, P., Handschuh, S., Vargas-Vera, M., Motta, E., & Ciravegna, F. (2006).** Annotation for knowledge management : requirements and a survey of the state of the art. *Journal of Web Semantics,* Vol. 4, No. 1, pp. 14–28. DOI: 10.1016/j.websem. 2005.10.002.
22. **Stumme, G. & Maedche, A. (2001).** Bottom-up merging of ontologies. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Vol. 1, pp. 225–230.
23. **Hotho, A., Maedche, A., & Staab, S. (1998).** *Ontology-based text document clustering.* pp. 1– 13.
24. **Blomqvist, E. (2008).** Pattern ranking for semi-automatic ontology construction. *Proceedings of the ACM Symposium on Applied Computing March,* pp. 2248–2255. DOI:10.1145/ 1363686.1364224.
25. **Khoury, P.E., Coquery, E., & Hacid, M. (2008).** An ontological interface for software developers to select security patterns. *19th International Workshop on Database and Expert Systems Applications,* pp. 297–301. DOI: 10.1109/DEXA. 2008.110.
26. **Reinhartz, I. & Mark, B. (2019).** Extracting core requirements for software product lines. *Requirements Engineering,* Vol. 25, pp. 47–65. DOI: 10.1007/s00766-018-0307-0.
27. **Schuegerl, P., Rilling, J., & Charland, P. (2008).** *Enriching se ontologies with bug report quality*. pp. 1–15.
28. **Priss, U. (2006).** Formal concept analysis in information science. *Annual Review of Information Science and Technology*, Vol. 40, No. 1, pp. 521–543. DOI: 10.1002/aris.1440400120.
29. **Guo, J. & Gibiec, M. (2016).** Tackling the term-mismatch problem in automated trace retrieval. *Empirical Software Engineering,* Vol. 22, pp. 1103–1142. DOI: 10.1007/s10664-016-9479-8.
30. **Tilley, T., Cole, R., Becker, P., & Eklund, P. (2005).** A survey of formal concept analysis support for software engineering activities. *Formal Concept Analysis, Lecture Notes in Computer Science,* Vol. 3626, pp. 250–271. DOI: 10.1007/ 11528784_13.
31. **Cleary, B., Exton, C., Buckley, J., & English, M. (2009).** An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering,* Vol. 14, pp. 93–130. DOI: 10.1007/s10664-008-9095-3.
32. **Cimiano, P., Hotho, A., Stumme, G., & Tane, J. (2004).** Conceptual knowledge processing with formal concept analysis and ontologies. ***Eklund P. (eds)*** *Concept Lattices. ICFCA´04, Lecture Notes in*

Computer Science, Vol. 2961, pp. 189–207. DOI: 10.1007/978-3-540-24651-0_18.

33. **Formica, A. (2006).** Ontology-based concept similarity in formal concept analysis. *Information Sciences*, Vol. 176, No. 18, pp. 2624–2641. DOI: 10.1016/j.ins.2005.11.014.
34. **Hlomani, H. & Stacey, D. (2014).** Approaches, methods, metrics, measures, and subjectivity in ontology evaluation : A survey. *School of Computer Science, University of Guelph,* Vol. 1, pp. 1–5.
35. **Brewster, C., Alani, H., Dasmahapatra, S., Street, P., & Wilks, C.B.Y. (2004).** Data driven ontology evaluation. *International Conference on Language Resources and Evaluation.*
36. **Kaur, A. & Jindal, S.G. (2017).** Bug report collection system. *7th International Conference on Cloud Computing, Data Science & Engineering – Confluence.* DOI: 10.1109/CONFLUENCE.2017.7943241.
37. **Brank, J., Grobelnik, M., & Mladeni, D. (2005).** A survey of ontology evaluation techniques. *Proc. of 8th Int. Multi-Conf. Information Society,* pp. 166– 169.
38. **Data & Dimensions. (2016).** *The six primary dimensions for data*.
39. **Evo Ont-Software (2017).** *Evolution ontology.*
40. **IFI (2017).** https://files.ifi.uzh.ch/ddis/oldweb/ddis/research/evoont/.
41. **Baetle Bug and Enhancement Tracking Language (2017).** http://code.google.com/p/baetle/
42. **Helios-Bug Ontology (2016).** https://heliosplatform.sourceforge.net/ ontologies/helios_bt.html.

Article received on 29/10/2019; accepted on 06/03/2020. Corresponding author is Shubhra Goyal Jindal.