SUMMARIZATION OF SOFTWARE ARTIFACTS: A REVIEW

Som Gupta[1] and S.K Gupta[2]

[1]Research Scholar, AKTU Lucknow, UP, India,
[2]Computer Science Department, BIET Jhansi, UP, India

ABSTRACT

Summarization of software artifacts is an ongoing field of research among the software engineering community due to the benefits that summarization provides like saving of time and efforts in various software engineering tasks like code search, duplicate bug reports detection, traceability link recovery, etc. Summarization is to produce short and concise summaries. The paper presents the review of the state of the art of summarization techniques in software engineering context. The paper gives a brief overview to the software artifacts which are mostly used for summarization or have benefits from summarization. The paper briefly describes the general process of summarization. The paper reviews the papers published from 2010 to June 2017 and classifies the works into extractive and abstractive summarization. The paper also reviews the evaluation techniques used for summarizing software artifacts. The paper discusses the open problems and challenges in this field of research. The paper also discusses the future scopes in this area for new researchers.

KEYWORDS

summarization; software artifacts; mining software repositories; extractive summarization; abstractive summarization;

1. INTRODUCTION

Summarization is to reduce the content of a document in such a manner that the important information about the document is preserved. Summarization techniques are broadly classified into extractive and abstractive techniques. Extractive summarization is to extract the important sentences from a document in same way as they appear in the original document and arrange them to create a summary of specific length. Whereas abstractive summarization is to understand the text and apply linguistic rules to create a summary. Natural language techniques are majorly used for abstractive summaries.

There are various software artifacts which are created during a software life cycle along with a working source code like requirements document, design documents, bug reports, etc. These artifacts are usually archived for future use for understanding of system during software maintenance or evolution phase. Reading and understanding of a document is a time-consuming task [1] but is must to support any software development task. For example, when a change is to be implemented to the system, proper understanding of system is essential as there exists many dependencies in a system and failure to handle them may result in bugs or system failure. Summarization of software artifacts help save the developer's efforts and time while performing

software engineering tasks. It has been observed that most of the times, manually written summaries are incomplete, too-short or outdated. Summarization helps creating the documents automatically and thus relieving the programmers from the tedious task of documentation [2]. Source code and bug reports are two major artifacts on which most of the summarization techniques have been applied to generate the summaries. Source code summaries help know the description of sections of code and know how a piece of code is related to other parts of source code. Bug reports not just contain the defects but also the reasons behind the bugs, feature enhancement ideas, steps for resolution of bugs, etc. [4]. Bug reports summarization helps user read, investigate and understand multiple aspects of a defect and helps perform many tasks like bug reports duplication, etc.Low time to market and tough competition have raised the need of summarization tools to help developers. Summarization is a very complex task. Till now it has not been very clear on what exactly should go to the summary and what should not be included in the summary [3]. Evaluation of summaries is also a challenging task as it is difficult to find the accurate effectiveness from human evaluation.

The structure of paper is as follows. Section 2 briefly gives an overview to a general summarization procedure. Section 3 gives a brief overview of the methodology used for presenting the various works in the field of software summarization. Section 4 gives a brief introduction to the extractive summarization techniques and works performed for generating extractive summaries for software artifacts. Section 5 gives a brief introduction to the abstractive summarization techniques and various works performed for generating abstractive summaries for software artifacts. Section 6 discusses the evaluation techniques used for summarizing software artifacts. Section 7 discusses the application of software summarization. Section 8 lists down the open problems and challenges in the field of summarization of software artifacts. Section 9 discusses the areas where there is a future scope for research. And finally, the conclusion.

2. SUMMARIZATION PROCESS: AN OVERVIEW

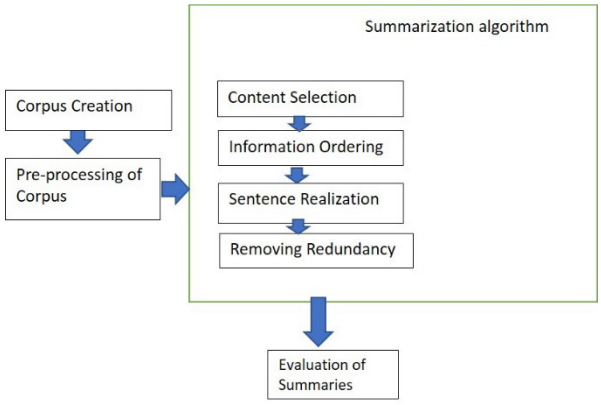

Fig 1: General Summarization Process

A typical summarization process follows these steps:

1. Corpus Creation: It is to collect or extract the documents of desired granularity as per the requirements to perform summarization [5].

2. Pre-processing of corpus: It mainly includes identification of sentence boundaries, tokenization, stops words removal, stemming, case folding and noise removal. There are

various open source NLP toolsavailable for pre-processing of corpus like python NLTK, Apache OpenNLP, Stanford CoreNLP, etc.

3. Summarization Algorithm Application:

3.1. Content Selection: Many Algorithms and approaches like natural language processing, machine learning based approaches, information retrieval based approaches are applied to select the important sentences from the artifacts to generate the summary. One of the famous algorithm used for content selection is:

PageRank- It is a graph based algorithm used for web links for determining the ranking of web pages. But it can be used in the software engineering context as well. In context of source code summarization, node can represent important functions or methods in the program and edge a relation between the methods. TextRank is an algorithm for summarization of text based on PageRank algorithm. By applying this algorithm important functions or methods to be included in the summary can be obtained.

3.2. Information Ordering: After the content to be added are extracted, it is important to score the sentences. There are various sentence scoring methods available like word-based scoring, sentence-based scoring and graph based scoring methods. In word-based scoring method, each word is assigned some score and to compute the total score of sentence, all the word scores are summed up. For sentence-based scoring methods, features are analyzed using cue-phrases (sentences started by “in summary”, “in conclusion”, “our survey”, etc.), position of sentence, resemblance of sentence to title and many other features like these. For graph-based scoring method, TextRank, bushy path of node i.e. no of edges connecting a node to other node and aggregate summary by counting the number of edge connecting a node to other nodes in the bushy path are used [6].

3.3. Sentence Realization: It is to simplify the sentences. In extractive summarization, the selected sentences are kept in the summary as in original document but in case of abstractive summarization, natural language summaries are produced and for them simplifying the sentences to reduce the length is required.

3.4. Removing Redundancy: It is to remove the redundant sentences. Redundancy is when multiple sentences have same content. Sentence fusion and textual entailment are two widely used techniques to remove the redundancy among abstractive summaries but they can be adapted for extractive summaries as well. Maximal Marginal Relevance is also used for removing the redundancy among the sentences.

4. Evaluation of Summaries: Last step of summarization process is to evaluate the summaries so generated. There are two ways to evaluate the summaries: intrinsic and extrinsic evaluation. Intrinsic evaluation is when the system is evaluated with itself and extrinsic evaluation is when the system is evaluated in terms of how the completion of other tasks is affected. Intrinsic evaluation mostly focuses on the informativeness and coherence whereas extrinsic evaluation mostly focuses on the efficiency and acceptability [7].

