SURVEY ON INFORMATION RETRIEVAL AND NATURAL LANGUAGE PROCESSING METHODS IN SOFTWARE ENGINEERING TASKS

Rajat Sarin
Sayantani Goswami
Siddarth Udayakumar

1 CONTENTS

Survey on Information Retrieval and Natural Language Processing in Software Engineering Tasks.

2 INTRODUCTION

In the field of software engineering, there are daily changes to software involving software code, bug reports for the software code. These changes can be ascertained by getting information from the software

2.1 INFORMATION RETRIEVAL

2.2 NATURAL LANGUAGE PROCESSING

3 SCOPE

The scope of this survey was to observe and report our findings from the different topics. Some of the research relied on Information retrieval methods and some of the papers relied on Natural Language Processing. There are different software engineering tasks but we chose a few tasks and read research papers dealing with the tasks. Our objective was to obtain information from the papers provided and provide a survey of what each paper related to the topic dealt with.

4 SURVEY PROCESS

For this survey, we have taken up the following software engineering techniques. Initial survey just concentrated on only a single topic – Feature/Concept/Bug Localization. But, the scope was widened to include other topics that were of our interest. The topics chosen for this survey were taken based on the number of papers available and the type of method used by the authors of the papers in their research. The topics that were chosen are as follows:

1) Automatic Document Generation.
2) Concept/Feature/Bug Localization
3) Software Re-use
4) Source Code Pre-processing
5) Bug Triage
6) Traceability

We read the introduction to the papers, get information about the method or tool the authors of the papers relied on for their research, check the hypotheses or research questions the authors attempted to answer, analyze the final results for the research and infer about the methods used. Under analysis, we checked the final results used, how the results were judged and checked for any trends or if a particular method was similar to another method.

All the information obtained from the paper was documented in word documents. Each summary of the paper consisted of the important details of the paper including the method used, analysis of methods and the results obtained. Finally, all of the collected information was consolidated to form the final survey report we have here.

All the papers here relied on the standard Information Retrieval technique or techniques that used Natural Language Processing or Machine Learning. Some papers used a combination of information Retrieval and Natural Language Processing in the form of tools or new algorithms.

The following table Table1 shows the number of papers covered for the survey. The total number of papers used for the survey came up to 163 papers.

TOPIC	NUMBER OF PAPERS
Concept/Feature/Bug Localization	50
Bug Triage	26
Traceability	37
Source Code Pre-processing	10
Automatic Document Generation	24
Software Re-use	16
Total	163

Table 1

Majority of the papers chosen for the survey were published between 2011 and 2015. But, we have also included a few more papers that were published late in 2015 and in 2016. Some of the newer papers, especially after 2013, used Natural Language processing techniques and Machine Learning techniques for performing the tasks listed above.

5 INITIAL OBSERVATIONS

Before we get started with the information we observed from each of the topics, we initially read the all the papers and we observed that all the topics we have chosen for this survey covered the standard Information Retrieval techniques involving Vector Space Modelling, Latent Semantic Indexing and Latent Dirichlet Allocation. Some of the papers, especially those which were published after 2013 relied on Natural Language Processing by using their own algorithms or tools. Some of the papers relied on

combining Information Retrieval with Natural Language Processing while some combined Information Retrieval with techniques like Web Mining [1]. Papers relying on Natural Language Processing involved a form of machine learning where the algorithms were trained using data. Out of all the topics listed, we noticed that all the topics relied more on standard, proven Information Retrieval techniques except for Automatic Document Generation where most of the papers had their base in Natural Language Processing. The results for Automatic document generation were also found to be efficient and quite reliable. Information Retrieval was also deemed to be more reliable in topics such as traceability, source code pre-processing and feature/concept localization. In topics related to software re-use, both Information Retrieval and Natural Language Processing techniques produced similar results. The same conclusion holds true for the papers on bug triage.

6 TOPICS

The details about the survey for each individual topic considered is listed down below. We have analyzed each topic separately and have provided details about what we observed from the papers related to the topics.

6.1 AUTOMATIC DOCUMENT GENERATION

Automatic Document Generation is a process of automatically generating textual description or code documentation from the software's source code. This automatic generation of features is not limited only to source code. It can be applicable in creating test cases, comments in the source code and textual description of a method or a feature in the source code. This process can also be used for generating documentation about how a program or a software can function based on analysis of the software's source code.

We took into consideration 24 papers for that dealt with automatic document generation. All of these papers relied on either the standard Information Retrieval methods or Natural Language Processing techniques. But the paper By McBurney and McMillan [2] used a combination of Information Retrieval and Natural Language Processing.

Some of the Information Retrieval techniques that were discussed included using Vector Space Models and Latent Semantic Indexing [3], Term Frequency – Inverse Document Frequency (Tf-Idf) [5], new approaches using custom tools such as SCAN [5] and using eye tracking on software programmers [6]. Another paper [7], described a tool called CODES (mining sourCe cOde Descriptions from developEr diScussions) which extracts documentation from forums like Stack Overflow and creates JavaDoc Descriptions from it. There were other methods such as the tool ACE, suggested by Rigby and Robillard, which used an island parser to identify code elements [8].

