International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600
SKRGC Since 2012

http://www.ijcsjournal.com Volume 12, Issue 1, No 2, 2024. ISSN: 2348-6600
Reference ID: IJCS-494 PAGE NO: 3398-3411

EXAMINE THE CONVERGENCE OF SOFTWARE ENGINEERING'S MULTI-DOMAIN RESEARCH OPPORTUNITIES

V.VIJAYALAKSHMI
M.Sc Student,
Department of Computer Science,
Rajeswari College of Arts and Science for Women,
Bommayapalayam, Tamil Nadu, India.
Email ID: vijayalakshmivenugopal29@gmail.com

Dr.A.SARANYA
Assistant Professor,
Department of Computer Application,
Rajeswari College of Arts and Science for Women,
Bommayapalayam, Tamil Nadu, India.
Email ID: drsaranyarcw@gmail.com

Abstract

A significant part of the software industry is played by Software Engineering (SE), since almost every industry, business, and operation requires a specific type of software. Throughout the course of an application's lifecycle, software engineering offers a variety of levels that fulfill various purposes. The sector first created applications mostly for home usage and small businesses. At that point, the accounting data is also monitored, and the SDLC processes are highly minimal and controlled. The quantity of complex applications required by the software industry revolution clearly results in high-quality SDLC.

Keywords: Software Engineering, AI, Natural Language Processing, Data Mining, Big Data.

I. INTRODUCTION

The path of modern research is uncertain, particularly in the area of SE [1]. SE is always expanding and has a totally distinct dimension when it travels. There is a great need for a modern software development forum, and the SE should provide one. Not even the SDLC's core processing can meet the demands of the industry. Technology convergence contributes to SE's success and determinism. Big data, AI, NLP, data mining, and other data collection and processing technologies are frequently utilized in SE to deliver timely, valuable results.

The main research facets of SE and how it will integrate with other interdisciplinary approaches are highlighted in this study. Here's how the document is structured: The overview and fundamentals of software engineering are covered in Section II. The significant research contributions previously made by the researchers are highlighted in Section III. The convergence of data mining and SE is discussed in Section IV. This section went into great detail on how the concepts of data mining actually helped with the SDLC after large-scale applications were developed. In Section V, it is explained how AI helps with software development for a quick and economical SDLC.

In a few years, artificial intelligence (AI) will become a major area of convergence in SE due to the vast amount of research problems in the field. The use of Natural Language Processing (NLP) in text extraction in SE and multilingual software development is highlighted in Section VI. Section VII discusses ontologies and their contributions to the SE. The use of big data to derive knowledge from the vast array of SDLC data is covered in Section VIII. The paper is concluded in Section IX.

2. SINGLE SOFTWARE ENGINEERING COMPONENTS:

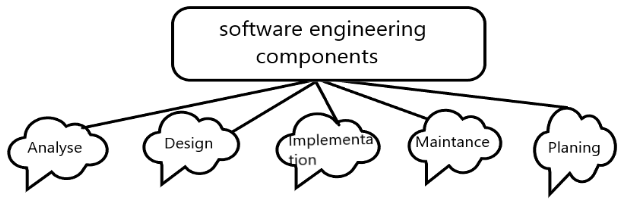

Figure 1: Single Components

Over the past fifty years, software engineering has changed; the term returned to general usage after it was used in the title of a conference on the topic organized by the North Atlantic Treaty Organization in 1968. Initially, software engineering was developed as a response to the perceived software crisis of the 1960s and 1970s, which referred to the difficulties organizations faced in delivering high-quality software systems on schedule and within budget.

The definition of software engineering (SE) is defined as "the application of engineering to software" [2]; that is, "the application of a systematic, disciplined, quantitative approach to the development, operation, and maintenance of software." As the subject has grown, software engineering (SE) has produced a range of methods for dealing with issues such software requirements, design, testing, and maintenance. Software development techniques such as the spiral model, waterfall model, and incremental development are successfully used to create high-quality software on schedule and within budget [3]. Agile software development has grown in acceptance as a more modern approach to developing sophisticated systems than the other more conventional approaches [4].