3. METHODOLOGY

We have used IEEE Xplore for finding the relevant research papers. We have summarized the articles according to the technique used by them for summarization and type of evaluation. We have presented them into four columns namely name of authors, artifact on which they performed summarization, technique used in their work and the corpus on which they performed their work. We have classified the extractive summarization works into Information-Retrieval Based, Machine Learning Based, and Topic Models. We have also listed the extractive summarization works which have used Crowdsourcing approach and eye-interaction based approach.

For abstractive summarization works, we have classified them into Structure Based Approach and Semantic Based Approach.

We have also classified the works according to the evaluation techniques employed for evaluating their summaries into intrinsic and extrinsic evaluation. We have identified the various techniques used in the field of software summarization for evaluation.

From the various research papers obtained from IEEE Xplore, we have identified the applications of software summarization and classified them into Bug Reports Digestion, Improving Traceability link recovery, program comprehension, automatic documentation generation. We have identified the future scope areas in the field of software summarization and classified them into unit test case summarization, duplicate bug reports detection, source code summarization, summarization using crowdsourcing, summarization using eye-tracking interactions, improving sentence Ranking Techniques, Creating personalized summaries and Visualization.

4. EXTRACTIVE SUMMARIZATION: AN OVERVIEW IN THE CONTEXT OF SOFTWARE ARTIFACTS

Extractive summarization is to generate summaries by extracting sentences from the original text. They use statistical analysis of features to locate the important sentences from the text. Various features which have been used for sentence extraction like title word feature, keyword features, sentence length features, proper noun features, upper case features, cue-phrase features, biased word features, font based features, pronouns, presence of non-essential features, sentence-to-sentence cohesion and discourse analysis [8]. There are various approaches which have been used for generating extractive summaries like IR-based approaches, Natural Language Processing, Machine Learning based approaches, Topic Models, etc. We have analyzed the papers published in recent previous years and classified them as per the technique they use:

- Information Retrieval Based:

  - Vector Space Model(VSM): It is based on term-frequency matrix for a document and is used for modifying the weights of indexing terms which helps in finding the relevance of sentences or documents. Documents are represented as vectors. Dimensionality of vector is basically the size of vocabulary. Cosine similarity is mainly used for calculating the similarity between the document and the query.

- Latent Semantic Indexing(LSI): It is mainly used for the document indexing and retrieval. Term-Document matrix is constructed and then Singular Value Decomposition(SVD) is applied to reduce the matrix.
- Latent Dirichlet Allocation(LDA): It is a generative probabilistic topic model. Each document is treated as a group of topics where each topic is assigned some probability of generating some words.

Antoniol et al. [9] have used probabilistic IR model and VSM to recover the traceability links for C++ projects between the source code and free text documents. Sonia Haiduc et al. [10] used both the extractive and lightweight abstractive summarization techniques to automatically generate the source code summaries. For extractive summarization, they used VSM and LSI using log, tf-idf and binary entropy as weighting mechanisms. Sonia Haiduc et al. [11] in their another paper used extractive summarization using lexical and structural information about source code to generate the summaries. They used LSI and cosine similarity to obtain the information for their summaries.

- Machine Learning Based Approach: It is one of the widely used approach for performing extractive summarization.
- Supervised Learning Approach: Here the likelihood of label is suggested by using labeled training dataset [12]. Various classifiers like SVM, Naïve-Bayes, Decision trees, etc. are used for training to extract the useful features to be included in the summary when we have documents and their respective reference summaries.
  Rastkar et al. [13] applied extractive summarization techniques for automatically generating the summaries for bug reports using supervised machine learning. They used binary classifiers for producing the summaries. They trained the classifiers on e-mail threads, combination of e-mail threads and meetings, and meetings and bug reports based on 24 features classified into 4 categories namely structural features, participant features, length features and lexical features. Rastkar et al. [14] in another paper used extractive multi-document summarization by identifying the sentence level features to describe the code changes.

Rigby et al. [15] have used three feature decision tree classifiers to extract the code elements for StackOverflow Site Discussions. Nazar et al. [16] used SVM and Naïve Bayesian classifiers to create code fragment summaries from online FAQs. They extracted the features using data-driven crowdsourcing mechanism. Ying et al. [17] used supervised machine learning based approach using Naïve Bayes and SVM classifiers by considering syntactic and query related features for generating code fragment summaries.

- Unsupervised Learning Approach: These techniques try to find out the labels from the data. These techniques assign centrality and diversity measures to the sentences for deciding whether to include it to the summary or not. K-means clustering is widely used to select the features to be included in the summary [18]. Euclidean distance is used to measure the similarity between two sentences. There are various unsupervised learning approaches which have been used for summarizing software artifacts. Mani et al. [19] have used centroid, Maximum Marginal Relevance, Grasshopper and DiverseRank unsupervised techniques for summarizing bug reports. Lotufo et al. [4] used unsupervised bug reports summarization to generate the summaries and used PageRank to calculate the probabilities. They used number of topics shared between the sentences, number of times

sentence evaluated by other sentences and number of topics shared with title and description of bug report to ranks the sentences from relevance perspective. Ferreira et al. [20] used Cosine similarity, Euclidean distance, Louvain community detection and PageRank methods to rank the comments in bugs to generate the extractive summaries.

- Semi Supervised Learning Approach: Andrea Di Sorbo [21] proposed a semi-supervised learning approach called DECA (Development Emails Content Analyzer) where they classified the emails according to their content into various categories like opinion asking, information seeking, information giving, feature request, etc. by using natural language parsing. They also used this approach to re-document the source code.

- Topic Models: It is a statistical model where each document is represented as a set of topics. Each topic is a set of words which mostly occur together. Each topic is given some probability based on frequencies and co-

Occurrence frequencies. Topic assignments for each token is calculated by using topic model. Topic models follows Bayesian paradigm. HPAM (Hierarchical Pachinko Allocation Model) is one technique for the topic models.

HPAM: Directed Acyclic Graphs(DAG) is used for representing the topics. Bag-of-words representation is used and each document is represented as vector with V components where V is the size of vocabulary. Few researches are found which have used NLP based techniques to create the extractive summaries. Paul W. McBurney et al. [22] have used Topic Models to create the source code summaries focusing on the presentation of summaries. They used HDTM algorithm extract the hierarchy of topics from the source code. Brian P. Eddy et al. [23] proposed a topic model based approach to summarize the source code and then compared their results with the approaches used by Sonia Haiduc et al. [10]. Fowkes et al. [24] have created a tree based algorithm called TASSAL (Tree Based AutoFolding Software Summarization Algorithm) which works on Abstract Syntax Tree of source code and uses Topic Models for autofolding of source code which is to fold the less informative code areas by focusing on the file specific tokens. They used TopicSum model to source code where the common Java tokens, common javaDoc comment tokens and common header comment tokens were taken as input to construct the model and as output it categorized the token to whether specific to file, project or general Java code.