Buse and Weimer [4] in their paper devised an automatic technique for documentation for changes in software programs. Their paper used Natural Language Processing to generate human readable documentation for program changes using the version control log messages. They achieved this using their algorithm called DeltaDoc for documentation. The algorithm was trained using 1000 messages from software logs and was used to answer the "whats" and "whys" of documenting the software changes to a program. The results were compared with human written log messages and checked by human developers to see if the algorithm produced a much more correct output. The result found was that

DeltaDoc produced better results 89% of the time. This was a technique that relied entirely on NLP. Similarly, there were other papers that relied on Natural Language Processing [9], [10], [11] which used JSummarizer (A tool to automatically generate natural language summaries of Java Classes), UnitTestScribe (Another tool that combines static analysis, NLP, backward slicing and code summarization to generate natural language documentation of unit test cases) and CloCom (A tool that analyzes software repositories for comment generation) respectively.

We observed that Automatic document generation had the most number of papers that relied on Natural Language Processing (NLP) as a technique. Also, most of the papers were published after 2013, when NLP started to get used in research as a more automated way of performing software engineering tasks. We obtained the accuracy values for how good a NLP based tool worked in the paper and created a small graph (Fig 1). The graph uses accuracy values for three projects considered from papers [4], [12] and [11] respectively. For Delta Doc, the accuracy value corresponds to accuracy of generated documentation. [12] dealt with generating parameter comments in the source code the accuracy value corresponds to automatically generated comments for Java methods. Finally, [11] dealt with automatically generated comments for the source code. The accuracy of the comments generated was low but the number of bad comments generated was high. Also, please note that the accuracy values shown here are not for the same data sample. Each project was tested with its own set of data varying in numbers.

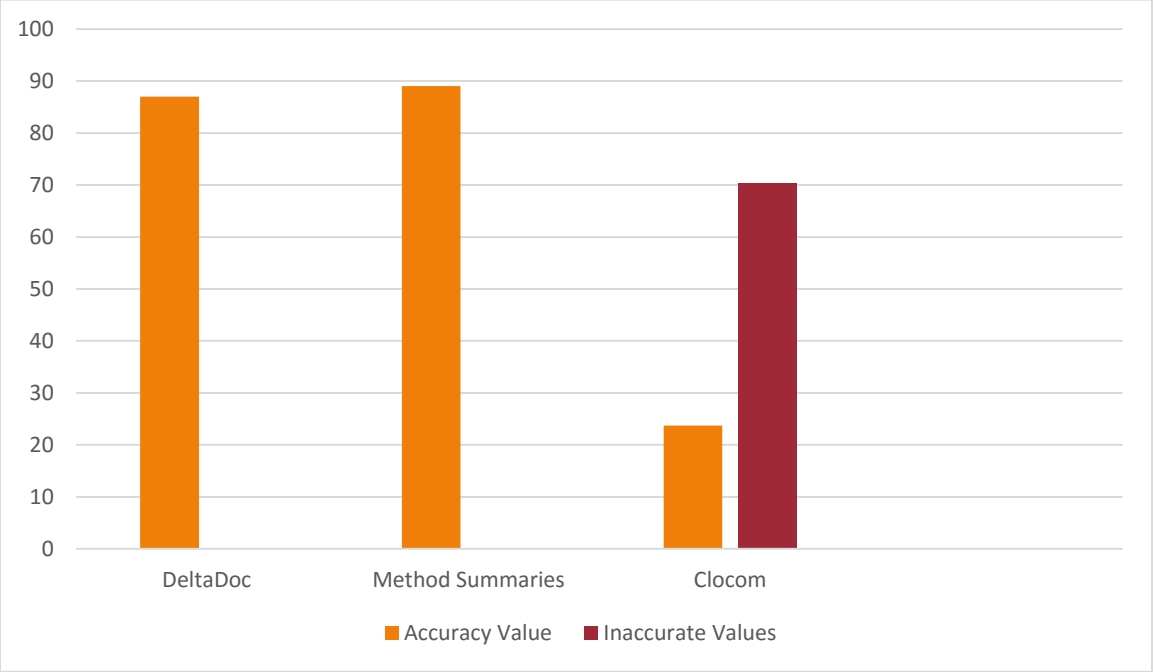

Fig1

6.2 Feature/Concept Location and Bug Localization

The other topic that we chose spans over two other topics – Feature Localization and Bug Localization. Concept location is the technique of identifying parts of a software system that implements a specific

concept that originates from the problem or the solution domain [13]. Feature Location is a technique that aims at locating software artifacts that implement a specific program functionality which is also known as a feature [14]. In Bug Localization, a developer uses information about a bug in the software to locate the portion of the source code to modify to correct the bug. The papers we read for this topic presented a method of identifying either a concept, a feature or a bug by combining different sources of information for efficient location. From the papers we observed, we noticed that both Information Retrieval techniques (IR) and Natural Language Processing (NLP) techniques were used for research. In certain cases, Information Retrieval techniques were used in combination with other techniques (Example: Web mining) were used for feature location. There was a single instance where Information Retrieval was used in combination with Natural Language Processing for feature location.

We observed that for all the topics – feature, concept location and bug localization, the majority of the papers relied only on Information Retrieval techniques in standard form or in the form of custom research tools developed by the authors. Even papers that were published recently preferred to stick to Information Retrieval techniques. Very few papers used Natural Language Processing as a technique. Some more new papers after 2014 for bug localization relied on Natural Language Processing but then again, most of the research relied on Information Retrieval tasks crediting IR for being more reliable in this aspect.

