Automatic Release Notes Generation

Author

Mubashir Ali

FALL 2017 – MS-17 (CSE) 00000206994

Supervisor

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

Automatic Release Notes Generation

Author

Mubashir Ali

FALL 2017 - MS-17 (CSE) 00000206994

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Software Engineering

Thesis Supervisor:

Dr. Wasi Haider Butt

Thesis Supervisor's Signature:-__________________________________

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

DECLARATION

I certify that this research work titled *“Automatic Release Notes Generation”* is my own work under the supervision of Dr. WasiHaider Butt and co-supervised by Dr. MuazzamKhattak. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Mubashir Ali

FALL 2017 - MS-17 (CSE) 00000206994

PLAGIARISM CERTIFICATE (TURNITIN REPORT)

The plagiarism of this thesis has been checked. Turnitin report approved by the supervisor is attached.

Signature of Student

Mubashir Ali

FALL 2017 - MS-17 (CSE) 00000206994

Signature of Supervisor

Dr. Wasi Haider Butt

LANGUAGE CORRECTNESS CERTIFICATE

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student
Mubashir Ali
FALL 2017 - MS-17 (CSE) 00000206994

Signature of Supervisor
Dr. Wasi Haider Butt

ACKNOWLEDGEMENTS

I am extremely thankful to **ALLAH** Almighty for his bountiful blessings throughout this work. Indeed this would not have been possible without his substantial guidance through every step, and for putting me across people who could drive me though this work in a superlative manner. Indeed none be worthy of praise but the Almighty. In addition, my admirations be upon Prophet **Hazrat Muhammad (PBUH)** and his Holy Household for being source of guidance for people.

I would like to express my special thanks to my supervisor **Dr. WasiHaiderButt** and co-supervisor **Dr. MuazzamKhattak** for his generous help throughout my thesis, and for being available even for the pettiest of issues. My thanks for a meticulous evaluation of the thesis, and guidance on how to improve it in the best way possible.

I am profusely thankful to **Dr. ArslanShaukat** and **Dr. Urooj Fatima** for an excellent guidance throughout this journey and for being part of my evaluation committee.

It is indeed a privilege to thank my Mother, my father **Zulfiqar Ali** and my elder brother **Mudassir Ali** for their constant encouragement throughout my degree and research period. The sense of belief that they instilled in me has helped me sail through this journey. I would like to thank my Family & friends.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance in this period.

Dedicated to my exceptional parents, excellent siblings and my best friend whose tremendous support and cooperation led me to this wonderful accomplishment. I am truly indebted to you all.

ABSTRACT

Release Notes (RNs) are one of the important artifacts in software development and maintenance. RNs are required when a new release/version of a software project is planned to deploy or launch. RNs are the documentation of all the changes made to the new project It contains description of newly added features, bug fixes, improvements, removed functionality, library changes etc. Generating these notes manually is very complex and time consuming task. In this research we present an automatic approach for generation of RNs. The intention of this research is to generate automatic RNs for Node JS project. The implementation of the proposed approach is done in Python programming language. Our system extract changes from Git repository, summarizes these changes, get deprecated features, library changes, fetches issues from issue tracker and link these issues to code etc. The system hierarchically setup these changes and produce an output in a word document. This approach is the state of the art approach for node Js projects. We evaluate our results manually from 15 industry software Engineers. The results obtained from our system shows that these RNs are very good and accurate then ones produced manually.

Keywords— Software Release Notes; Software documentation; Software Maintenance; Programming language processing; Natural Language Processing;

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

This chapter contains a brief introduction of the research performed. **Section 1.1** explained the release notes. The problem statement is stated in **section 1.2**.**Section 1.3** includes the proposed methodology and **Section 1.4** provides a brief overview to our research contribution. Lastly, the thesis organization is stated in **Section 1.5**.

1.1. Introduction

Software development life cycle has many steps. Different kinds of artifacts are associated with them in these stages. One of these artifacts is RNs. RNs are generated when a new version of a software project is planned to release. It aids developer in maintaining project. These notes contain new features, improvements, changes to classes, libraries, methods etc. They are very helpful for the developer to identify and classify different API changes [1].Usually these notes are generated manually. Release manager or developers create these notes by analyzing the changes made to project. Generating manually is very difficult and time consuming.

Continuous delivery methodology of software engineering improves the customer's satisfaction and product's quality which are the most important factors in software development [2], [3]. There was still a gap between user and developer. They need to communicate using a document. RNs are these documents through which they communicate with each other and exchange their progress and ideas with this. These notes are generated on every new version/release of their project. Writing RNs on every release is difficult task for a developer. Automation of these notes helps developer to focus on the quality of project. These notes contain all the new changes made to the new version. Changes made to library, changes made to license, newly added features, deleted features, fixes made to bugs etc are few of changes we are talking about that RN contain. This document is very helpful for the developer to find bugs, very meaningful to manager to learn more about the project, and if the project is handover to another engineer it will aid the engineer to learn about the project very easily. Testers get advantage from the document in testing.

Automation in software engineering enhances the whole lifecycle from requirements to maintenance. Manual generation of RNs is very complex and time consuming task which slow down the development. Every stakeholder has different types of concerns with these documents.

Developers are interested in structural changes like changes to functions, classes etc, end-user read the document to learn what new feature has been added to the system and the tester needs to know which part of the system is to test. It is very difficult and time taking to generate separate notes for every types of stakeholders manually which generally took 8 hours [1]. In case of a project where multiple developers working, manual generation of these notes are more complex and difficult for the release manager because he/she does not know everything about the system.

Generating RNs automatically saves time and other resources. The notes generated automatically are more accurate and concise than manual ones [4]. RNs at a different level of granularity and in different formats can be generated easily. The problem in RN generation is what types of changes (made to the new system) are to select and how to prioritize them in the final documene. [5] Conducted a study in which they analyze RNs of multiple software projects and evaluate them by looking at their features and how they present the final document. The main output of their study is they discuss about the type of changes which should be in the RNs. Discussion of different Machine Learning and Artificial Intelligence algorithms of RN feature extraction is the part of their research too.

1.2. Problem Statement

Software automation changed the world of software development. Software RNs are generated when deploying the new build of a project. It is very helpful for developers in maintenance and further development. Generating these RNs manually are very complex and time consuming. It takes 8 hours, on average, to generate these notes [4]. It is even more complex and difficult if more than 1 developers working on the same project.

There are very limited research and algorithms available for automation of this process. They have done very good work but needs a lot of improvement. The main limitation of these tools and algorithms is that they only worked on JAVA projects. Our focus of research is node JS. There is no research available that automate the release notes generation for Node JS.