3. A REVIEW OF THE WORKS:

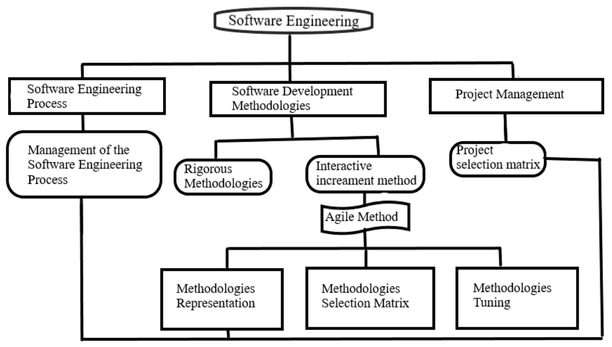
Software Engineering
Software Engineering Process
Software Development Methodologies
Project Management
Management of the Software Engineering Process
Rigorous Methodologies
Interactive increament method
Project selection matrix
Agile Method
Methodologies Representation
Methodologies Selection Matrix
Methodologies Tuning

Figure 2: Review of Literature

The most current uses of data mining in software engineering have been thoroughly surveyed by Taylor et al. [5]. They also go over potential problems with software data mining and the prerequisites for success. One of the most comprehensive papers on the topic is that produced by Halkidi et al. [6]. In-depth analyses of data mining techniques and their successful application in software engineering are featured in their work.

As per Aouf et al. [7], clustering algorithms can be employed to detect implicit patterns in data and derive valuable insights. Software fault detection via association rule mining was demonstrated by Chang and Chu [8].Gegick et al. [9] demonstrated the usefulness of text analysis in fault finding, whereas Runeson et al. [10] emphasized the importance of NLP in handling identical flaw issues. Islam and Brankovic [11] suggested methods for guaranteeing data mining privacy.

Mostly, this was accomplished by adding noisy data to certain areas of the dataset. However, iterative mining was proposed by Ma and Chan [12] in their effort to find overlapping patterns in noisy data. Ma and Chan were interested in using noisy data to protect privacy, but Islam and Brankovic [11] [12] addressed the process of removing noisy data in order to accomplish the goal of obtaining meaningful information.

Humans can understand natural language text with little difficulty. Viliam [13] discusses the significance of textual processing on natural language text. Farid talks about creating natural language writing using the UML class diagram. The purpose of this work is to reinforce the perspective of generating NL specification from class diagrams by describing several NL based systems. The study demonstrates how to generate semantically valid sentences that explain the structure of UML string names using WordNet [14]. To create class models, Reynaldo employs controlled NL text of requirements.

The study presents some preliminary findings from text parsing for ambiguity. The document presents the author's research strategy to incorporate requirement validation into the RAVEN project [15].

Design models and analysis models in UML can be generated from natural language text using an automated program called UMGAR, developed by Deva Kumar and colleagues. They have completed this assignment using Java RAP, Word Net 2.1, and Stanford parser [16]. By developing the SPIDER tool, Sascha and colleagues proposed a round trip engineering approach. Concerns of requirements-level flaws leaking into the design and coding phases were discussed in the article. Developers can create a UML model by using the behavioral features that are presented in the NL text [17].

To date, numerous synergies between software engineering and ontologies have been presented by researchers. Within the domains of software modeling [25], software engineering [24], model transformations [26], software maintenance [27], software understanding [28], software methodologies [29], and software community of practice [30], for instance, ontologies are suggested for use. Additionally, methods for reasoning and modeling over ontologies are suggested using software engineering tools.

The convergence of ontologies and software engineering has garnered the interest of standards groups, leading to ongoing initiatives in this area. In an effort to create best practices for utilizing ontologies in software engineering, Ontology-Driven Architecture (ODA) was developed by the Software Engineering Best Practices Working Group of the W3C [31]. Perhaps the most significant result to date is the Ontology Definition Metamodel (ODM), which is proposed to be the standard for the Object Management Group (OMG) [32].

Model-driven engineering principles enable the software development process to incorporate ontology languages, or ontologies, through the use of the ODM standard [33]. Although these numerous efforts demonstrate many benefits for different aspects of software and ontology engineering or offer an in-depth overview of the state of the art in the field [29, 33], none of them employ a comprehensive software lifecycle framework to assess and examine the applications of ontologies in various software engineering domains.