Lukins, Kraft and Etzkorn [15] used Latent Dirichlet Allocation (LDA) which is an extensible IR model that has more advantages than Latent Semantic Indexing (LSI) for bug localization. They used their method on Mozilla browser that was written in the C++ language and found that in LDA performed better than other approaches that used LSI in this regard. Thung, Le and Kochhar [16] in their paper developed a tool called BugLocalizer which was implemented as a Bugzilla plugin. BugLocalizer accepts a bug report and the source code repository as input and returns a list of files in the repository that may be the cause for the bug listed in the bug report. BugLocalizer was compared to another tool called BugLocator which relied on revised version of the Vector Space Model. BugLocalizer also relies on the same Information Retrieval based technique (Vector Space Model). Thung et Al found that BugLocalizer also performed better than other tools or methods that relied on Information Retrieval techniques like VSM and Latent Semantic Indexing (LSI) [16].

Tian, Yuan et Al [17] developed a bug localization method that had its base in Natural Language Processing by using multi-factor analysis. They proposed an approach based on Machine Learning that would assign a priority to a bug based on the information available. They extracted different factors from the bug report including author, severity, related reports, temporal data and the product. These factors are used as features to train the Machine Learning Model through a classification algorithm. Experiments based on this research showed that for a data set consisting of more than 100,000 bug reports from Eclipse, that their approach outperformed the baselines by 209%.

Moving on to feature and concept location, we found that there were more Information Retrieval tasks that used standard Information Retrieval techniques like Vector Space Models, Latent Semantic Indexing (LSI) and variations on the Vector Space Models. Marcus, Sergeyev, Rajlich & Maletic [13] in their research paper used Latent Semantic Indexing to address the problem of concept location. In their work, they pass the source code for pre-processing and obtain a corpus of the source code. Then the corpus is subjected to Single Value Decomposition (SVD) to create the LSI space. User queries to the LSI space will result in the answers for the query. Their hypothesis was that LSI allows the user to identify all functions in the

document that implement the concepts or use the concepts. They also used partially generated automatic queries apart from user queries. The case study for their work on the NCSA Mosaic browser showed that using LSI was more advantageous since the method is language independent and LSI was able to identify words and identifiers from the source code that are related to a user-specified term or phrase within the context of the software system. In another case, Concept location was performed by using Latent Semantic Indexing in combination with web mining [18]. Other papers used IR techniques like Vector Space Models and Latent Dirichlet Allocation.

Concept Location was also performed by using Natural Language Processing in the paper by Hill, Shanker and Pollock [19]. In their paper, they proposed a search technique that uses information such as position of the query word and its semantic role to calculate relevance. This paper used Phrasal Concepts to improve software search. Their approach, called PCT scores source code by weighing query words based on their location in the code, their semantic role, head distance and usage information. They compared this to another search technique called GES which takes in a natural language query and returns a ranked list of methods. They found that PCT performed better than GES. Another Paper that dealt with NLP for concept location was by Martie and Van Der Hook [20] who proposed an algorithm called MorewithLess and MWL-ST Hybrid which use relevance, diversity and conciseness in ranking code search results.

We observed that Information Retrieval techniques that relied on standard methods like Latent Semantic Indexing and Vector Space Models and their variations were used more than Natural Language Processing techniques. The results were improved when the IR techniques were used in combination with other techniques and sources.

6.3 SOFTWARE RE-USE

If you see today, systems are designed and developed by composition of the existing components that are parts of other systems. This happens in most of the engineering disciplines. The focus of Software Engineering has been to develop new and original systems but today it is more recognized as reusing the existing components and software systematically to achieve better software efficiently at a much lower price. The principle of software reuse is based on three types: Application System Reuse, Component Reuse and Object and Function Reuse. In Application reuse, the entire application either by incorporating it without doing any change to the other systems or by development of application families is carried out. Component reuse involves reusing components of an application (sub-systems or single objects). Object and function reuse involves reusing the software components that use a function or a well-defined object.

Some of the papers that we read dealt with some of the tools and techniques involved in software engineering and software reuse. The below papers were based on various indexing methods. They described concepts of Information Retrieval (IR) and Natural Language Processing (NLP). The paper "An Information Retrieval Approach for Automatically Constructing Software Libraries" by Maarek, Y. S., Berry, D. M., and Kaiser, G. E. [21] proposed a design and a tool called GURU, that automatically assembles conceptually structured software libraries from a set of unindexed and unorganized software components. We saw how Guru extracts the indices from the natural language documentation associated with the software components by using an indexing scheme that is based on lexical affinities and their statistical distribution. The paper "Full Text Indexing Based on Lexical Relations, an Application: Software Libraries" by Maarek, Y. S. and Smadja, F. A. [22] discussed about GURU as well and explains the tool in contrast to classical indexing scheme, where GURU's scheme does not require any priori information on the context

of the document to be analyzed, and provides a conceptual representation of the document without attempting to actually understand it. The next paper “Assessing software libraries by browsing similar classes, functions and relationships" by Michail, A. and Notkin, D. [23] presented two matching techniques which involves matching of similar classes and functions across libraries, as well as various relationships between them i.e. name matching and similarity matching. The paper also demonstrates a tool called CodeWeb to show how their approach can be applied in practice. The paper “An experiment in software component retrieval” by H. Mili, E. Ah-ki, R. Godin, and H. Mcheick [24] involved a research that centers around exploring methodologies for developing reusable software, and developing methods and tools for building inter-enterprise information systems with reusable components. The researchers set out to develop, evaluate, and compare two classes of component retrieval methods which, supposedly, strike different balances along the costs/benefits spectrum, namely, the (quasi-) zero-investment free text classification and retrieval versus the ‘up-front investment-laden’ but presumably superior controlled vocabulary faceted indexing and retrieval.