To automate the release notes generation is the problem we want to solve through this research. Automation of these notes save time, cost and other resources.

2.3. Proposed Methodology

In this research we proposed an approach based on NLP and heuristic rules to generate these rules automatically. We analyzed the existing approach to highlight the standards used generation of release notes. After studding the previous research we realized that there is need for improvement in this research. Moreover every previously proposed research generate these notes for JAVA programming language. There wasn't any research done on JavaScript or any of the JavaScript framework. We propose and developed a novel approach to get better results than previous a state of the art systems for JavaScript project (specifically Node JS).

The **Figure 1** shows the overall process of the proposed approach. There are four main modules in the proposed system. Every step process some data and provide the output to the next step.

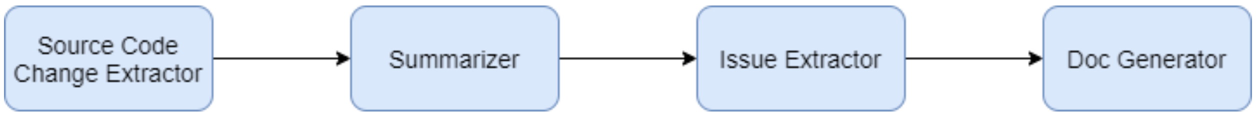


Figure 1. Flow of Proposed Approach

The entire research is done in a systematic way. Flow of the research is shown in **Figure 2**. First step is to identify the problem then the solution for the identified probloem is proposed. The next step is to carry out a comprehensive Systematic Literature Review (SLR) which becomes the foundation of the proposed solution. Researches related to the proposed solution are analyzed and compared.

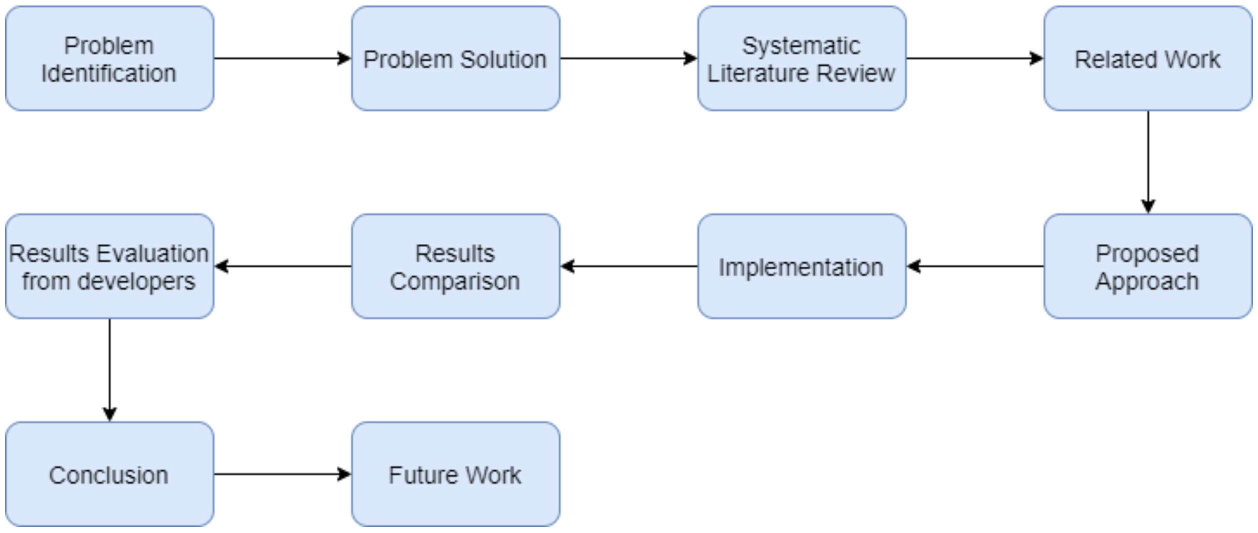


Figure 2. Research Flow

The proposed work includes a fully automated approach to generate the RNs. The proposed methodology has been compared with the previous methodologies to compare the improvement.

1.4. Research Contribution

The main contribution made by our proposed research is to save time, cost and other resources in software development life cycle and maintenance.

Some of the key contributions of our proposed research are listed below:

- We have performed anSLR on automated generationrelease notes. Through this literature review, we identified different techniques used in past for this purpose from researches reported in detail in **Chapter 2**. Similarly, our SLR has drawn the NLP algorithms used. The concluded results show that our study would provide an advantage to the researchers in future it is the very first systematic literature review carried out on automated release notesgeneration. It will help practitioners in this field to overview the results to explore more investigate maturity of this process.
- We provide a list of all the tools and plugins used in industry, for different platforms, for generation of release notes generation.
- We compare all the tools and techniques used in this context.

- We have exploited text mining for the determination and development of a novel framework more specifically rational expression for the transformation rules on basis of formal language theory.
- We proposed a novel approach to generate release notes for JavaScript projects
- Industry projects are used to evaluate the results generated by our proposed approach. Further, the results are validate from different industry developers of Node.js.
- The proposed approach is implemented in python.

1.5. Thesis Organization

Figure 3 shows how thesis are organized. The short description of every chapter is listed below.

Chapter 1: Introduction offers a brief introduction containing the background study, problem statement, research contribution and thesis organization. **Chapter 2**: Literature Review provides the detailed literature review highlighting the work done in the domain of automated bug report summarization. The systematic literature review is composed of three main sections. First section is review protocol which gives details on the methodology using which the literature review is carried out. Section two offers details on research works, whereas; section three highlights the research gaps that we encountered. **Chapter 3**: Proposed Methodology covers the details of proposed methodology used for identification of problem. How this approach works and what are the main modules of this process. It will also discuss the module in detail with all its architectural problems and solutions. **Chapter 4**: Implementation presents the detailed implementation regarding the proposed tool. **Chapter 5**: Validation provides the validation performed for our proposed methodology using nine important case studies. **Chapter 6**: Discussion and Limitation contains a brief discussion on entire work performed along with limitations to our research. **Chapter 7**: contains the Future Work and Conclusion.