Lot of researches are done where eye-tracking interactions are used to improve the selection of subset of keywords for summaries. Rodeghero et al. [25] have generated the extractive summaries of java methods by finding out the important keywords based on eye tracking study of 10 java professionals where they were asked to read and write the summaries especially for those methods which were uncommented by focusing on eye-movements, gaze fixations and regressions.

In addition to IR-based approaches, Machine Learning Based approaches, Natural Language Processing Approaches, Eye-tracking interactions, crowdsourcing is also one of the emerging approaches for helping generate the summaries. Crowdsourcing is a web 2.0 based phenomenon which is a data driven model based on problem solving approach in context of text summarization. Crowdsourcing is a notion where virtually everyone can participate online. In Crowdsourcing, the organization identifies the tasks and then post them online to a crowd to interested people. The organization then evaluates the results by considering the contributions

from the crowd of people who submitted their results by performing the tasks. Crowdsourcing model has been used by Lloret et al. [26]. Hong et al. [27] and Mauyama et al. [28] for text summarization process but there are very few papers on this approach for software artifacts. Nazar et al. [16] have used this model for extracting source code features manually for summarization of source code fragments. Masudur Rahman et a. [29] have used crowdsourcing for mining insightful comments from online QA site StackOverflow. Even though it can be used for generating abstractive summaries as well but most of the researches are for extractive summaries only.

Table 1: Summary of Studies on Extractive Summarization

Author	Artifact	Method	Corpus
Lotufo et al. [4]	Bug Reports	Machine Learning(Unsupervised)	Debian, Mozilla, Launchpad, Chrome
Antoniol et al. [9]	Source Code(C++)	Information Retrieval(VSM)	Albergate, LEDA
Haiduc et al. [10]	Source Code	Information Retrieval (VSM + LSI)	ATunes
Haiduc et al. [11]	Source Code	Information Retrieval + Machine Learning (LSI+ Cluster Based)	ATunes
Rastkar et al. [13]	Bug Reports	Machine Learning(Supervised)	Eclipse, FireFox, Thunderbird
Rastkar et al. [14]	Source Code	Machine Learning(Supervised)	Eclipse Mylyn, CONNECT
Nazar et al. [16]	Source Code	Machine Learning	Eclipse, Netbeans FAQ
Ying et al. [17]	Source Code	Machine Learning(Supervised)	Eclipse FAQ
Mani et al. [19]	Bug Reports	Machine Learning(Unsupervised)	Eclipse, Mozilla, Gnome, KDE
Ferreira et al. [20]	Bug Reports	Machine Learning(Unsupervised)	Bootstrap, AngularJS, jQuery
Andrea Di Sorbo et al. [21]	Emails	Machine Learning (Semi Supervised)	
McBurney et al. [22]	Source Code	Topic Models(HDTM)	JHotdraw, jajuk, jEdit, jTopas, nanoXML, siena
Fowkes et al. [24]	Source Code	Topic Models (extension of TopicSum)	Storm, elasticSearch, Spring-framework, libgdx, bigbluebutton, netty
Rodeghero et al. [25]	Source Code	Eye-tracking interactions	

5. ABSTRACTIVE SUMMARIZATION: AN OVERVIEW IN THE CONTEXT OF THE SOFTWARE ARTIFACTS

Abstractive summarization is to create the summaries where there may be novel sentences which are not present in the original document. It deals majorly with the NLP techniques and requires deep analysis of text. It involves information integration, sentence compression and reformulation [12]. Abstractive summarization techniques are categorized into Structured Based Approach and Semantic Based Approach.

Structured Based Approach finds out the important information from a document using trees, lead and body phrase structures, etc. They are classified into Tree-Based, Template-Based, Ontology-Based, Lead and body phrase method and rule-Based method [30]. From the analysis of recent years papers on summarization, we have observed that most of the works on abstractive summarization in software artifacts have used Template and Rule-based methods.

In Semantic Based Approach, semantic information about the document is used and is fed into the Natural Language Generation system. Verb phrases and Noun phrases are identified and processed by using linguistic data. They are categorized into multimodal semantic model, item based method, and semantic graph based. Multimodal document generation is used when the document contains both images and data. Semantic model is constructed by using knowledge representation in the form of ontologies. Information density metric is calculated to select the sentences to generate the summaries. In information item based approach, abstract representation of source document is created and then is processed to generate the summaries. In semantic graph based method, semantic graph is created where verbs and nouns are represented as nodes and edges to semantic relation between them [30].

Buse et al. [31] generated comments from the exceptions thrown by java methods by using symbolic execution to identify under which conditions the exceptions are thrown and then used the templates to create the summaries. Sridhara et al. [32] found that templates used for comments generation by Buse et al. [31] are inadequate for creating general comments and used method signature and body as input to generate the leading descriptive natural language summary contents for Java methods. They used lead and body phrase method to create the summaries. Buse et al. [33] in another paper combined symbolic execution and code summarization to create automatic summaries for describing code changes. They proposed an algorithm called DELTADOC which takes two versions of software as input and produces human-readable summaries describing method changes as output.

Sonia Haiduc et al. [10] used extractive and lightweight abstractive techniques for summarization. For lightweight abstractive summaries, Structure based approach, lead and body phrase method was used. From evaluation, they found that lead based summaries scored more than any other methods they employed. Sarah Rastkar et al. [34] extracted structural information like how methods interact with the concerned methods and other methods and natural language information like what the concerned method has about and then represented them in the form of ontologies and used RDF graph for ontologies representation. They created the summaries by finding the similarities between the methods of concern and by finding source code important for implementing the concerned method. Latifa Guerrouj et al. [18] generated the summary for a code element from StackOverflow site discussions by extracting the natural language text from developer's discussions. They extracted the language identifiers out of natural language text by applying island parser. They used term-proximity to find out the context of identifiers and then create the language model from them. Cortes-Coy et al. [35] used method stereotypes to identify the method responsibilities in a class for generating the commit messages and used templates to generate the sentences from the classes. Laura Moreno et al. [36] used template-based summarization to generate the natural language summaries for source code changes of a project. One sentence was generated for each change.

Moreno et al. [37] in another paper developed an Eclipse plugin called jSummarizer to generate the natural language summaries for Java classes by using stereotype of class. Abid et al. [38] used method stereotypes to find out the lines of code that reflect the main action of a method and used the separate templates for each stereotype to automatically create the summaries using static program analysis.