The paper “Applying information-retrieval methods to software reuse: a case study” by Stierna, E. and Rowe, N. [25] this paper examined the effectiveness of information-retrieval methods to find matching English requirements for two software systems. While a certain amount of human judgment is always necessary in determining reuse, the goal as per this research was to greatly reduce the necessary effort. The next paper “Using an information retrieval system to retrieve source code samples” by Sindhgatta, R. [26] proposed a tool called ‘JSearch’ where sample source code, published by the developers is preprocessed and indexed. The client portion of the tool is available as a plug-in in Eclipse IDE and as published website within the organization. The server portion of the tool consists of an Information retrieval system that creates indexes on the code repository and enables querying on indexes.

The paper “Exemplar: EXEcutable exaMPLes ARchive” by Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., and Cumby, C. [27] showed a novel code search engine called Exemplar (EXEcutable exaMPLes ARchive) to bridge the mismatch between the high-level intent and low-level implementation details of applications while searching for applications that are highly relevant to development tasks. Exemplar combines information retrieval and program analysis techniques to reliably link high-level concepts to the source code of the applications via standard third-party Application Programming Interface (API) calls that these applications use. The paper “Finding Relevant Applications for Prototyping” by Grechanik, M., Conroy, K. M., and Probst, K. A.” [28] proposed the approach Exemplar (EXEcutabl exaMPLes ARchive) for finding highly relevant software projects from a large archive of executable applications. After a programmer enters a query that contains high-level concepts (e.g., toolbar, download, smart card), Exemplar combines these concepts with information retrieval and program analysis to retrieve projects that implement high-level concepts entered as part of this initial query. The next paper "Recommending Source Code Examples via API Call Usages and Documentation” proposed a new method to recommend source code examples to developers by querying against Application Programming Interface (API) calls and their documentations that are fused with structural information about the code. The researchers' work leverages information from the source code’s list of API calls and those calls’ usage documentation. They used text based Information Retrieval (IR) to associate user queries with documentation of API calls. The paper “Portfolio: Finding Relevant Functions and Their Usages” by McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., and Fu, C. [29] talked about the approach Portfolio used for finding highly relevant functions and projects from a large archive of C/C++ source code. In Portfolio, the researchers combined various natural language processing (NLP) and indexing techniques

with a variation of PageRank and spreading activation network (SAN) algorithms to address the need of programmers to reuse retrieved code as functional abstractions.

The next paper “How Well Do Search Engines Support Code Retrieval on the Web?” by Sim, S. E.; Umarji, M.; Ratanotayanon, S. & Lopes, C. [30] talked about the in-depth study of the phenomenon of code retrieval and its implications for the development of code search tools. The study had two main contributions: the methodology employed to create the scenarios that were given to participants in the study and the empirical results. In the study, the authors obtained three statistically significant results. It was easier to find reference examples than components for as-is reuse. A general-purpose search engine was the most effective overall on all tasks. However, code-specific search engines were more effective when searching for subsystems, such as libraries. More informally, if you were only allowed to use one search engine, you should choose a general-purpose one, such as Google. If you could pick and choose depending on task, you should use a code-specific search engine to find subsystems and a general-purpose search engine for all other searches.

As known, reuse recommendation systems support the developer by suggesting useful API methods, classes or code snippets based on code edited in the IDE. Existing systems based on structural information, such as type and method usage, are not effective in case of general purpose types such as String. To alleviate this, the paper “Identifier-Based Context-Dependent API Method Recommendation” by Heinemann, L., Bauer, V., Herrmannsdoerfer, M. & Hummel, B. [31] proposed a recommendation system based on identifiers that utilizes the developer’s intention embodied in names of variables, types and methods.

We saw in the above papers how crucial software reuse is and how some of the great technologies and tools by various researchers can help to increase the efficiency and productivity. Also, software reuse helps in lowering the cost, developing the software faster and lowers the risk involved.

6.4 Source Code Pre-processing

6.5 Bug Triage

Bug reports are essential software artifacts that describe software bugs, especially in open-source software. Lately, due to the availability of a large number of bug reports, a considerable amount of research has been carried out on bug-report analysis, such as automatically checking duplication of bug reports and localizing bugs based on bug reports. To review the work on bug-report analysis, this paper presents an exhaustive survey on the existing work on bug-report analysis.

As programmers can hardly write programs without any bugs, it is inevitable to find bugs and fix bugs in software development. Moreover, it is costly and time-consuming to find and fix bugs in software development. Software testing and debugging is estimated to consume more than one third of the total cost of software development. To guarantee the quality of software efficiently, many projects use bug reports to collect and record the bugs reported by developers, testers, and end-users. Actually, with the rapid development of software, vast amounts of bug reports have been produced. Although a large number of bug reports may help improve the quality of software, it is also a challenge to analyze these bug reports. To address this practical problem, many researchers proposed various techniques to facilitate

bug-report analysis. Work on bug triage aims to triage bug reports automatically, and includes bug-report prioritization, duplicate bug-report detection, and bug-report assignment.