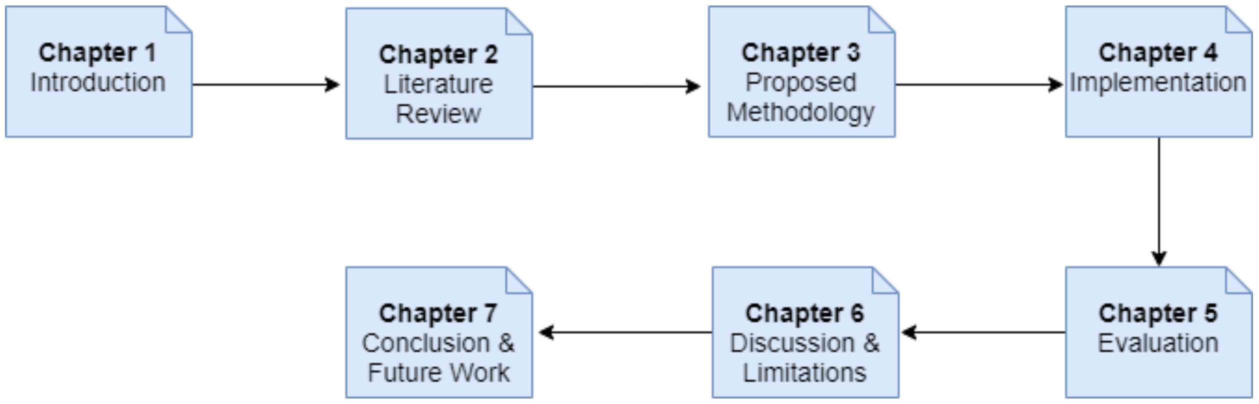


Figure 3. Thesis Outline

Chapter 2

Literature Review

CHAPTER 2: LITERATURE REVIEW

RNs play a vital role in software development. RNs are one of the essential artifacts about a new release of the software. Despite their importance, RNs are usually generated manually. Generating them manually prone to errors and time consuming as it contains a description of new features, bug fixes, license changes, deprecated libraries, new Application Program Interface (API), and other changes made to the software. There are different tools available to generate RNs automatically from issue tracker and source code repositories. Researchers are working on the automation of RNs from decades to enhance the software development process. In this chapter, we present the algorithms published by different researchers, available plugins, and tools in the market for this task. This survey of the past techniques used for the automation of RNs is useful and will be more helpful for future researches regarding this problem.

Introduction

Software automation changed the world of software engineering by improving software maintenance processes, methods, and tools[7]. These are the means of communication between the developer and other stakeholders. RNs are usually generated when a new version of the software system is planned to deploy. It contains information about the changes in new release i.e., fixes of issues, new functionalities, deprecated libraries, and APIs. They are helpful for the developer to identify and classify different API changes [1]. Different RNs are generated for different types of stakeholders. Changes to code, structural changes, a new library, and addition/removal of functions and classes are essential for developers while end-users concerned with new features and bug fixes.

Usually, these notes are generated manually by the developer or release manager. Creating these notes manually is a challenging and effort prone task, which generally takes 8 hours [8]. Emergency updates often occur in projects, especially in mobile applications, due to deployment issues or other essential requirements/features [9]. Keeping track of these changes in such cases is hectic. Most of the time, more than one developers work on a single software and creating RNs, in this case, is very complicated as a different section of the software is developed by a different developer. Creating RNs by a manager or single developer, in that case, is very limited in its features as he/she won't have a thorough knowledge of all the changes and features.

Continuous delivery improves the quality of product and increases customer satisfaction [10][3]. Still, there is a gap between customer feedback and the development of new features in the continuous delivery method [11] as there is no way to generate RNs for every delivery. Creating RNs for every stakeholder that matches the target group is another problem in RNs generation, and it takes time. Every stakeholder have their own concerns with the RNs [12], e.g.

- Testers need to know which functionality to test
- Project manager concerns with the new features and bug fixes
- A developer wants to know the changes occur to code (structural changes, changes to functions/class and API etc.)

Automatic RN's generation enhance the quality of document and also it save other resources. RNs at a different level of detail and in multiple document formats can be generated easily. According to research [8], automatic creation of these notes are more precise and correct than maunal generation.

Another main problem in RNs is which type of changes are to include in the final docment of RN and how to prioritize them in the document. Abebe and his colleagues conducted a research which compare different RNs and their fiearures [13]. They have discussed which changes to include it RNs and which are not. Different machine learning models are mentioned in their article which are used to identify type of changes and features for RNs. IT companies are shifting towards continuous delivery and launch/relase their product on weekly or monthly basis [6]. In this case automation is more helpful.

Many researchers devoted their studies and time to automate this process of RNs generation. They achieved significant results in this area. Few algorithms fully automate the creation of these notes while others work on semi-automation of this process. But there is a lack of some systematic literature study of the past techniques that can give the overall understanding of the future researcher that which technique is better and how it works. This paper explores different research conducted since 2000 regarding these notes creation. A systematic literature review provides an excellent quality assessment of a research study by using a reliable and accurate methodology [14].

The rest of the chapter is structured as in **section 2.2** we will discuss the methodology used for this literature review, **Section 2.3** is about different techniques, tools and plugins proposed for this automation and their results and **section 3.4** conclude the whole discussion.

2.2. Methodology

In [15], Evidence-based software engineering provides the standards by which the best of evidence are addressed in regards to this research study with human values and practical experiences in the process of decision making by just considering the software maintenance and development. Systematic Literature Review (SLR) is our methodology in this paper. There are four stages: Research Questions, Data Search, Inclusion and Exclusion Criteria, and Data Collection.

2.2.1. Research Questions

We have developed the following research questions for conducting this systematic study of literature:

- How many algorithms and tools are proposed since 2000 in this context
- What methodology and techniques used to automate the process of RNs creation
- Semi-automatic creation of RNs.
- What specific tools and plugins are available in the market for this task?

2.2.2. Data Search

For research articles searching from IEEE, Springer, Elsevier, and ACM databases, we selected keywords: RNs, Natural Language Processing (NLP), Programming Language Processing, Source code, deep learning, and Issues Linking. Keywords have been used with the number of filters to find the research articles related to our topic and criteria. The four selected databases are used for the searching research articles with different filters i.e. publication date (must be in 2008-2020) and combination of above keywords with AND/OR operators. Moreover, **Table 1** contains the list of all searching keywords which are used with AND/OR operators and results of their search. After performing with the above filtering and keywords we selected a total of 6183 research papers seems related to our research questions. Further, we have done deep analysis and filtering on research articles by applying our inclusion and exclusion rules. The last filtering is done on by reading and understanding their abstract, general study and publications. The further process is depicted in **Figure 4**.

2.2.3. Inclusion and Exclusion Criteria

Criteria for research paper and tool selection is its relevance with our research questions and result generated by the methodology. Inclusion criteria is:

- Included all those tools and techniques that directly or indirectly related to the automatic/semi-automatic creation of RNs.
- Include research or tool which at least answer one of the research questions.
- Research conducted after 2000 will be included in the paper.
- Scientific databases used for research are IEEE, Springer, Elsevier,and ACM. Research articles from these databases will only be selected.