Kamimura et al. [39] have used static analysis of source code by finding the key method invocations and then generated the human-oriented summaries of unit test cases by filling the pre-defined templates. Masudur Rahman et al. [29] have used topic modelling and PageRank

algorithm to automatically generate the code comments from StackOverflow discussions related to source code. They used Stanford POS tagger to identify the personal pronouns and possessive pronouns and used natural language processing tools to refine and reformulate the comments. Christopher Vendome et al. [40] have proposed an approach called UnitTestScribe to automatically generate the natural language unit test case summaries. They used natural language processing, static analysis, backward slicing and code summarization techniques for summarization and used general description of test case methods, focal methods, assertions and internal data dependencies of variables in assertions to create the summaries. Shen et al. [41] have proposed an approach to automatically generate the commit messages by summarizing source code. They used template based summarization to describe the commit intents and reason for changes in the system.

CrowdSourcing has been used for generating the abstractive summaries as well. Badihi et al. [24] have created a tool called CrowdSummarizer which uses crowdsourcing, gamification and natural language processing to generate the accurate and comprehensive natural language summaries for java program methods. They used template based method to generate the abstractive summaries.

6. Evaluation Techniques for Software Artifacts Summarization

Evaluation of summaries is a challenging task as it is hard to come up with the notion of what the correct output is. Also, summarization is about compression, so compression ratio should be considered while evaluating the summaries. Visualization is one of the important part of summarization. Compression ratio and visualization when taken together makes evaluation a difficult task. Evaluation techniques are broadly classified into intrinsic and extrinsic evaluation. Intrinsic evaluation is performed on the system itself whereas Extrinsic evaluation is when the impact of system is evaluated like relevance assessment, traceability link recovery in case of software artifacts, reading comprehension, etc.

6.1 Intrinsic Evaluation

Intrinsic Evaluation techniques are more popular in software artifacts summarization. Intrinsic evaluation is to evaluate the automatically generated summaries of the system by comparing them against the gold-set standard summaries for the same system. Intrinsic evaluation techniques mostly focus on redundancy, irrelevant content, coherence and informativeness [42] [43]. Intrinsic evaluation techniques may either involve human intervention or may be automatic. For automatic evaluation, gold-set standard summaries are first created for the artifacts

Table 2: Summary of Studies on Abstractive Summarization

Author	Artifact	Method	Corpus
Buse et al. [31]	Source Code	Template Based	Azureous, DrJava, FindBugs, FreeCol, hsqldb, jEdit, jFreeChart, Risk, tvBrowser and Weka
Buse et al. [33]	Source Code	Template Based	FreeCol, jFreeChart, iText, Phex, and jabref
Sridhara et al. [32]	Source Code	Lead and Body Phrase	Megamek, SweetHome3D, jHotDraw, Jajuk
Rastkar et al. [34]	Source Code	Ontology Based+ Information Item Based	jHotDraw, Drupal and Jex

Guerrouj et al. [18]	Question Answer Site's Discussions	Rule Based	Discussions of Android.* package
Haiduc et al. [10]	Source Code	Lead and Body Phrase	ATunes
Cortes-Coy et al. [35]	Source Code	Template Based	ElasticSearch, Spring Social, jFreeChart, Apache Solr, Apache Felix and Retrofit
Moreno et al. [36]	Source Code	Template Based + Information Item Based	41 Apache community open source projects and 14 other projects like FindBugs, FireFox, Google Web Toolkit, etc
Abid et al. [38]	Source Code	Template Based	HippoDraw
Kamimura et al. [39]	Unit Test Case Source Code	Template Based	jFreeChart
Moreno et al. [37]	Source Code	Stereotype Based+ Template Based	ATunes
Badihi et al. [24]	Source Code	Template Based	11 open source java applications
Vendome et al. [40]	Unit Test Case Source Code	Template Based	srcML.NET, Sando, Glimpse, Google-api-dotnet
Shen et al. [41]	Source Code	Template Based	Elastic Sesarch, Spring Social and Apache Solr

and then the automatically created summaries are compared against them. Precision, Recall, F-Score, Cosine Similarity, etc. are used as measures to compute the performance in case of extractive summaries whereas Relevance-Assessment, Pyramid Scores are used in case of abstractive summaries.

Precision: It refers to the fraction of a total number of sentences in the generated summary which belongs to the gold-set standard summary. It is the measure of how accurate the summaries are. It generally refers to the usefulness in the context of summarization [43].

Precision= (No of lines selected from GSL) \ (No of sentences in the generated summary)
Where GSL refers to Gold-Set Standard Summary Lines

R-Precision: It works well with the variable length summaries. It is the ratio of number of top-R correct words to the number of top-R relevant words returned by the automatic approach.

Recall: It is how much percentage of the sentences in the gold-set standard summaries are present in the generated summary. Recall refers to the completeness in the context of summarization and represents the ability of algorithm to select the sentences from the generated summary [19].
Recall= (No of lines selected from GSL) \ (No of lines in GSL)

Pyramid Score: It is used for assessing the content selection quality in summarization where multiple annotators are available. It is a recall-oriented evaluation metric. It is semantically driven analysis concerned with analyzing the variations in human summaries. It is mainly used for abstractive summaries. Pyramid scores tell how often content units from peer summaries occur in reference summaries.

A=weight of content expressed in the summary
B= weight of an ideally informative summary (Gold-set standard summary) with same number of summary content unit(SCU)

Information with same meaning, even if expressed with different words sis termed as same SCU.

Pyramid Score=A \ B

Pyramid precision is the fraction of sentences present in the summary that are present in atleast one of the golden set summary. Pyramid Recall is the fraction of sentences present in one of the golden-set summaries that are present in the generated summary.

F-Score: There is always a trade-off between precision and recall. F-Score combines the results of both precision and recall. F-Measure acts as a harmonic mean to solve the precision and recall trade-off problem.

F-Score= (2 * (Precision * Recall)) \ (Precision + Recall)

ROUGE (Recall Oriented Understudy for Gisting Evaluation): It is recall oriented metric for evaluation of summaries by comparing the automatically generated summaries against the reference summaries by calculating the number of overlapping units like N-Grams. It measures the quality and correctness of the summary [44]. It is cheap and easy to perform. But it has been found that it should be used only when the large amount of test data is available [45].

Cosine Similarity:

Cosine Similarity (D, E) = (D. E) \ (|D|. |E|)
Where D and E are the word frequency vectors of two text documents. It lies between 0 to 1. If the value is 1, it means documents are nearly identical.

Relative-Utility [45]: Precision and Recall which are used for statistical evaluation suffers from the problem of human variation and semantic equivalence problem,to address this problem, relative utility method is used where multiple judges score each sentence in the summary. Based on their ranks to various sentences, quality of summary is evaluated. This technique requires a lot of manual efforts for sentence tagging.