A bug report goes through several resolution status over its lifetime. When a bug is first reported, the bug report is marked as UNCONFIRMED. When a triager has verified that the bug is not duplicate and indeed a new bug, the status is set to NEW. Then the triager assigns the bug report to one proper developer, and the status is changed to ASSIGNED. Then the assigned developer reproduces the bug, localizes it and tries to fix it. When the bug has been solved, the bug report is marked as RESOLVED. After that, if a tester is not satisfied with the solution, the bug should be reopened with the status set to REOPEN; if a tester has verified that the solution worked, the status is changed to VERIFIED. The final status of a bug report is CLOSED, which is set when no occurrence of the bug is reported. Also, there are more than one possible sequent status for the status RESOLVED. If a developer makes necessary code changes and the bug is solved, the status will be changed from RESOLVED to FIXED. If the developer does not fix the bug due to some reason, the status will be set to WONTFIX. If the problem could not be reproduced by the developer, the status will be changed to WORKSFORME. If the developer finds that the report is a duplicate of an existing bug report, the status will be changed to DUPLICATE. If the bug reported by the bug report is not an actual bug, the status will be changed to INVALID.

Bug-report triage is a process of checking bugs, prioritizing bugs, and assigning bugs to proper developers (i.e., bug assignment). In practice, most bug reports are triaged manually to developers. However, for a large open-source project, every day an ignorable number of bug reports are submitted. It is a labor-intensive task for developers to read, analyze, prioritize these bug reports, check duplication of these bug reports, and assign these bug reports to a proper developer from hundreds of candidates. Moreover, these procedures are also error-prone. To assist triagers who are responsible for the process of bug-report triage and improve the bug triaging efficiency, many researchers have focused on automating bug-report triage, including prioritizing bug reports, checking duplication of bug reports and assigning bug reports to developers.

As a large number of bug reports exist in the bug tracking system for a large project, it is time-consuming and effort-consuming to deal with these bug reports. Some bug reports are more important than others. Due to time limit, it is necessary to deal with important bug reports early. That is, triagers assign priority to bug reports and prioritize these bug reports based on their priority.

To facilitate bug-report prioritization, the authors of the paper “Bug prioritization to facilitate bug report triage” proposed a bug priority recommender based on SVM and Naive Bayes classifiers. Besides, they compared the results of these two classifiers in terms of accuracy, and found that when using text features for recommending bug priority, SVM performs better, while when using categorical features, Naive Bayers performs better.

The authors of the paper “A discriminative model approach for accurate duplicate bug report retrieval” proposed a novel approach based on discriminative approach that assigns different weights to words in bug reports when calculating textual similarity. Their empirical studies showed that their approach outperforms all former techniques that used merely textual information. Later, they further improved their approach in the paper “Towards more accurate retrieval of duplicate bug reports” by bringing in a textual similarity metric called BM25F. After that, also based on BM25F, authors of the paper “Improved duplicate bug report identification” proposed to consider the similarity between new bug reports and multiple existing bug reports (instead of just the most similar bug reports) to decide whether the bug reported by the new bug report is actually a duplicate bug.

In the paper “Duplicate bug report detection with a combination of information retrieval and topic modeling”, the authors proposed to apply topic modelling to detect duplicate bug reports, which measures the semantic similarity between words.

In the paper “Detecting Duplicate Bug Reports with Software Engineering Domain Knowledge”, the authors proposed a method to partially automate the extraction of contextual word lists from software-engineering literature. Evaluating this software-literature context method on real-world bug reports produced useful results that indicated that this semi-automated method had the potential to substantially decrease the manual effort used in contextual bug deduplication while suffering only a minor loss in accuracy.

In the paper “A comparative study of the performance of IR models on duplicate bug detection”, the authors compared the performance of the traditional vector space model (using different weighting schemes) with that of topic based models, leveraging heuristics that incorporate exception stack frames, surface features, summary and long description from the free-form text in the bug report.

In the paper “Identifying Linux Bug Fixing Patches”, the authors proposed an approach that automatically identified bug fixing patches based on the changes and commit messages recorded in code repositories. They compared their approach with the keyword-based approach for identifying bug-fixing patches used in the literature, in the context of the Linux kernel. The results showed that their approach could achieve a 53.19% improvement in recall as compared to keyword-based approaches, with similar precision.

To facilitate bug report digestion, the authors of the paper “Modelling the ‘Hurried’ Bug Report Reading Process to Summarize Bug Reports” proposed an unsupervised, bug report summarization approach that estimated the attention a user would hypothetically give to different sentences in a bug report, when pressed with time. They pose three hypotheses on what makes a sentence relevant: discussing frequently discussed topics, being evaluated or assessed by other sentences, and keeping focused on the bug report’s title and description. The results suggest that their hypotheses were valid, since the summaries have as much as 12% improvement in standard summarization evaluation metrics compared to the previous approach. The evaluation also asked the developers to assess the quality and usefulness of the summaries created for bug reports they have worked on. Feedback from developers not only showed the summaries were useful, but also pointed out important requirements for this, and any bug summarization approach, and indicated directions for future work.

There is a tremendous wealth of code authorship information available in source code. Motivated with the presence of this information, in a number of open source projects, an approach to recommend expert developers to assist with a software change request (e.g., a bug fixes or feature) was presented in the paper “Triaging Incoming Change Requests: Bug or Commit History, or Code Authorship?”. It employed a combination of an information retrieval technique and processing of the source code authorship information. The relevant source code files to the textual description of a change request were first located. The authors listed in the header comments in these files were then analyzed to arrive at a ranked list of the most suitable developers. The approach fundamentally differs from its previously reported counterparts, as it did not require software repository mining. Neither did it require training from past bugs/issues, which is often done with sophisticated techniques such as machine learning, nor does mining of source code repositories, i.e., commit.