Our exclusion criteria is:

- Exclude research study which is irrelevant to our research questions.
- Exclude research study which is repeated.
- Exclude the duplicate research study, if similar research is proposed in two research articles, only the latest work will be included.
- Reject research studies proposed before 2000.
- All research must be in English language.

Table 1. Execution of Search String

Sr. No.	Keywords/ Search Terms	Operators	Number of Research Papers			
			IEEE	Springer	ACM	Elsevier
01	Release notes and NLP	AND	2	32	1	11
		OR	3286	50451	164	18773
02	Release notes and Programming Language Processing	AND	25	0	15	0
		OR	15085	1825	164	1446
03	Source Code and NLP	AND	43	1873	42	499
		OR	37912	118767	9238	64109
04	Release notes and Source code	AND	39	356	18	138
		OR	35120	73315	164	48278
05	Release notes and Deep Learning	AND	8	17	2	4
		OR	28942	36167	164	21205

06	Issues Linking and Release notes	AND	115	722	11	1
		OR	10015	4555	5695	3512

2.2.4. Data Collection

We got many algorithms, tools, and plugins for the automatic creation of RNs. After reviewing their proposed algorithms and their results after quality assessment, we selected a few algorithms and tools. Quality assessment is an essential phase in research studies. We have performed the inclusion and exclusion criteria to find research articles from databases and select only those research articles that meet our criteria.

Firstly, we chose different research articles and tools through their titles and recorded their Bibliographic data. Then start analyzing their abstract and select the most relevant researches. At this phase, we excluded many research articles and chose the rest. After that, we study the proposed algorithms, their methodology, results generated by their algorithms, and how they evaluated their results and recorded their summaries. At this stage, few research articles are rejected.

Results of our selected papers, algorithms, tools and plugins are discussed in the next section.

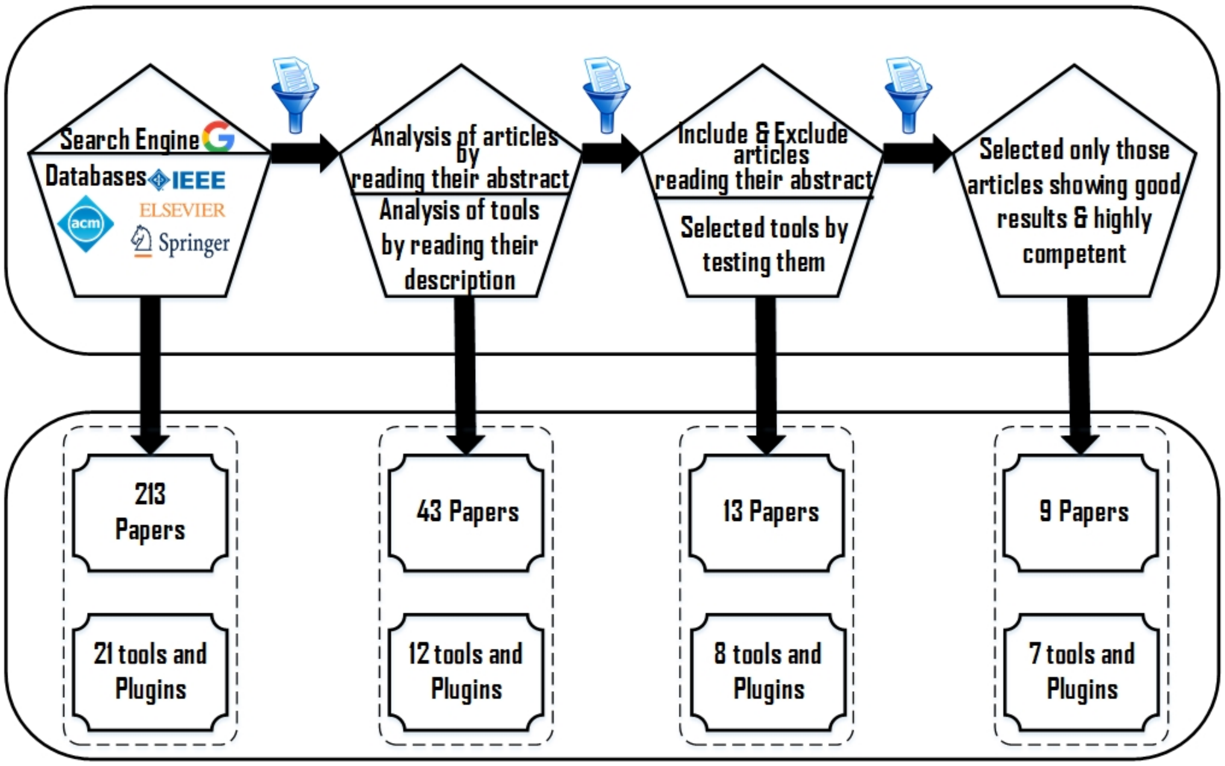

Search Engine
Databases IEEE
ELSEVIER
acm
Springer
Analysis of articles by reading their abstract
Analysis of tools by reading their description
Include & Exclude articles reading their abstract
Selected tools by testing them
Selected only those articles showing good results & highly competent
213 Papers
21 tools and Plugins
43 Papers
12 tools and Plugins
13 Papers
8 tools and Plugins
9 Papers
7 tools and Plugins

Figure 4. Systematic Review Process (SRP)

2.3. Research Findings

We have successfully got the target research algorithms and tools but didn't find any previous systematic research articles describing our research questions. We have seen very few research articles regarding the automatic generation of RNs from our databases. We have also spent our time in searching for different plugins and libraries used for this task from the search engine. In our opinion, this study is the first systematic literature review of automatic RNs generation. The found research algorithms and plugins are discussed below.

2.3.1. Research Algorithms and Techniques

Laura Moreno worked to create RNs between commits automatically from the versioning system and issue tracker [8]. He extracts changes from the versioning system, converted these changes to NLP sentences, and then link these changes to issue tracker. They summarize the code changes and link it to information extracted from commit notes and issue trackers. They used srcML toolkit [16] to parse source code and extract facts about file been added, removed, deleted, etc. Summaries for java classes are generated by the technique proposed by Laura Morenoa and her co-workers [17]. They conduct 4 case studies for completeness, importance, industry developer level, and in the field study.

In[12], a semi-automatic approach is proposed for the creation of RNs. The focus of their research is agile methodology and user-specific notes. According to their research, every stakeholder has different concerns with RNs. They elicit information about a release from build server, issue tracker, and version control system and then generate a list of changes for this release. There should be a release manager who will review RNs, can edit it, and also customize creation by filtering.