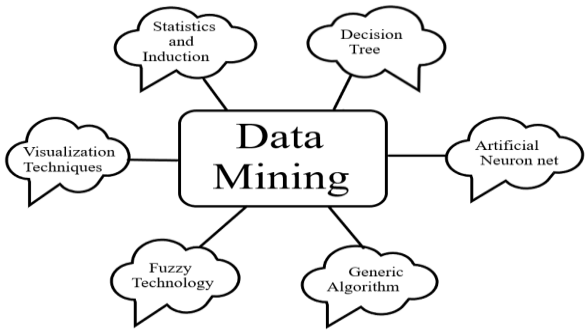

Figure 3: Data Mining

Data mining has been proposed in research works as a means to help industrial-scale software system maintenance, debugging, and testing because of its capacity to manage huge volumes of data and its effectiveness in identifying hidden patterns of knowledge (Table 1). The findings of the mining process will help software system engineers anticipate software failures, extract and categorize frequent problems, validate relationships across library categories, analyze defect data, find recurring patterns in source code, and ultimately alter the development process.

Practitioners and researchers in victimization data mining can generally utilize the data from software system engineering to better manage their projects and create software systems that are of higher quality that are delivered on schedule and under budget. Three things are fundamentally wrong with data mining for software system engineering [18]: the objective, the input data that is used, and the mining technology that is employed. Enhancing code completion tools, for instance, is another objective [19].

Table 1: Phases of Development Utilizing

Software Creation Phase	Approaches to Data Mining	Incoming Data	Findings from Data Analysis
Requirement Elicitation Stage-1	Classification of Data	Documentation of Input Data	Classification of Requirements
	Text Mining	Lists of Mailing	Data Compilation
Design Stage-2	Clustering Analysis	Design Document	Compiling and Tagging Data
Implementation Stage-3	Clustering	Source Code	Software Operations
	Classification	Manager of Software Configuration	Bug Monitoring
	Text	Mining Code of Reference	Bug Tracking
	Regular Association and Pattern Mining Guidelines	Defect	Error Correction
		The graph of program reliance	Neglected Conditions
Testing stage-4	Classification	I/O variables in the program framework	An infrastructure that generates Sets for Functional Testing
		Program Runs	Classifiers of Software Behaviour
	Clustering	Qualities of Performance	Different types of Performance Profiles

Requirements Elicitation Stage-1:

Obtaining and specifying the requirements for a software system is called requirements elicitation. The purpose of requirements elicitation is to guarantee that a thorough and precise understanding of the needs and requirements of the client forms the foundation of the software development process. The process of identifying, gathering, analyzing, and fine-tuning software system requirements is known as requirements elicitation.

This crucial task is usually completed at the start of the project and is a part of the software development life cycle. Business owners, end users, and technical specialists are among the stakeholders involved in requirements elicitation from across the firm. Clear, succinct, and well-defined requirements are the end result of the needs solicitation process, and they form the basis for the software system's design and development.

Design Stage-2:

Programmers use software design as a tool to help with software coding and implementation by transforming user requirements into a viable form. This involves converting the requirements of the client—which are outlined in the Software Requirement Specification (SRS) document—into a format that can be readily implemented using programming language.

The first stage of the Software Design Life Cycle (SDLC) shifts focus from the problem domain to the solution domain and is known as the software design phase. A system is viewed as a collection of parts, or modules, having distinct behaviors and bounds in software architecture.

Implementation Stage-3:

Software implementation is the process of incorporating an application into the daily operations of a business. Choosing a provider and setting a budget usually mark the start of the process. The installation of the program, data migration, and feature testing could be the next phases. When implementing new project management, supply chain management, and enterprise resource planning tools, businesses employ software implementation. When a development team modifies an already-existing application, the same procedure may be used.

Testing Stage-4:

Software testing is a technique for evaluating a software program's functionality. The procedure verifies that the program is free of bugs and determines whether it actually satisfies the anticipated criteria. Finding flaws, defects, or missing requirements in relation to the real requirements is the goal of software testing. Its primary objective is to gauge an application's or software program's specifications, functionality, and overall performance.

5. PROGRAMMING AND SOFTWARE ENGINEERING AUTOMATION:

The last several years have witnessed a significant change in the field of software engineering. Software development should become more dependable and easier with the use of artificial intelligence (AI) and software intelligence technologies. Machine-driven testing and issue detection tools are the applications of AI that most effectively improve software development, according to a Forrester research paper [20] on the subject (Fig 1).