Text retrieval (TR) techniques have been widely used to support concept and bug location. When locating bugs, developers often formulate queries based on the bug descriptions. More than that, a large body of

research uses bug descriptions to evaluate bug location techniques using TR. The implicit assumption is that the bug descriptions and the relevant source code files share important words. In the paper "On the Relationship between the Vocabulary of Bug Reports and Source Code", the authors presented an empirical study that explored this conjecture. They found that bug reports shared more terms with the patched classes than with the other classes in the system. Furthermore, they found that the class names were more likely to share terms with the bug descriptions than other code locations, while more verbose parts of the code (e.g., comments) would share more words. They also found that the shared terms may be better predictors for bug location than some TR techniques.

In the paper "Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-Feature of Bug Reports", the authors proposed a novel method for the bug triage and bug severity prediction. First, they extracted topic(s) from historical bug reports in the bug repository and found bug reports related to each topic. When a new bug report arrives, they decided the topic(s) to which the report belonged. Then they utilized multi-feature to identify corresponding reports that had the same multi-feature (e.g., component, product, priority and severity) with the new bug report. Thus, given a new bug report, they were able to recommend the most appropriate developer to fix each bug and predict its severity. To evaluate this approach, they not only measured the effectiveness of the study by using about 30,000 golden bug reports extracted from three open source projects, but also compared some related studies. The results showed that this approach was likely to effectively recommend the appropriate developer to fix the given bug and predict its severity.

In the paper "AUSUM: Approach for Unsupervised Bug Report SUMmarization", the authors presented the results of applying four unsupervised summarization techniques for bug summarization. Industrial bug reports typically contain a large amount of noise—email dump, chat transcripts, core-dump—useless sentences from the perspective of summarization. These derail the unsupervised approaches, which are optimized to work on more well-formed documents. They presented an approach for noise reduction, which helps to improve the precision of summarization over the base technique (4% to 24% across subjects and base techniques). Importantly, by applying noise reduction, two of the unsupervised techniques became scalable for large sized bug reports.

We observed that most of the papers dealt with using Information Retrieval techniques for bug triage but with the more recent papers, we observed that researchers have focused on automatic documentation of bug reports for triaging. However, there has not been very satisfactory results in bug triaging when using Natural Language Processing instead of Information Retrieval techniques.

6.6 TRACEABILITY

Traceability is the process of identifying links from system documentation to the source code of a software. There might be a wide variety of software artefacts in the process of developing the software such as source code, design specifications, test cases, etc. These are also documented. Maintaining the traceability links between the artefacts and their documentation is an important task. After going through the papers dealing with traceability, we observed that the papers used either Natural Language Processing as a technique and Information Retrieval techniques. The majority of the papers dealt with Information Retrieval Techniques using Vector Space Retrieval and Latent Semantic Indexing. One of the papers dealt with using NLP in combination with IR.

Sundaram, Hayes, Dekhtyar and Holbroo [32] in their paper use three different methods – Vector Space Retrieval, Vector Space Retrieval with a Thesaurus and Latent Semantic Indexing to study traceability. They used datasets from MODIS, CM-1 and Waterloo. They implemented the Information Retrieval Techniques as mentioned above as a part of their Requirements Tracing Tool called RETRO. IR methods were applied with relevance feedback to the problem of tracing textual artifacts and demonstrated that, in certain cases, secondary measures significantly affect the analyst's perception of the quality of the results. Numerous examples were found where the assessment of the results of a trace given primary measures was very different from the result assessment using secondary measures. In practice, however, it will be up to human analysts to supply relevance feedback, and as such, it is impossible to envision analysts to be 100% correct in their decisions.

Panichella, De Lucia and Zaidman [33] in their paper proposed an adaptive version of relevance feedback that considers both the software artefact and previously classified links for deciding how to apply feedback. The Relevance feedback was initially suggested by Hayes et al and Panichella et al improved on the technique. They compared their research with a pure IR Vector Space Model technique. They used three different software projects for observing results. They noticed that relevance feedback does not always improve the accuracy of IR methods, such as VSM, when applied to software artefacts. The adaptive feedback yielded strong, statistically significant improvements with respect to the standard feedback, which provides benefits only for requirement tracing.

Alhindawi, Meqdadi, Bartman and Maletic [34] in their paper used a Natural Language Processing technique to solve the traceability problem. They used an IR technique, Latent Semantic Indexing to identify the traceability links from system documentation to source code. They proposed Tracelab – a tool for conducting rapid experiments in traceability uncovering research. They used NLP by creating a sequential pattern mining tool called SPADE (Sequential Pattern Discovery Algorithm). They found that the results obtained by running Tracelab over KDE office was promising to demonstrate Tracelab as an environment to uncover traceability links.

Similarly, Kamalabalan et Al [35] created a tool to establish artefact traceability links and visualization of the links. This research involved combining IR techniques with NLP. They named the tool SATAnalyzer (Software Artefact traceability Analyzer). After the artefact extraction process, relationships among artefacts are built using semi-automated process. Learning algorithms are able to extract specific patterns involved in a specific environment such as naming differences between design and source artefacts due to coding conventions. The results obtained from the tool showed that the tool performed well in extracting classes, attributes and behaviours from requirement artefacts and establishing traceability links.

We observed that the Information Retrieval techniques were found to give better results in terms of accuracy in comparison with NLP techniques.

7 CONCLUSION

Thus we found how Information Retrieval techniques and Natural Language Processing techniques are used in different software engineering tasks. We also observed how each task is performed and how variations of the techniques are applied for each task. One point that was mentioned while researching

for the papers was that Natural Language Processing is relatively new to being used in software engineering tasks and that it has been used only after 2013. It is still a developing field and has its own set of issues which varies from task to task. Information retrieval though, has been mentioned as being more reliable because it has been in use for a very long time. We would also like to thank Professor Andrian Marcus for providing us with the resources for getting started with this survey. Finally, we found this survey to be a very knowledgeable and enriching experience.