Reno [18] is a tool that is used for the generation of RNs. The generated are in a standard format organized into the group's new features, bug fixes, known issues, etc. are also generated automatically with this tool using source code. The release manager is responsible for generating and providing these notes to developers.

In [19], researchers developed a model-based approach to generate automatically by discovering patterns from different satellite projects. They perform an empirical study on a broad set of satellite projects in order to classify entities of RNs. After that, patterns are defined for these categories and at last generated by the discovery algorithm. Colored Petri Nets (CPN) is used as a discovery algorithm in this project.

RNs are important as they identify what is delivered to users. [20]Chandrasekra proposed a methodology in his research article for the generation of RNs automatically for Team Foundation Server (TFS). They deeply study and inspect every steps of software methodology from requirement to deployment and automate these note's creation.

In [21], a technique is proposed to populate a release history database that combines versions data with the bug report. In this way, they maintain a structured database. A full meaning data can be extracted by querying this structured data. H. Gall and his colleagues extract dependencies between modules and classes from change log and source code [22].

Generating natural language summaries of source code aid in the documentation of a project. There are many algorithms proposed for generating summaries from source code [17],[23], [24] and [25].

2.3.2. Plugins and Libraries

By using the Google search engine to find different plugins and libraries are used for the creation of RNs automatically. Most of them just used a change log to create, and few also considered other artifacts of software engineering i.e., issues, software requirements, etc. **Table 2**shows all these tools.

Atlassian generator [26], is a deployment plugin developed for Bamboo Server. It generates in HTML format from the commit differences between two releases. It includes all the commits between the current and the last commit. You have to configure it in a bamboo server.

Chronicler [27] is an open-source node.js that works with github repo. It will listen for pull request events from Github Webhooks. When a pull request is merged, Chronicler will create a new release draft OR edit an existing one with the new pull request info. Final output contains the list all pull requests messsages.

Azure pipeline generator [28] is a PowerShell task and only supported on windows. A node.js version of this plugin is also available for multiple platforms. It generates files as part of

VSTS/TFS build or release pipeline. It works with both Azure Server and Azure DevOps Services. There are multiple templates for RNs depends upon requirements. The power shell runs each line of the template as command and generates output for RNs.

Another tool [29] we found is a python package used to generate RNs using pull requests. It requires a GitHub access token. It gets the description for each pull request and put these data into RNs.

Semantic generator [30] generates change log using conventional changelog. First will have to configure it in a configuration file and add preset and keynotes to search. For every preset, the corresponding conventional package must be installed.

Automated RNs generator [31] and Better PDF Exporter for JIRA [32] plugins are used with the Jira server to automate RNs generation. Multiple formats are supported by the former one i.e., pdf, JSON, markdown, etc. Scheduling RNs generation is also available. In the Better pdf exporter, multiple pdf templates are available in this plugin to use. Jira Query Language (JQL) is used to extract any information from any issue on Jira and inject it anywhere in the notes in both plugins.

A Gitlab release note generator [33] that create RNs on the latest tag. It requires node.js or Docker. It generates RNs between two tags of the same branch. It generates the changelog of these two tags. It looks for the following labels in commits enhancement, breaking change, feature, and bug.

Table 2. Identified Plugins and Libraries

Sr. No.	Selected Plugins and Libraries	Requirements
01	Attlassian Release Notes Generator	Bamboo Server
02	Chronicler	Node JS and Github
03	Release Notes Generator	Github
04	Semantic Release Notes Generator	Github
05	Azure Pipeline Release Notes Generator	Power shell or Node JS and VSTS or TFS
06	Automated Release Notes Generator	JIRA Server
07	Gitlab Release Notes Generator	Gitlab

2.4. Conclusions

Automation of artifacts in software engineering helps developers and managers to save time and increase the quality of reports. RNs play a vital role in every new release of software for every stakeholder. It takes time to generate and thorough analysis of source code, new features and resolved issues. Many researchers spend their time to automate the creation of these notes. In this paper, we investigate the efforts of different researchers and developers in this regards. All these generate very good results but according to our knowledge and understanding, Arena [8] algorithm and their result are the best one among all these tools. They evaluate their results from developers and managers. [12] Proposed technique is also very good as their target software methodology is agile development and they cover different stakeholders. It is concluded that researchers and engineers have proposed very good algorithms for RN generation automatically but there is still need for improvement.

Limitations of our study are that we limited our study to just include the of software projects. We only considered the research articles related to automation of software and did not discuss the formation of software. In the future work, we will discuss the formation of and their structure.

Chapter 3

Proposed Methodology

CHAPTER 3: PROPOSED METHODOLOGY

This chapter contains detail of the proposed methodology. **Section 3.1** discusses the targeted core concepts explanation, **Section 3.2** provides the detailed proposed methodology.

3.1. Core Concepts Explanation

3.1.1. NLP

Natural Language processing usually referred as NLP is an Artificial Intelligence Branch. This field is related to helping computers to interpret the human language, understand it and then able to manipulate it [34]. Basically, Natural Language processing is actually a subfield of Computer Science, Artificial Intelligence and Linguistics and it is related to the interaction between human's languages and computer. It's about how to teach computers that how to analyze and process the natural language. This field can be further breakdown in speech recognition, natural language understanding, natural language summarization and natural language generation etc. Natural language processing is used to help the developers to manage and organize the work knowledge to perform their tasks like summarization, named entity relationship, translation, information retrieval or relationship extraction, speech recognition and topic segmentation etc.[35]. This field helps computers and create automated systems that can understand and analyze a human languages like Arabic, Latin or English etc.

3.1.2. Abstract Syntax Tree (AST)

AST represents the syntactic structure of source code. It is used in interpreter and compiler to generate machine code or evaluate expression[36]. It tells the interpreter how to interpret the statement and compiler how to translate the code. It represents the code in a tree based data structure. It contains different nodes. The node that is above the all is called root node, node with one of more child node is called interior node and a node without any child is called leaf node. We use AST to get the syntactic information from code i.e. functions/methods changes, new classes, modified classes, endpoints etc.

3.1.3. Control Flow Graph (CFG)

CFG is a graphical representation of the flow of a program. It shows all the possible paths that can be traversed during the execution of program[37]. We implemented this module to get all the returns and non-return ending statements of functions. This module aid in developing the natural language summaries of function.

3.1.4. Def-Use Chain

Def-Use chain is a data structure of a program that contains all the definition and uses of a variable[38]. It is necessary in our system because we traverse back every selected variable (fulfilled the rules). Without this, it is not possible to generate the exact natural language summary for a statement or expression containing a variable.