The software development industry can be permanently changed by artificial intelligence (AI) in the era of machine learning. Nowadays, the focus is more on choosing the right data to train a neural network that can solve a particular problem without the need for human involvement than it is on process if-then-else loops. This might revolutionize how problems are resolved, the resources employed, the way people think, and even the idea of what a developer is. We will examine several methods that artificial intelligence (AI) can improve software development, as well as some potential drawbacks and the reasons this strategy works.

- **Fast Prototyping:** Traditionally, it takes months or even years to design a technology product that meets business needs. However, machine learning is speeding up this process by allowing faculty members who are not as technically skilled to develop technologies that use either visual or linguistic interfaces.
- **Intelligent Programming Assistants:** Reading documentation and troubleshooting code take up a significant portion of developers' work. This time will be reduced by wise programming assistants who offer just-in-time assistance and recommendations, such as pertinent documents, best practices, and code samples. Examples of such assistance are Codota for Java and Kite for Python.
- **Automated Analytics & Error Handling:** During the development phase, programming assistants can automatically identify and indicate frequent errors based on prior knowledge. Machine learning can be used to evaluate system logs after a technology has been implemented in order to promptly and even proactively identify faults. It may even be possible in the future to modify the software system so that it

International Journal of Computer Science
Scholarly Peer Reviewed Research Journal - PRESS - OPEN ACCESS
ISSN: 2348-6600
http://www.ijcsjournal.com Volume 12, Issue 1, No 2, 2024. ISSN: 2348-6600
Reference ID: IJCS-494 PAGE NO: 3398-3411

reacts to mistakes automatically and without the need for human input.

- **Automated Code Reworking:** Clear code is essential for long-term maintenance and teamwork. Large-scale refactoring is a necessary and perhaps painful need when businesses modernize their technologies. The practice of examining code and automatically boosting its readability and efficiency will be instinctive to machine learning.
- **Accurate Estimates:** It is well known that software development exceeds both budget and schedule constraints. Trustworthy approximations necessitate in-depth knowledge, contextual awareness, and acquaintance with the implementation team. Machine learning will be trained on historical project data, such as user stories, feature definitions, estimates, and actuals, in order to make more precise predictions about work and expense.
- **Making Strategic Decisions:** You spend a lot of time arguing over which options and products to cut or grade. Business executives and engineering teams will be able to identify efforts that may optimize impact and minimize risk with the help of an AI solution that has been trained on all previous development projects and business aspects. The AI solution will also evaluate the performance of current apps.

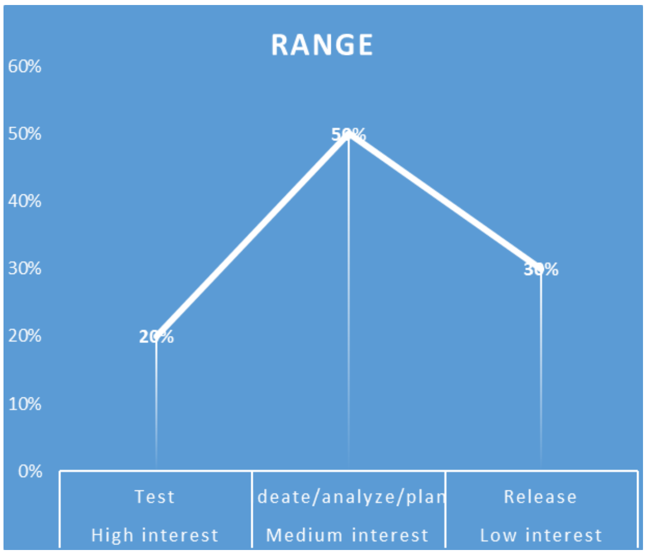

Figure 4: AI for SDLC

High Interest:

- ✓ Examine Technical Coding.
- ✓ Develop test cases proactively and suggest a Testing Strategy.
- ✓ Discover faults and target testing on the area's most likely to contain flaws.
- ✓ Acknowledge graphics and user interface
- ✓ Apply machine learning to forecast test results in the future.
- ✓ Reduce business risk and expand the Scope of Testing.

Medium Interest:

- ✓ Evaluate trends between Projects.
- ✓ Determine the Requirements.
- ✓ Offer options for Reusing.
- ✓ More Comprehensive Specifications.

Low Interest:

- ✓ Improve distribution Protocols.
- ✓ Reduce Flaws.
- ✓ Conduct log analysis to improve Traffic Routing.