Sonia Haiduc et al. [10] used intrinsic online evaluation that is Relative Utility method for evaluating their extractive and lightweight abstractive summaries so generated by automatically summarizing the source code. The summaries were evaluated by four developers who answered the questions on 4-likert scale. They also used follow-up questionnaire to understand how developers performed evaluation. Eddy et al. [46] also used the same intrinsic online, relative utility method that is 4-scale Likert-scale evaluation technique as used by Sonia Haiduc [10] to evaluate their source code summaries. Sonia Haiduc et al. [11] in their other paper evaluated their summaries using Pyramid method. Pyramid Evaluation helped find out if the automatically generated summaries are like the set of manually generated summaries. Latifa Gueerrouj et al. [18] used R-precision as an evaluation metric to evaluate how close the generated summaries are with gold-set standard summaries. Rastkar et al. [13] used both the intrinsic and extrinsic evaluation metrics for evaluating the summaries. They evaluated their classifiers using Area Under ROC Curve(AUROC), pyramid precision, precision, recall, and F-Score.

McBurney et al. [2] in their first cross-validation assessment, compared their source code summaries with summaries generated by JavaDoc. They evaluated their summaries based on accuracy, content adequacy and conciseness. McBurney et al. [22] in other paper used human

based intrinsic evaluation by recruiting 3 participants to find out the accuracy and effectiveness of the extractive summaries generated for source code of a project.

Mani et al. [19] used precision, recall, pyramid score, and F-score to determine the effectiveness of unsupervised learning approaches for summarizing bug reports and comparing them with the supervised learning based approaches used by Rastkar et al. [13]. Li et al. [40] used human based intrinsic evaluation to find how concise, expressive and complete are the unit test cases summaries generated by unit test source code. Cortes-coy et al. [47] and Shen et al. [41] evaluated the commit messages generated by code changes based on correctness, content adequacy and expressiveness. Badihi et al. [24] evaluated their code summaries in terms of precision, recall, F-score and overall accuracy. They used 3-scale likert scale to rate the summaries in terms of conciseness, content adequacy and accuracy. Lotufo et al. [4] in the first part of their evaluation used intrinsic evaluations and compared summaries produced by Rastkar et al. [13] by using precision, recall, pyramid score and F-score.

6.2 Extrinsic Evaluation

Extrinsic evaluation techniques are used where the effect of summarization is analyzed on a decision process. They are mostly used for judging the acceptability and accuracy of created summaries. For example, effects of summarization on analyzing the traceability links, analyzing the effects of summarization on question answering sites, etc. They are usually expensive, time-consuming and requires good amount of planning. Thus, are not used extensively for evaluation.

Relevance-Prediction [42] is one measure to measure the effectiveness of summaries on a specific task. It is determining if users can make accurate decisions with generated summaries. Reading Comprehension task is also used for extrinsic

Table 3: Summary of Studies on Evaluation Techniques for Software Summarization

Author	Artifact	Evaluation Method	Criteria (if used)
Rastkar et al. [1]	Bug Reports	Intrinsic	
McBurney et al. [2]	Source Code	Intrinsic + Extrinsic	Accuracy, Content Adequacy, Conciseness
Lotufo et al. [4]	Bug Reports	Intrinsic (Precision, Recall and Pyramid Score) + Extrinsic (Likert scale)	Usefulness
Haiduc et al. [10]	Source Code	Intrinsic (Relative Utility)	
Haiduc et al. [11]	Source Code	Intrinsic (Content Similarity, Pyramid Score)	
Rastkar et al. [13]	Source Code	Extrinsic (Task Based) + Intrinsic (AUROC, Pyramid Precision, Precision, Recall, F-Score)	Accuracy, Time to Completion, Participant Satisfaction
Rastkar et al. [14]	Source Code	Intrinsic (F-Score)	
Nazar et al. [16]	Source Code	Intrinsic (ROC, AUC,	

		Precision, Recall, F-Measure)	
Ying et al. [17]	Source Code	Intrinsic (R-Precision, ROC)	
Guerrouj et al. [18]	Source Code from StackOverflow Sites	Intrinsic (Content Similarity, ROUGE)	
Mani et al. [19]	Bug Reports	Intrinsic	
Ferreira et al. [20]	Bug Reports	Intrinsic (Precision, Recall, F-Score)	
McBurney et al. [22]	Source Code	Intrinsic (Human Evaluation)	Accuracy, Effectiveness
Badihi et al. [24]	Source Code	Intrinsic (Precision, F-Score)	Conciseness, Content Adequacy, Accuracy
Sridhara et al. [32]	Source Code	Intrinsic (Human-Based)	Accuracy, Content Adequacy, Conciseness
Buse et al. [33]	Source Code	Extrinsic	Information, Conciseness
Rastkar et al. [34]	Source Code	Intrinsic	
Cortes Coy et al. [35]	Source Code	Intrinsic	Content Adequacy, Conciseness, Expressiveness, Preferability
Moreno et al. [36]	Source Code + Issue Reports + Commit Messages	Extrinsic	Completeness, Importance, Applicability
Kamimura et al. [39]	Unit Test Cases	Intrinsic	
Li et al. [40]	Source Code	Intrinsic + Extrinsic	Conciseness, Expressiveness, Completeness
Eddy et al. [46]	Source Code	Intrinsic (Relative Utility)	
Masudur Rahman et al. [29]	Question Answering Site Discussions on Source Code	Extrinsic	Accuracy, Preciseness, Usefulness
Fowkes et al. [48]	Source Code	Intrinsic (Precision, Accuracy, Recall, F-Measure) + Extrinsic	Usefulness, Conciseness

evaluation. Rastkar et al. [13] used extrinsic task based evaluation to evaluate if the summaries so generated helps perform duplicate bug reports detection task. They evaluated their system with the help of 12 developers with at least 5 years of experience. They evaluated their system in terms of accuracy, time to completion and participants satisfaction. They used linear regression model and Chi-Square test to find out the accuracy and time to completion.

Masudur Rahman et al. [29] used extrinsic evaluation to evaluate if the automatic comments generated from StackOverflow source code discussions are found to be accurate, useful and precise in terms of describing potential issues, deficiencies and suggesting the future scopes for further improvement. McBurney et al. [2] in their second cross-validation assessment, assessed whether the contextual information about the methods in the summaries help programmers understand the behavior of method. They used cross-validation study method for evaluation.

Li et al. [40] along with the intrinsic evaluation, assessed their unit test cases summaries extrinsically by assessing how their summaries help developers understand the test cases. Moreno et al. [36] assessed their system using extrinsic evaluation in terms of completeness, importance and applicability. Lotufo et al. [4] in second part of their evaluation, assessed the quality of summaries in terms of usefulness while looking for solution around the bug for duplicate bug reports detection, understanding the bug reports in terms of status and open issues, bug prioritization, etc.

7. APPLICATIONS

7.1 Bug Reports Digestion

Bug reports are often too lengthy and involve discussions among multiple team members. Easily digestible bug reports mean that the user who consults the bug reports should be able to understand it easily and be able to seek the desired information quickly. In open source projects, the bug reports receive inputs from many contributors, which makes the bug reports difficult to understand. Summarization of bug reports help understand the bug reports easily. Ankolekar et al. [49] work aimed at bug reports digestion by developing a prototype semantic web interface, Dhruv for bug resolution messages. They identified the important information by aiming at why, who, what from the bug reports. Dit et al. [5] proposed a system to help developers find the comments related to their comments for improving the readability and understandability of thread. Lotufo eta l. [4] proposed unsupervised bug report summarization to facilitate the bug reports digestion.