3.1.5. Diff

Git diff is used to extract the changes between different artifacts. We use diff to extract the changes of files.

3.2. Proposed Methodology

The overall architecture of the proposed approach for RNs creation automatically is shown in **Figure 5**. Four main steps/modules of the system are:

- Extraction of changes
- Human language (English) sentence generation
- Extraction of issues
- Combined all information and generating final document

The first step is Changes Extractor which fetches the changes made to the new system in this new version. Diff command of git provides us this information in versioning control system. Newly added, deleted and modified files are extracted at higher level and on more detailed level it split all these files into classes, functions and methods. Changes made to libraries and license are also extracted.

Next we need to generate human language sentences for all these selected changes in the next step. Generating English language summaries for them is the responsibility of this module. Then all the summaries are attached to their original file in their original order.

Issues extractor used in this research is JIRA. All the issues reported in JIRA within the timeline of release are selected and linked to the code is the job of this Issue Linker.

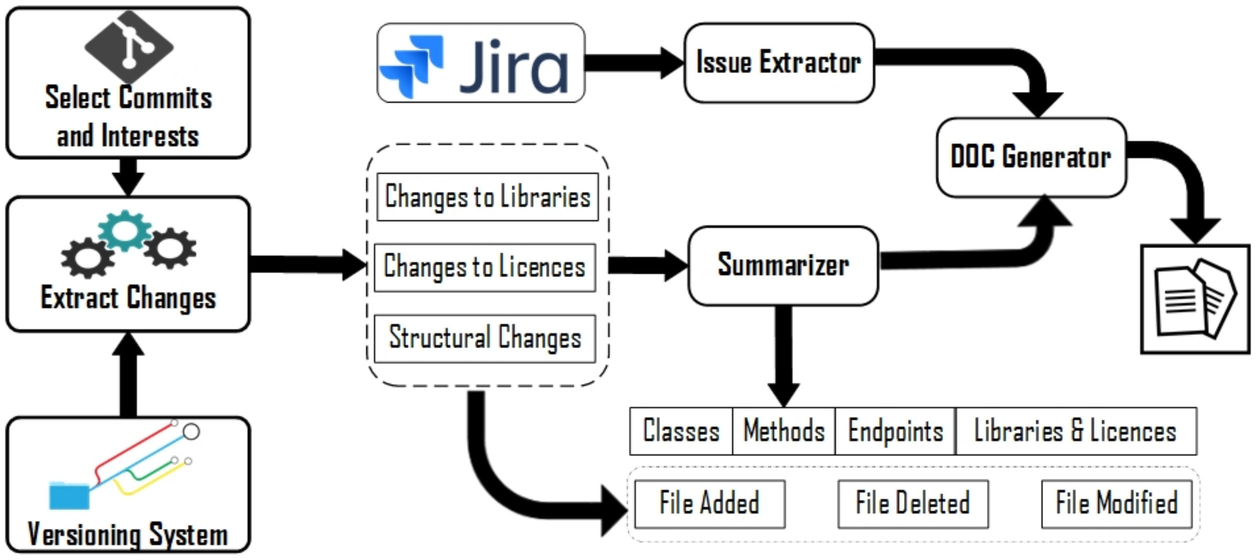


Figure 5. System Architecture

Doc Generator is the last step which processed all the data provided by previous step and organized them in a specific way for final output.

3.2.1. Changes Extractor

The first step of the changes extractor is to select the commits of interest that will be included in the new system. The system prompts the user to enter two dates for the two releases rn-1 (previous release date) and rn (new release date). The system extracts all the commits (c1,…, cn) occurs in these two dates. The last/latest commit cn contains all the changes of the previous commits. The last commit of latest release rn and last commit of previous rn-1 are provided to further processing. The following are the steps of changes extractor.

3.2.1.1. Source Code Analysis (diff of rn& rn-1)

Now we have the commits of interests, the changes extractor extract different type of information from these commits. Git provides diff command that keep track of all the files with their changes. We can iterate through files using diff and different iterator types like "A", "M" etc. All the files are then converted to their respective Abstract Syntax Tree (AST) for extracting changes. The following kinds of changes are extracted:

- **Files added and files removed:** Diff provides added and removed files of the new version (latest commit). We can iterate through diff information for added files with iterator type “A” and “D” for deleted files. It provides us the content of the file, path of the file, author of the file, collaborator of the file etc. We will keep this information for further processing. Next the system search for classes, functions and endpoints from these files and keep these information. AST is the plays a vital role in this extraction. By looping through its nodes we can easily find classes and functions.

- **Files Modified:** ”M” iterator type is used for extracting modified files. Diff provide two versions of all the modified files (one contain content before changes and one contains all the new changes). After converting both versions of all these files to their respective ASTs, the system compares them and extracts all the useful information from these files. The system keeps these information and categorize them into different types of changes lik**e**
    - Newly added classes
    - Newly added functions
    - Newly added endpoints
    - Changed classes
    - Changed functions
    - Changed endpoints
    - Deleted classes
    - Deleted functions
    - Deleted endpoints
- **Classes added, removed and modified:** Now we ASTs and other information about all added, deleted and modified files, the system is able to deep dive and extract classes, functions and endpoints. In modified files, if the class is present in new version and not in previous version it will be marked as new class in modified file. If the class is present in the last version and not in new version it will be marked as deleted file. If the classes is present in both files and any function, parent classes, constructor, setter or getter of the class in the new version is not matching to their respective method in the previous version then it will be marked as changed class in the modified file.
- **Functions added, removed and modified:** If the function is present in the new version and not in the previous version than it will mark as new function. If it is present in last version and not in the new version than it will be marked as deleted function and if the

function is present in the both the files and body of the function is not matching then it will be marked as changed/modified function.

- **Endpoints added, removed and modified: I**n after extracting the endpoint from modified the system check if the endpoint is exists in the new version of file and not in previous version then it will be marked as new. If it is present in previous and not in new changed version of file then it will be marked as deleted. If it is present in both versions of a file and method, URL or body of the callback/middleware changed then it will be marked as changed/modified.

3.2.1.2. Changes to Used Libraries

Libraries in node JS called packages and all these packages are listed in package.json file of the root folder of the project. Package.json file is a simple json (Javascript Object Notaion) file. The packages are listed under dependencies and devDependenceies keys. A library/package needed for node.js project is installed with NPM (Node Package Manager) or yarn (dependency manager for node.js). The system checks if the package.json file is changed in the new version or not with iterator type “M”. If the package.json file is changed in the new version of the project it will compare it with the package.json file of previous version.