6. NLP IN CONJUNCTION WITH SOFTWARE ENGINEERING:

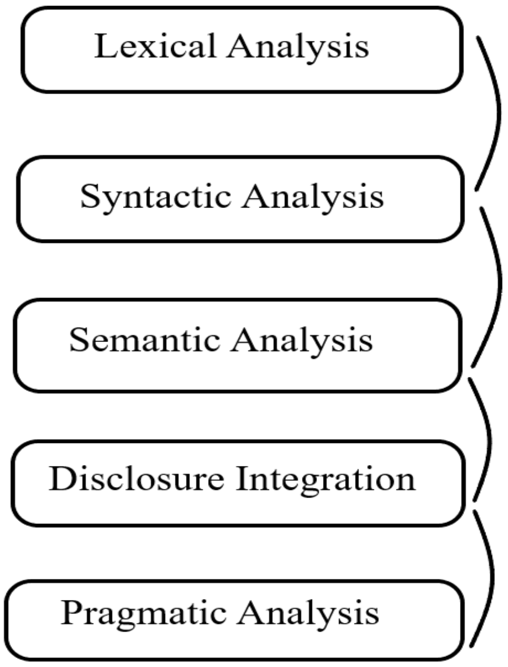

Figure 5: NPL in Conjunction with Software Engineering

The phases of the Software Development Life Cycle are as follows. These stages include instructions for creating software packages. Natural language processing can be used in all relevant Software Development Life Cycle sections. Its further accurate application occurs when the component or activity's artifacts are text files. For jobs involving natural language processing, plain text may be utilized as input. Textual generation basically refers to all the processes involved in human interpretation of the material.

The assumption that every requirement for a future system notably exists in textual type is not supported by the evidence, as implied by a number of NLP-based requirements capture approaches [21]. While it is true that some data is naturally presented as text—typically, process descriptions or pre-established procedures—a much greater amount of information can be found in diagrams or in the client's physical environment.

It is obviously unrealistic to rely solely on the text for information or to expect the client to reduce all of his or her requests to textual form. Tools to scan, search, browse, and classify that text undoubtedly could be helpful in creating a comprehensive and precise statement of wishes, assuming, however, that the requirements definition task is being carried out by an intelligent human and that a significant body of the machine-readable text is accessible [22]. This may not necessarily indicate that a free text is automatically understood.

6. SEARCH ONTOLOGIES:

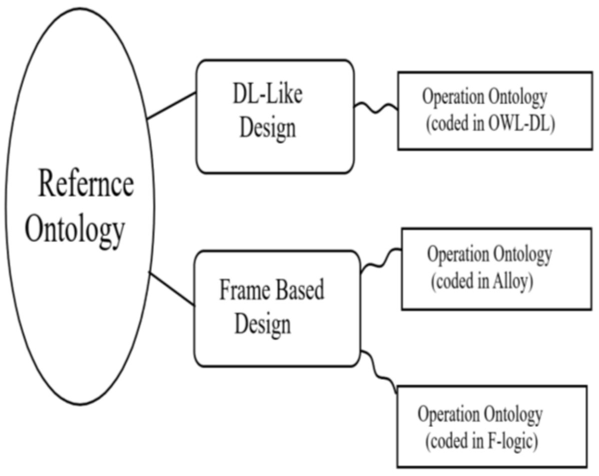
Refernce Ontology
DL-Like Design
Operation Ontology (coded in OWL-DL)
Frame Based Design
Operation Ontology (coded in Alloy)
Operation Ontology (coded in F-logic)

Figure 6: Search Ontologies

Domain knowledge can be formalized through the use of ontologies. The main application of ontologies is knowledge exchange, which is done through knowledge-based applications. The main target audience for ontology development is AI experts who are familiar with a range of methods from the field of AI. However, a sizable portion of the software business is unaware of this information. Many suggestions have been made to use ontologies in software engineering to close the knowledge gap between practitioners and AI approaches.

Ideas include using UML diagrams while creating ontologies. As a matter of fact, the university-developed recipient software includes a pane that clearly specifies how UML diagrams are used in ontology construction. From the created ontology, a UML diagram will be created using this practicality. Nevertheless, It is also clear that the visualization of semantic ideas derived from describe reason and other concepts found inside the semantic web languages is unaffected by software engineering approaches alone.