7.2 Improving Traceability Link Recovery

Recovering and managing traceability links for software artifacts is an important but difficult and time-consuming process as the size of software artifacts is usually large. Traceability link recovery helps improve the program comprehension, software maintenance, ensuring the completeness of project with respect to proper test coverage available or not, impact analysis and during code reuse. Most of the traceability link recovery works have used information-retrieval approaches.

Antoniol et al. [9] have used IR-based approaches to recover the traceability links between the source code and natural language based documents like requirements document, design documents, error logs, etc. Sridhara et al. [32] automatically created the comments to help aid the program comprehension and software maintenance. Aponte et al. [43] have used text summarization IR-based approaches to generate the summaries for software artifacts like requirements document, source code, design documents, test cases and bug reports to identify the candidate links for traceability recovery. Baccheli et al. [51] discussed about recovering traceability links between the source code and emails using text matching. Rigby et al. [15] proposed a novel traceability recovery approach to automatically determine the code elements from Question Answering site, StackOverflow's discussions. Marcus et al. [52] proposed an approach to recover the traceability links between source code and documents by using LSI.

7.3 Program Comprehension

During software maintenance and evolution, a programmer has to go through the source code if proper documentation is not available. Going through the whole source code to make changes in a software is a time-consuming task. Software source code summaries help know quickly what the

source code is doing. Buse et al. [33] proposed an algorithm called DELTADOC which uses symbolic execution information from log messages and code summarization techniques to find the code changes. For maintain the legacy systems, especially for design recovery, lot of documents should be looked up. With text summarization, understanding the legacy systems become easy [9].

Analyzing the code is a non-trivial task especially for beginners. Source code comments not just help in comprehending the source code but also give insights about the quality, issues and future scope. Edmund Wong et al. [53] proposed an approach called CloCom to automatically generate the comments for target software projects by detecting the code clones.

Kamimura et al. [39] automatically generated the summaries for unit test cases which are important. When the source code evolves, it is essential to modify the unit test cases as well. Summaries will ease the comprehension of unit test cases which are usually difficult to understand. Panichella et al. [54] extracted the method description from bug reports and email-threads to help developers understand the source code. Fowkes et al. [24] presented the automatic method for code folding based on code's content which aids in program comprehension while developers want to understand the new code bases, locate the relevant source code and perform code reviews. Hill et al. [51] used context of words around the query terms, method signatures and applied natural language processing techniques to extract the natural language phrases to distinguish between the relevant and non-relevant searches. Automatic formulation of queries using this contextual search helps locate the relevant code elements easily.

7.4 Automatic Documentation Generation

Documentation is expensive to produce and generate as with the change in requirements due to continuous changes in the system, the documentation should be updated timely. But due to lack of time and resources, it becomes difficult for programmers to write the documentation often [3]. Automatic summarization helps in automatically generating the documents. Lot of work has been done in the field. McBurney [3] in his paper has discussed the objectives to improve the automatic documents generation. With their summarization, few of the works where documents are prepared automatically from summarization are mentioned below.

7.4.1 Release Notes Generation

Release notes describe the changes to the system from the previous release to the current release. They describe the changes in terms of documentation changes, license changes, code changes and library upgradations, fixed bugs, modified features, etc. ARENA is one approach towards automatic release notes generation where source code changes are integrated with issue tracker to capture the changes [36].

7.4.2 Commit Notes Generation

Commit notes describe the code changes in the system. They help developers understand how a change to the system spanned across the system's various artifacts and thus help developers support software maintenance tasks like bug triaging, code changes, impact analysis, traceability link recovery, etc. Cortes-Coy et al. [47] presented an approach called "ChangeScribe" to automatically generate the natural language descriptive commit messages describing the change

stereotypes, type of changes and impact of changes using code summarization. Shen et al. [41] proposed an approach like one developer by Cortes-Coy et al. [47] to automatically generate the commit messages describing not just what all changes have been implemented in the system but also the motivation behind the code change. They used ChangeDistiller to find out the change set of source code changes, then identified the what part of information by using stereotype of methods and classified the commit messages according to the type of maintenance supported to explain the why part of information for a commit message.

8. OPEN PROBLEMS AND CHALLENGES

- For evaluation purpose, the generated summaries are usually compared against gold-set standard summaries which are human generated and many times this evaluation is performed by humans itself. Stress, fatigue, or experience can affect the results.
- There are no well-defined standards for good documentation. It is not very clear what the good summary is.
- In conversation related artifacts, due to informal nature of artifacts and presence of source code, segmentation of sentences becomes a challenging task. Sentence Chunking is a non-trivial problem for informal texts like comments in source code, bug reports and email-threads [4].
- As in extractive summarization, the sentences are extracted as in original document and are arranged in the summary as they appear in original document, it is possible to have more redundancy and long length of summaries. Overlapping information may not be captured properly. If pronouns are not properly handled considering the context, then there are chances of erroneous representation of text.
- Whether extractive summarization is most appropriate way to summarization or abstractive, is still an open question [1].
- Representation of abstractive summaries is yet a problem as there is no fixed representation for a text.
- Supervised machine learning based approaches require large training dataset. Gold-set standard summaries are created manually and requires a significant amount of time and efforts. Manually creating a large training dataset is a hindrance towards machine learning for summarization [4]. In summarization process the training dataset is mostly labelled manually and sometimes can be biased towards data on which model was learnt [19]. Quality of corpus affects the quality of summaries and thus good quality corpus is essential for this approach if targeting the quality summaries.
- Summarization of code fragments poses lot of challenges as code fragments are not complete code.
- Heterogeneous and multi-dimensional nature of complex artifacts poses a lot of challenges in summarizing complex software artifacts as they contain various types of fragments like xml, source code, text, etc. and each type of fragment contributes differently to overall knowledge [55].

9. FUTURE SCOPE

9.1 Unit Test Cases Summarization

More work is required for creating the truly useable unit test case summaries. More work on understanding how the summarization of unit test cases eases the work of developers, how to use them in test case generation tools, how to locate the test cases when software fails due to code changes, how will they help in supporting different tasks like testing support is required [39].

9.2 Duplicate Bug Reports Detection

More work on improving the effectiveness of systems for duplicate bug reports detection and for recommending the similar changes to help in software evolution is required.

9.3 Source Code Summarization:

More work on summarization of heterogeneous complex artifacts is required. Most of the focus of studies in source code summarization is for C++ and Java language. Studies can be extended for generating the summaries for other object-oriented languages as well [38]. More work on assessing the quality and effectiveness of summaries for maintenance tasks like feature location or debugging is required [24]. More work on summarizing source code fragments from various sources like online forums is required [21]. The benefits of source code summarization to various other tasks like automatic reverse engineering and re-documentation need to be explored further [11]. More work on creating the techniques which consider the structural information from source code is required [10]. Term based techniques for quickly eliminating the irrelevant entities and NLP based techniques for better evaluation of relevance of remaining entities can be combined to create source code summaries [46].