3.2.1.3. Changes to License

Package.json file also contains license information. Comparing the package.json file of both versions will provide the change of license, if any.

3.2.2. Summarizer

The output of the changes extractor (changed files, changed classes/functions/endpoints, added classes/functions/endpoints, deleted classes/functions/endpoints, libraries, license) are provided to summarizer. This module will generate Natural Language Summaries in English language for these changes.

How this module generate summary for classes, functions, endpoints are discussed in detail in below sections.

3.2.2.1. Method Summary:

Very few researchers' worked on the automation of summary generation for methods. Some semi-automatic techniques extracted un-commented and prompt the user to enter the summary for that code snippet as their summary [39], [40]. Some of them worked on the generation of comments for the method using high-level abstractions [41].

In Node JS a method can be written in multiple syntax formats i.e. simple functions with function keyword, arrow functions, pure functions etc. It is more complicate to generate summaries for methods in node JS as compare to other OOP (Object Oriented Programming) languages. We have developed an approach inspired from the one proposed by [42]. In this approach, the summary is generated by heuristic rules. **Figure 6** shows how this approach works.

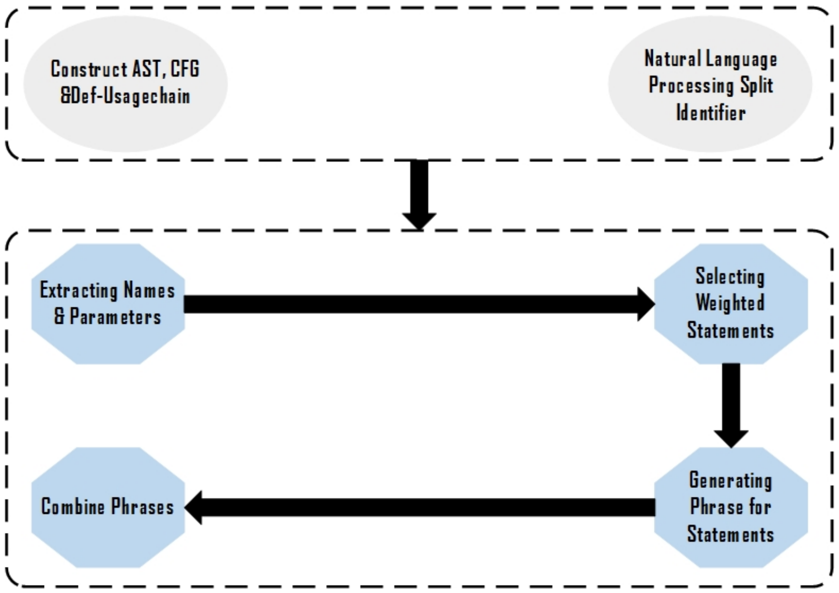

Construct AST, CFG &Def-Usagechain
Natural Language Processing Split Identifier
Extracting Names & Parameters
Selecting Weighted Statements
Generating Phrase for Statements
Combine Phrases

Figure 6. Flow of Natural language Summary Generation

Before all the steps this module constructs AST, Control flow Graph (CFG) and def-use chain for the method. Then the identifier (name of the function) and parameters are extracted next. The following are the steps used in generating summaries for method.

1) Selecting weighted Statements:

As we know that not all the statements are equally contributed to the method. Some of the statements are very important and some are not like socket.emit(data) and console.log(). We need to selected only important statements from the system and we called them weighted statements. [43], [44] Proposed algorithms to extract Word phrases from the signature of methods for different purposes. We did this step with heuristic rules inspired by [45] to select these statements. The following are the conditions/rules for selecting statements:

- **Ending statement:** Ending statements are the exit points of a function. All the ending statements are counted as weighted statements and can be extracted using CFG.
- **Void-Return statements:** The statement which does not return any values to a variable/constant does something important and is selected.
- **Data facilitating statements:** Those statements that return some value and that is stored in a variable and that variable is used in any other weighted statement are selected.
- **Controlling statements:** All the controlling statements which contain any other weighted statements in their body will be considered as weighted statement.

We will select these statements in their original order in which they appeared in the method. Unnecessary information will be omitted from the summary.

Algorithm 1 Rules for generating phrases

```
Result :   Phrase for statement
Name  =  Split(input)
Class   =  getClass(input)
Arguments  =  getArguments(input)
phrase;
if name == 1 &&name == verb; then
    phrase = name + arguments
else
    if  hasTrailingPastParticiple(name); then
        phrase = name
    else
        if  hasLeadingPreposition(name); then
            phrase = class + name
        else
            if  hasLeadingVerb(name); then
                verb = getVerb(name)
            else
                if  objectName(name); then
                    ext = getObject(name)
                end
                if hasPreposition(name); then
                    prep  =  getPrepositions(name)
                    for all prep p; do
                        wb = getWordBeforePrep(name,p)
                        wa = getWordAfterPrep (name,p)
                        phrase = verb + wb + p + wa + class
                    end
                else
                    phrase = verb + ext + class
                end
            end
        end
    end
end
```

Now we have weighted statements, the next step is to generate phrases for all these statements. The first step is to find if it is a Noun Phrase (NP), Verb Phrase (VP) or Preposition Phrase (PP). E. Hill, L. Pollock and K. Vijay-Shanker proposed an algorithms in their research for generating phrases [45]. Our algorithm is similar to that one but with little changes with. The three main steps in this are:

- Split identifiers into space-delimited phrases
- Determine if the name is NP, VP or PP
- Identify the verb, direct object, preposition and indirect object of the verb

In splitting of identifier into space-delimited phrases the system check if the identifier is camel case, Pascal, capital or small i.e. “addToJobs” is camel case, “AddToJob” is Pascal, “addtojob” is small and “ADDTOJOB” is capital identifier. After splitting all of these will be “add to jobs”. In the next step the system will look at the type of phrase it is. **Algorithm 1** shows the rules developed for identifying the type of a phrase. At last it construct the final phrase by adding parameters and phrase.

Finally all the information i.e. phrases, function name, path and parameters are combined to create the summary.

3.2.2.2. Class Summary

Classes contain different properties/variables and functions/methods. The first step is to extract name of the class and parent class (if it is inherited from any class). Other information contains constructor, setters, getters and other methods. All these are methods and summary for these methods are generated in the same way as for method summary module does. At last all these information are combined to create final summary of the class.

3.2.2.3. Endpoints Summary

Endpoints contain URL, method, middleware and other callback functions. Using AST we will extract the method for the endpoint i.e. POST, GET, PUT, DELETE etc and the URL of that specific endpoint. The remaining are only middleware and callback functions. These both are simple functions. The summary for these functions are generated with the same technique used in the method summary module.