8 REFERENCES

[1] Revelle, Meghan, Bogdan Dit, and Denys Poshyvanyk. "Using data fusion and web mining to support feature location in software." Program Comprehension (ICPC), 2010 IEEE 18th International Conference on. IEEE, 2010.

[2] McBurney, Paul W., and Collin McMillan. "Automatic documentation generation via source code summarization of method context." Proceedings of the 22nd International Conference on Program Comprehension. ACM, 2014.

[3] Haiduc, Sonia, et al. "On the use of automated text summarization techniques for summarizing source code." Reverse Engineering (WCRE), 2010 17th Working Conference on. IEEE, 2010.

[4] Buse, Raymond PL, and Westley R. Weimer. "Automatically documenting program changes." Proceedings of the IEEE/ACM international conference on automated software engineering. ACM, 2010.

[5] Medini, Soumaya, et al. "Scan: an approach to label and relate execution trace segments." Journal of Software: Evolution and Process 26.11 (2014): 962-995.

[6] Rodeghero, Paige, et al. "Improving automated source code summarization via an eye-tracking study of programmers." Proceedings of the 36th International Conference on Software Engineering. ACM, 2014.

[7] Vassallo, Carmine, et al. "CODES: mining sourCe cOde Descriptions from developErs diScussions." Proceedings of the 22nd International Conference on Program Comprehension. ACM, 2014.

[8] P. C. Rigby and M. P. Robillard. Discovering essential code elements in informal documentation. In 32nd ACM/IEEE International Conference on Software Engineering, pages 832–841, 2013

[9] Moreno, Laura, et al. "Jsummarizer: An automatic generator of natural language summaries for java classes." Program Comprehension (ICPC), 2013 IEEE 21st International Conference on. IEEE, 2013.

[10] Li, Boyang, et al. "Automatically Documenting Unit Test Cases."

[11] Wong, Edmund, Taiyue Liu, and Lin Tan. "CloCom: Mining existing source code for automatic comment generation." Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on. IEEE, 2015.

[12] Sridhara, Giriprasad, Lori Pollock, and K. Vijay-Shanker. "Generating parameter comments and integrating with method summaries." Program Comprehension (ICPC), 2011 IEEE 19th International Conference on. IEEE, 2011.

[13] Marcus, Andrian, et al. "An information retrieval approach to concept location in source code." *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. IEEE, 2004.

[14] Rubin, Julia, and Marsha Chechik. "A survey of feature location techniques."*Domain Engineering*. Springer Berlin Heidelberg, 2013. 29-58.

[15] Lukins, Stacy K., Nicholas A. Kraft, and Letha H. Etzkorn. "Source code retrieval for bug localization using latent dirichlet allocation." *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on*. IEEE, 2008.

[16] Thung, Ferdian, et al. "BugLocalizer: integrated tool support for bug localization." Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014.

[17] Tian, Yuan, et al. "Automated prediction of bug report priority using multi-factor analysis." Empirical Software Engineering 20.5 (2015): 1354-1383.

[18] Revelle, Meghan, Bogdan Dit, and Denys Poshyvanyk. "Using data fusion and web mining to support feature location in software." Program Comprehension (ICPC), 2010 IEEE 18th International Conference on. IEEE, 2010.

[19] Hill, Emily, Lori Pollock, and K. Vijay-Shanker. "Improving source code search with natural language phrasal representations of method signatures." Proceedings of the 2011 26th IEEE/ACM International Conference on

[20] Martie, Lee, and André van der Hoek. "Sameness: an experiment in code search." Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on. IEEE, 2015.

[21] Maarek, Y. S., Berry, D. M., and Kaiser, G. E., (1991), "An Information Retrieval Approach for Automatically Constructing Software Libraries", IEEE Transactions On Software Engineering, vol. 17, no. 8, pp. 800-813.

[22]Maarek, Y. S. and Smadja, F. A., (1989), "Full Text Indexing Based on Lexical Relations, an Application: Software Libraries", in Proceedings of SIGIR89, pp. 198-206.

[23] Michail, A. and Notkin, D., (1999), "Assessing software libraries by browsing similar classes, functions and relationships", in Proceedings of IEEE International Conference on Software Engineering (ICSE'99), pp. 463-472.

[24] H. Mili, E. Ah-ki, R. Godin, and H. Mcheick. An experiment in software component retrieval. Information and Software Technology, 45, 2003, 45, pp. 633-649

[25] Stierna, E. and Rowe, N., (2003), "Applying information-retrieval methods to software reuse: a case study", International Journal on Information Processing and Management:, vol. 39, no. 1, pp. 67-74.

[26] Sindhgatta, R., (2006), "Using an information retrieval system to retrieve source code samples", in Proceedings of 28th IEEE/ACM International Conference on Software Engineering (ICSE'06), pp. 905-908.

[28] Grechanik, M., Conroy, K. M., and Probst, K. A., (2007), "Finding Relevant Applications for Prototyping", in Proceedings of 4th IEEE International Workshop on Mining Software Repositories (MSR'07), pp. 12-15.