Multi-document source code summarization considering the packages and classes is required [10]. TASSAL [48] is one of the example of content based model for creating source code summaries but deep learning and other techniques can be used for content based model to create good coverage summaries which to capture the class relationships and other semantic information of source code. Fowkes et al. [48] autofolded the source code at file level, autofolding at statement level can be considered for future work. Automatic documents generation tools which produce documents by summarizing source code are not very expressive [41], work is required to produce more readable and expressive messages. More work on finding what is the most important information to be included in the automatic documents generation summaries and how to classify them is required [36].

9.4 Summarization Using Crowdsourcing

There are very few works on utilizing the crowdsourcing approach for summarizing software artifacts. Nazar et al. [16] have used crowdsourcing for summarization of code fragments in small scale. More work on extending it to large scale is required. More areas where crowdsourcing can be utilized for summarization like corpus creation, features extraction, debugging needs to be identified.

9.5 Summarization Using Eye-Tracking interactions

More work on analyzing the effects of gaze-time, fixations and regressions on long and multi-word identifiers for identifying the important keywords is required. Eye-tracking studies from different roles like testers, security experts, database experts, etc. need to be compared [38].

9.6 Improving Sentence Ranking Techniques

More work on improving the precision of ranking techniques is required. More work on evaluating the influence of text preprocessing in on the quality of ranking techniques is required. More work on improving the estimation of sentence relevance is required [4] as improving the sentence relevance will improve the quality of summary. More characteristics of bug reports and other artifacts need to be considered for summarization to increase sentence relevance.More work on integrating the summarization algorithms and the presentation of summaries with the IDEs is required [41] [4] [36]. Highlighting the important information in the summaries can be done to improve the summary visualization [36].

9.7. Creating Personalized Summaries

Most of the summaries generated till now are general purpose summaries. According to the role of person, the requirements from a document changes. For example, a developer looking for duplicate bug report needs different information than a person looking for bug triaging. By identifying the topics or terms relevant to person, the summary contents can be changed. More work on generating personalized and targeted summaries specific to a role of person or related to some specific software engineering task like bug localization or code review is one of the very potential future research area [17] [24] [48].

9.8. Summary Visualization

For large documents, proper summary visualization helps understand, locate and navigate the summaries easily [4]. More work on creating the optimal interfaces to support navigation based on summaries is required [4]. More work on integrating the summarization algorithms and the presentation of summaries with the IDEs is required [41] [4] [36]. Highlighting the important information in the summaries can be done to improve the summary visualization [36].

CONCLUSION

Summarization of software artifacts help support number of software engineering tasks like duplicate bug reports detection, program comprehension, code search, software maintenance and evolution, automatic documents generation, finding traceability links, etc. The paper gives an overview of the state of the art of summarization techniques mainly in terms of extractive and abstractive summarization. Summaries are either evaluated intrinsically on itself or extrinsically on task basis. The paper gives an overview of evaluation techniques used for summarizing software artifacts. Heterogeneous complex data in software artifacts, in-complete code in code fragments, no well-defined standards for summaries poses challenges during summarization. The paper has also discussed the areas where there is scope of future work like creating personalized summaries according to roles and tasks, improving summary visualization, improving automatic documents generation, unit test cases summarization, etc.

REFERENCES

[1] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: A case study of bug reports," in ICSE, 2010, pp. 505–514.

[2] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," Transactions on Software Engineering, vol. 42, no. 2, pp. 103–119, 2016.

[3] P. W. McBurney, "Automation documentation generation via source code summarization," in International Conference on Research Advances in Integrated Navigation System, l 2015, pp. 35–44.

[4] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," in Inter-National Conference on Software Maintenance, 2012, pp. 430–439.

[5] N. Nazar, Y. Hu, and H. Jiang, "Summarizing software artifacts: A literature review," Springer Journal of Computer Science and Technology, pp. 883–909, 2016.

[6] R. Ferreria, F. Freitas, L. de Souza Cabral, R. D. Lins, R. Lima, G. Franca, S. Jsimske, and L. Favaro, "A context based text summarization," in 11th IAPR International Workshop on Document Analysis System, 2014, pp. 66–70.

[7] M. Indu and K. K. V, "Review on text summarization evaluation methods," in International Conference on Research Advances in Integrated Navigation System, April 2016.

[8] S. Saziyabegum and P. S. Sajja, "Literature review on extractive text summarization approaches," International Journal of Computer Applications (0975-8887), vol. 156, no. 12, Dec 2016.

[9] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 28, no. 10, pp. 970–983, Oct 2002.

[10] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in 17th Working Conference on Reverse Engineering, 2010, pp. 35–44.

[11] S. Haiduc, J. Aponte, and A. Marcus, "Supporting program comprehension with source code summarization," in ICSE 2010, May 2010, pp. 223–226.

[12] N. Rahman and B. Borah, "A survey on existing extractive techniques for query-based text summarization," in International Symposium on Advanced Computing and Communication, 2015.

[13] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," Transactions on Software Engineering, vol. 40, no. 4, 2014.

[14] S. Rastkar and G. C. Murphy, "Why did this code change?" in ICSE 2013, 2013, pp. 1193–1196.

[15] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in ICSE 2013, San Francisco, CA, USA, 2013, pp. 832–841.

[16] N. NAZAR, H. JIANG, G. GAO, T. ZHANG, X. LI, and Z. REN, "Source code fragment summarization with small-scale crowd-sourcing based features," Front. Comput. Sci.), Oct 2015.

[17] A. T. T. Ying and M. P. Robillard, "Code fragment summarization," in ESEC/FSE'13, 2013, pp. 655–658.

[18] L. Guerrouj, D. Bourque, and P. C. Rigby, "Leveraging informal documentation to summarize classes and methods in context," in 37th International Conference on Software Engineering, 2015, pp. 639–642.

[19] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: Approach for unsupervised bug report summarization," in SIGSOFT'12/FSE-20, 2012, pp. 1–11.

[20] I. Ferreira, E. Cirilo, V. Vieira, and F. Mourao, "Bug report summarization: An evaluation of ranking techniques," in 2016 X Brazilian Symposium on Components, Architectures and Reuse Software, 2016, pp. 101–110.

[21] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall, "Development emails content analyzer: Intention mining in developer discussions," in 30th IEEE/ACM International Conference on Automated Software Engineering, 2015, pp. 12–23.

[22] P. W. McBurney, C. Liu, C. McMillan, and T. Weninger, "Improving topic model source code summarization," in ICPC 2014, June 2014.

[23] B. P. Eddy, J. A. Robinson, N. A.Kraft, and J. C. Carver, "Eval-uating source code summarization techniques: Replication and expansion," in ICPC 2013, 2013, pp. 13–22.