7. MASSIVE DATA PERTAINING TO SOFTWARE ENGINEERING:

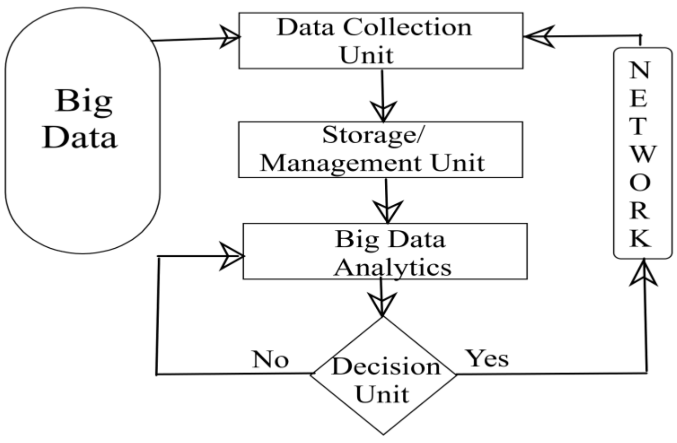

Figure 7: Massive Data Pertaining

Software packages manage ever-growing datasets in addition to offering novel algorithms, state-of-the-art system architectures, and system structures ready to tackle each of the five Vs of massive data [23]. The time has come for software to leverage knowledge extracted from enormous amounts of data, which include user profiles, coding patterns, engages in, software source code, issues from retracing systems, caution and updates, errors, and all types of logs.

The challenges facing software engineering research in this area include creating new tools that collect data from sensor-based systems and employ data mining and machine learning approaches to reveal knowledge that is not available to individuals. But in order to improve software quality, it must be brought to people's attention and felt. To better understand what users want from systems, this involves studying about the creation and decommissioning software structures, freely available parts, and consumers patterns, choices, and conduct. It also involves learning about tools and techniques for spotting chances for unique feature and performance upgrade. These include using log files (big, gigabytes, or updated very quickly) returned from multiple complex distributed systems and infrastructures to identify the specific root causes of failures and system halts10; obtaining execution information on indications and shifts in context that trigger adaptations; and executing predicting and predictive analytics for assertive strategy and preparation of modification acts.

8. CONCLUSION

This paper makes a compelling case for the collaboration of software engineering with other fields in order to create better software and applications. New text processing techniques are required due to the growing amount of data related to software development. Technologies like big data and data mining will offset the need for text processing. The fields of AI and NLP are those that can help SE create fast, error-free application development. The research collaborations, difficulties, and future directions in various disciplines are described in this publication.

REFERENCE

[1] Xu, Haiping, Future Research Directions of Software Engineering and Knowledge Engineering. International Journal of Software Engineering and Knowledge Engineering. 415-421, 2015.

[2] Abran, A., Moore, J.W., Bourque, P., Dupuis, R., Tripp, L.L.: Guide to the Software Engineering Body of Knowledge, IEEE, 2004

[3] Bhalerao, Shilpa & Puntambekar, Devendra & Ingle, Maya, Generalizing Agile Software Development Life Cycle. International Journal on Computer Science and Engineering. 1, . 2009.

[4] Agile Manifesto, http://agilemanifesto.org

[5] Taylor, Q.and Giraud-Carrier, C. "Applications of data mining in software engineering", International Journal of Data Analysis Techniques and Strategies, Volume 02, Issue 03, Page No (243- 257), July 2010.

[6] M. Halkidia, D. Spinellisb, G. Tsatsaronisc and M.Vazirgiannis, "Data mining in software engineering", Intelligent Data Analysis 15, Page No (413–441), 2011.

[7] M. Aouf, L. Lyanage, and S. Hansen, "Critical review of data mining techniques for gene expression analysis," International Conference on Information and Automation for Sustainability (ICIAFS) 2008, Page No (367-371), 2008.

[8] C. CHANG and C. CHU, "Software Defect Prediction Using Inter transaction Association Rule Mining", International Journal of Software Engineering and Knowledge Engineering, Volume 19, Issue 06, Page No (747-764), September 2009.

[9] M. Gegick, P. Rotella and T. Xie, "Identifying security bug reports via text mining: an industrial case study", Mining Software Repositories (MSR), 7th IEEE Working Conference, Page No (11– 20), 2010.

[10] P. Runeson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing", Software Engineering, 2007. ICSE 2007. 29th International Conference, Page No (499 – 510), 2007.