[27] Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., and Cumby, C., "Exemplar: EXEcutable exaMPLes ARchive", in Proc. of 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), Formal Research Tool Demonstration, Cape Town, South Africa, May 2-8, 2010, pp. 259-262

[29] McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., and Fu, C., "Portfolio: Finding Relevant Functions and Their Usages", in Proc. of 33rd IEEE/ACM International Conference on Software Engineering (ICSE'11), Honolulu, HI, USA, May 21-28 2011, pp. 111-120

[30] Sim, S. E.; Umarji, M.; Ratanotayanon, S. & Lopes, C. "How Well Do Search Engines Support Code Retrieval on the Web?" in ACM Transactions on Software Engineering and Methodology (TOSEM), 2012, pp. 21

[31] Heinemann, L.; Bauer, V.; Herrmannsdoerfer, M. & Hummel, B. "Identifier-Based Context-Dependent API Method Recommendation" in 16th European Conference on Software Maintenance and Reengineering (CSMR'12), 2012, pp. 31-40

[32] Sundaram, Senthil Karthikeyan, et al. "Assessing traceability of software engineering artifacts." Requirements engineering 15.3 (2010): 313-335.

[33] Panichella, Annibale, Andrea De Lucia, and Andy Zaidman. "Adaptive user feedback for IR-based traceability recovery." Software and Systems Traceability (SST), 2015 IEEE/ACM 8th International Symposium on. IEEE, 2015.

[34] Alhindawi, Nouh, et al. "A tracelab-based solution for identifying traceability links using LSI." Traceability in Emerging Forms of Software Engineering (TEFSE), 2013 International Workshop on. IEEE, 2013.

[35] Kamalabalan, K., et al. "Tool support for traceability of software artefacts."Moratuwa Engineering Research Conference (MERCon), 2015. IEEE, 2015.

Bajracharya, S.; Ossher, J. & Lopes, C. "Sourcerer: An Internet-scale Software Repository" in Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation, IEEE Computer Society, 2009, pp. 1-4

Zheng, W.; Zhang, Q. & Lyu, M. "Cross-Library API Recommendation using Web Search Engines" in Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2011, pp. 480-483

Zhou, Jian, Hongyu Zhang, and David Lo. "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports." *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012.

Ye, Y. and Fischer, G., (2005), "Reuse-Conducive Development Environments", Journal Automated Software Engineering, vol. 12, no. 2, pp. 199-235.

McMillan, C., Poshyvanyk, D., and Grechanik, M., "Recommending Source Code Examples via API Call Usages and Documentation", in Proc. of 2nd ICSE2010 International Workshop on Recommendation Systems for Software Engineering (RSSE'10), Cape Town, South Africa, May 4, 2010

Sun, C.; Lo, D.; Wang, X.; Jiang, J. & Khoo, S.-C. "A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval" in Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1, ACM, 2010, pp. 45-54

Sun, C.; Lo, D.; Khoo, S.-C. & Jiang, J. "Towards More Accurate Retrieval of Duplicate Bug Reports" in 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11), 2011, pp. 253-262

Tian, Y.; Sun, C. & Lo, D. "Improved Duplicate Bug Report Identification" in 16th European Conference on Software Maintenance and Reengineering (CSMR'12), 2012, pp. 385-390

Nguyen, A. T.; Nguyen, T.; Nguyen, T.; Lo, D. & Sun, C. "Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling" in 27th IEEE/ACM International Conference on Automated Software Engineering (ASE'12), 2012, pp. 70-79

Karan Aggarwal, Tanner Rutgers, Finbarr Timbers, Abram Hindle, Russ Greiner, and Eleni Stroulia, "Detecting Duplicate Bug Reports with Software Engineering Domain Knowledge" 29th IEEE/ACM International Conference on Software Engineering (ICSE'07), 2007, pp. 499-510

Kaushik, N. & Tahvildari, L. "Comparative Study of the Performance of IR Models on Duplicate Bug Detection" in 16th European Conference on Software Maintenance and Reengineering (CSMR'12), 2012, pp. 159-168

Tian, Y.; Lawall, J. & Lo, D. "Identifying Linux Bug Fixing Patches" in 34th International Conference on Software Engineering (ICSE'12), 2012, pp. 386-396

Lotufo, R.; Malik, Z. & Czarnecki, K. Modelling the `Hurried' Bug Report Reading Process to Summarize Bug Reports" in 28th IEEE International Conference on Software Maintenance (ICSM'12), 2012, 20, pp. 516-548

Linares-Vasquez, M.; Hossen, K.; Dang, H.; Kagdi, H.; Gethers, M. & Poshyvanyk, D. "Triaging Incoming Change Requests: Bug or Commit History, or Code Authorship?" in 28th IEEE International Conference on Software Maintenance (ICSM'12), 2012, pp. 451-460

L. Moreno, W. Bandara, S. Haiduc, and A. Marcus. On the relationship between the vocabulary of bug reports and source code. In Proceedings of the International Conference on Software Maintenance (ICSM), pages 452–455, 2013.

G. Yang, T. Zhang, and B. Lee. "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports" In IEEE Annual Computer Software and Applications Conference (COMPSAC), pages 97–106, 2014.

Senthil Mani, Rose Catherine, Vibha Singhal Sinha, Avinava Dubey, "AUSUM: approach for unsupervised bug report summarization" in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ACM, 2012, pp. 11:1-11:11

Kanwal J, Maqbool O. Bug prioritization to facilitate bug report triage. J Comput Sci Technol, 2012, 27: 397–412

Zhang J, Wang X Y, Hao D, et al. A survey on bug-report analysis. Sci China Inf Sci, 2015, 58:021101(24), doi: 10.1007/s11432-014-5241-2