[24] S. Badihi and A. Heydarnoori, "Crowdsummarizer :automated generation of code summaries for java programs through crowd-sourcing," IEEE Software, pp. 71–80, 2017.
[25] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan, "An eye-tracking study of java programmers and application to source code summarization," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, pp. 1038–1054, 2015.
[26] E. Lloret, L. Plaza, and A. Aker, "Analyzing the capabilities of crowdsourcing services for text summarization," Springer Sci-ence+Business Media B.V. 2012, pp. 338–369, 2012.
[27] S. G. Hong, S. Shin, and M. Y. Yi, "Contextual keyword extrac-tion by building sentences with crowdsourcing," Springer Science+Business Media New York 2012, 2012.
[28] H. Mizuyama, K. Yamashita, K. Hitomi, and M. Anse, "A proto-type crowdsourcing approach for document summarization service," in IFIP International Conference on Advances in Production Management Systems, 2013, pp. 435–442.
[29] M. M. Rahman, C. K. Roy, and I. Keivanloo, "Recommending insightful comments for source code using crowdsourced knowledge," in SCAM 2015, Bremen, Germany, 2015, pp. 81–90.
[30] H. T. Le and T. M. Le, "An approach to abstractive text summa-rization," in International Conference of Soft Computing and Pattern Recognition, 2013, pp. 371–376.
[31] R. P. Buse and W. R. Weimer, "Automatic documentation inference for exceptions," in ISSTA 2008, 2008, pp. 273–281.
[32] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in ASE 2010, 2010, pp. 43–52.
[33] R. P. Buse and W. Weimer, "Automatically documenting program changes," in ASE'10, 2010, pp. 33–42.
[34] S. Rastkar, G. C. Murphy, and A. W. Bradley, "Generating natural language summaries for crosscutting source code concerns," in 27th International Conference on Software Maintenance, 2011, pp. 103– 112.
[35] L. F. Cortes-Coy, M. Linares-Vasquez, J. Aponte, and D. Poshy-vanyk, "On automatically generating commit messages via sum-marization of source code changes," in 14th IEEE Working Conference on Source Code Analysis and Manipulation, 2014, pp. 275–284.
[36] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora, "Arena: An approach for the automated generation of release notes," Transactions on Software Engineering, 2016.
[37] L. Moreno, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Jsummarizer: An automatic generator of natural language summaries for java classes," in ICPC 2013, San Francisco, CA, USA, 2013, pp. 230–232.
[38] N. J. Abid, N. Dragan, M. L. Collard, and J. I. Maletic, "Using stereotypes in the automatic generation of natural language summaries for c++ methods," in ICSME 2015, Bremen, Germany, 2015, pp. 561–565.
[39] M. Kamimura and G. C. Murphy, "Towards generating human-oriented summaries of unit test cases," in ICPC 2013, San Francisco, CA, USA, 2013, pp. 215–218.
[40] H. Li, C. Vendome, M. L. Vasquez, D. Poshyvanyk, and N. A. Kraft, "Automatically documenting unit test cases," in International Con-ference on Software Testing, Verification and Validation, 2016, pp. 341– 352.
[41] J. Shen, X. Sun, B. Li, H. Yang, and J. Hu, "On automatic summarization of what and why infoormation in source code changes," in 40th Annual Computer Software and Applications Conference, 2016, pp. 103–112.
[42] B. J. Dorr, C. Monz, S. President, R. Schwartz, and D. Zajic, "A methodology for extrinsic evaluation of text summarization: Does rouge correlate?" in Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, June 2005, pp. 1–8.
[43] J. Aponte and A. Marcus, "Improving traceability link recovery methods through software artifact summarization," in TEFSE 2011, May 2011, pp. 46–49.
[44] V. Gupta, "A survey of various summary evaluation techniques," International Journal of Advanced Research in Computer Science and Software Engineering, pp. 159–162, 2014.
[45] A. Nenkova, "Summarization evaluation for text and speech: Issues and approaches," in INTERSPEECH 2006, Sep 2006.

[46] B. P. Eddy, J. A. Robinson, N. A. Kraft, and J. C. Carver, "Evaluating source code summarization techniques: Replication and expansion," in ICPC, 2013, pp. 13–22.
[47] L. F. Cortes-Coy, M. Linares-Vasquex, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via sum-marization of source code changes," in International Working Con-ference on Source Code Analysis and Manipulation, 2014, pp. 275–284.
[48] J. Fowkes, P. Chanthirasegaran, and R. Ranca, "Autofolding for source code summarization," IEEE Transactions on Software Engi-neering, 2016.
[49] A. Ankolekar, K. Sycara, J. Herbsleb, R. Kraut, and C. Welty, "Sup-porting online problem-solving communities with the semantic web," in WWW '06 Proceedings of the 15th international conference on World Wide Web, 2006, pp. 575–584.
[50] B. Dit and A. Marcus, "Improving the readability of defect re-ports," in RSSE '08 Proceedings of the 2008 international workshop on Recommendation systems for software engineering, 2008, pp. 47–49.
[51] E. Hill, L. Pollock, and K. V. Shanker, "Automatically capturing source code context of nl- queries for software maintenance and reuse," in ICSE 2009, 2009, pp. 232–242.
[52] A. Marcus and J. I. Maletic, "Recovery of traceability links between software documentation and source code," International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, pp. 811–836, 2015.
[53] E. Wong, T. Liu, and L. Tan, "Clocom: Mining existing source code for automatic comment generation," in SANER 2015, Montreal, Canada, 2015, pp. 388–389.
[54] S. Panichella, J. Aponte, M. D. Penta, A. Marcus, and G. Canfora, "Mining source code descriptions from developer communications," in ICPC 2012, Passau, Germany, 2012, pp. 63–72.
[55] N.Moratanch and S.Chitrakala, "A survey on abstractive text summarization," in International Conference on Abstractive Summarization, 2016.
[56] L. Ponzanelli, A. Mocci, and M. Lanza, "Summarizing complex development artifacts by mining heterogeneous data," in 12th Working Conference of Mining Software Repositories, 2015, pp. 401–405.

Authors' Profiles

SomGuptais a Research Scholar in Computer Science Department in AKTU Lucknow. She has done MTech from IIIT Bangalore and BE in Computer Engineering from Gujarat University. She has 2 years of working experience. She has worked with American Express Technologies, Gurgaon; taught in Chandigarh University and has trained Graduates Java and Android technologies. Her area of interest includes natural language processing, machine learning, software engineering and databases.

Dr. S. K. Gupta is presently working as A.P. in Comp. Sc. & Engg. Deptt. of BIET, Jhansi, U.P., INDIA. Initially graduating in Computer Science & Engineering from H.B.T.I., Kanpur, U.P., INDIA and then M.E. from M.N.R.E.C., Allahabad, U.P., INDIA and after that completed Ph.D. in Computer Science & Engineering from Bundelkhand University, Jhansi, U.P., INDIA. His area of interest includes image processing and data mining.