[11] M. Z. Islam and L. Brankovic, "Detective: a decision tree based categorical value clustering and perturbation technique for preserving privacy in data mining," Third IEEE Conference on Industrial Informatics (INDIN), Page No (701-708), 2005.

[12] P. C. H. Ma and K. C. C. Chan, "An iterative data mining approach for mining overlapping coexpression patterns in noisy gene expression data," IEEE Trans. NanoBioscience, Volume 08, Issue 03, Page No (252-258), September 2009.

[13] V. Simko, P. Kroha and P. Hnetynka, "Implemented domain model generation", Technical Report, Department of Distributed and Dependable Systems, Report No. D3S-TR-2013-03, 2012.

[14] F. Meziane, N. Athanasakis and S. Ananiadou, "Generating Natural Language Specifications from UML Class diagrams", Requirement Engineering Journal, Springer-Verlag, London, vol. 13, no. 1, pp. 1-18, 2013.

[15] R. Giganto, "Generating Class Models through Controlled Requirements", New Zealand Computer Science Research Conference (NZCSRSC), Christchurch, New Zealand, 2008.

[16] G. Lu, P. Huang, L. He, C. Cu and X. Li, "A New Semantic Similarity Measuring Method Based on Web Search Engines", WSEAS Transaction on Computer, ISSN: 1109-2750, vol. 9, Issue 1, 2010.

[17] S. Konrad and B. H. C. Cheng, "Automated Analysis of Natural Language Properties for UML Models", [Online available], 2010.

[18] T. Xie, S. Thummalapenta, D. Lo and C. Liu, Data mining for software engineering, Computer 42, 55–62, 2009.

[19] Marcel Bruch, Martin Monperrus, Mira Mezini. Learning from Examples to Improve Code Completion Systems. In Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM Symposium on the Foundations of Software Engineering, ACM, 2009.

[20] https://reprints.forrester.com/#/assets/2/108/'RES121339'/reports

[21] M Saeki, H Horai, H Enomoto, "Sofitware Development Process from Natural Language Specification", 1lth International Conference on Software Engineering, 1989.

[22] P Loucopoulos P & R E M Champion, "Concept Acquiisition and Analysis for Requirements Acquisition", IEE Software Engineering Journal, (2) 1990.

[23] Kalbandi, Ishwarappa & Anuradha, J, A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology. Procedia Computer Science, 2015.

[24] Seok Won Lee and Robin A. Gandhi. Ontology-based Active Requirements Engineering Framework. In Proc. of the 12th Asia- Pacific Software Eng. Conf. 481–490, 2005.

[25] Holger Knublauch. Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protege/OWL. In Proc. of 1st Int'l WSh on the Model-Driven Semantic Web, 2004.

[26] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In Proc. of the ACM/IEEE 9th Int'l Conf. on Model Driven Eng. Languages and Sys., pages 528–542, 2006.

[27] Christoph Kiefer, Abraham Bernstein, and Jonas Tappolet. Analyzing Software with iSPARQL. In Proc. the 3rd ESWC Int'l WSh. on Semantic Web Enabled Software Eng., 2007.

[28] Ren´e Witte, Yonggang Zhang, and Juergen Rilling. Empowering Software Maintainers with Semantic Web Technologies. In Proc. of the 4th European Semantic Web Conference, pages 37–52. Springer, 2007.

[29] C´esar Gonz`alez-P`erez and Brian Henderson-Sellers. An Ontology for Software Development Methodologies and Endeavours, volume Ontologies for Software Engineering and Software Technology, pages 123–151. Springer, 2006.

[30] Anupriya Ankolekar, Katia Sycara, James Herbsleb, Robert Kraut, and Chris Welty. Supporting Online Problem-solving Communities with the Semantic Web. In Proc. of the 15th Int'l conf. on WWW, pages 575–584, 2006.

[31] P. Tetlow, J. Z. Pan, D. Oberle, E. Wallace, M. Uschold, and E. Kendall. Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. W3C Working Draft, 2006.

[32] OMG ODM. Ontology Definition Metamodel, 2006. http://www.omg.org/cgibin/doc?ad/06-05-01.pdf.

[33] Hans-J¨org Happel and Stefan Seedorf. Applications of Ontologies in Software Engineering. In Proc. of the Int'l WSh. on Semantic Web Enabled Software Engineering, 2006.