Now we have all the information i.e. URL, method, callback summaries, middleware summaries. The module combined them in a specific format for the final summary of the endpoint.

3.2.2.3. Libraries Summary

We have the information about the changed and newly added library (extracted in the changes extractor step). The detail of the library scrapped from npm official website.

3.2.3. Issue Extractor

Issues tracker used in our research is JIRA. This module fetches all the issues from tracker with different filtering options like tags and time. We extracted these issues along with their detail with big fixes and check if the issues reported during this new version time period entered by the user at start.

3.2.4. Doc Generator

Doc generator combines all the summaries created in the previous step and group them in file level (summaries belongs to same file will be group together). Python-docx is a python library that is used to generate word document in python. This module listed all the information under their respective headings and sections.

Chapter 4

Implementation

CHAPTER 4: IMPLEMENTATION

This chapter explain the implementation of the proposed methodology. The Python programming language is utilized to implement the proposed approach. **Section 4.1** explain the projects used to test the system and **section 4.2** gives detail about the experimentation.

4.1. Test Projects

We used two different projects for testing and development purpose. One project is a node.js backend application for a mobile application which contains Rest APIs for the client side application. The DB used in this application is mongoDB. The Express is used as node.js framework for that application.

Another project is a backend application for a health related project. It contains the implantation of web sockets. Mongo DB is used as database for the application and Express as node.js framework.

4.2. Experimentation

The proposed algorithm is implemented on Anaconda. Anaconda is a standard platform for Python data science and is leading in open source innovation for machine learning. Spyder used as an editor to develop and test the code. Spyder is official editor with Anaconda. Spyder is an IDE which is compatible and used with Anaconda for programming in python language. It is available with anaconda distribution. We have tested the proposed algorithm on core i-3 with 8 GB RAM and CPU of 2 GHz.

We have 4 main modules in the proposed algorithms. The main modules of the system are shown in **Figure 1**. The implementation of these modules are listed below.

4.2.1. Changes Extractor

The first and main step in this algorithm is extraction of source code changes of two version. We provide the Uniform Resource Locator (URL) of the target node.js project to our system. The system prompt for two dates. Our system used different libraries of python for the ease. *Gitpython*is one of them which is used to read git repository in python. The added, removed and modified files are extracted by using the above library. We extracted the classes, methods and endpoints from added files and removed files and saved them in an array.

In case of modified files the above mentioned library only give us the full content of the file for both two versions. This is our system job to extract the changed artifacts from these two versions, which is a little complex. We compare the same files of both versions and find whether the changes are made to classes, methods or endpoints. For classes we check if the classes in both files having same name but any/all of the following changed then the class is labeled as modified class.

- The constructor
- Getter methods
- Setter methods
- Properties
- Other methods
- New method

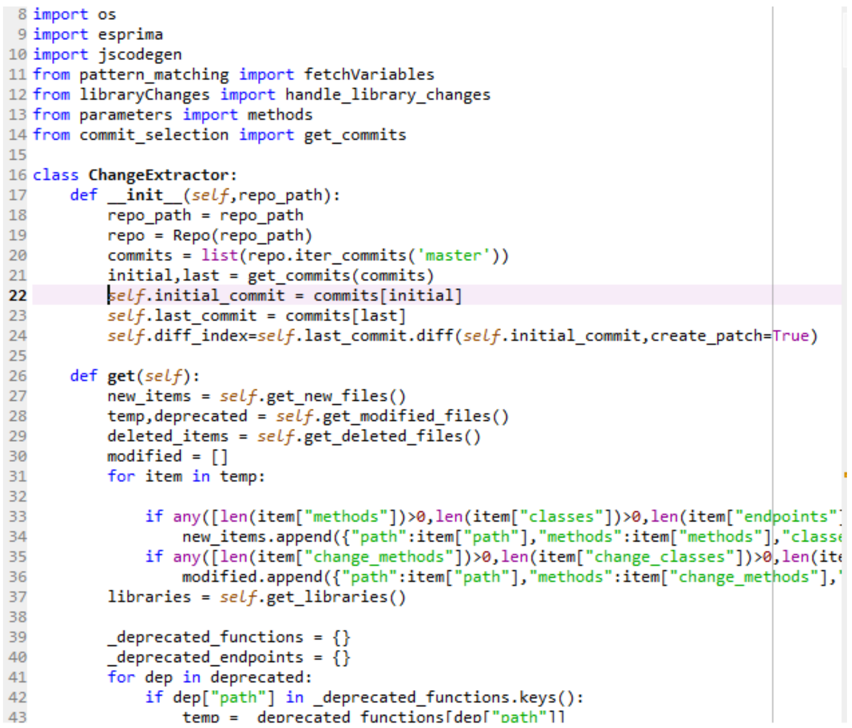
Similarly for methods we checked:

- The list of arguments
- Content of the method (Expressions and Statements)

For endpoints we checked:

- The METHOD type (POST, GET, DELETE, PUT etc.)
- The return data
- The number of middleware used
- The content of all the middleware
- The content of the endpoint (Expressions and Statements)

Finding if any of the above is changed is done with the help of *Esprima*. *Esprima*is at the top of list in these libraries. It is used to generate AST of JavaScript code. **Figure 7** show the code (small part of code) of the module:



```
import os
import esprima
import jscodegen
from pattern_matching import fetchVariables
from libraryChanges import handle_library_changes
from parameters import methods
from commit_selection import get_commits

class ChangeExtractor:
    def __init__(self,repo_path):
        repo_path = repo_path
        repo = Repo(repo_path)
        commits = list(repo.iter_commits('master'))
        initial,last = get_commits(commits)
        self.initial_commit = commits[initial]
        self.last_commit = commits[last]
        self.diff_index=self.last_commit.diff(self.initial_commit,create_patch=True)

    def get(self):
        new_items = self.get_new_files()
        temp,deprecated = self.get_modified_files()
        deleted_items = self.get_deleted_files()
        modified = []
        for item in temp:

            if any([len(item["methods"])>0,len(item["classes"])>0,len(item["endpoints"]
                new_items.append({"path":item["path"],"methods":item["methods"],"classe
            if any([len(item["change_methods"])>0,len(item["change_classes"])>0,len(ite
                modified.append({"path":item["path"],"methods":item["change_methods"],'
        libraries = self.get_libraries()

        _deprecated_functions = {}
        _deprecated_endpoints = {}
        for dep in deprecated:
            if dep["path"] in _deprecated_functions.keys():
                temp =  deprecated functions[dep["path"]]
```

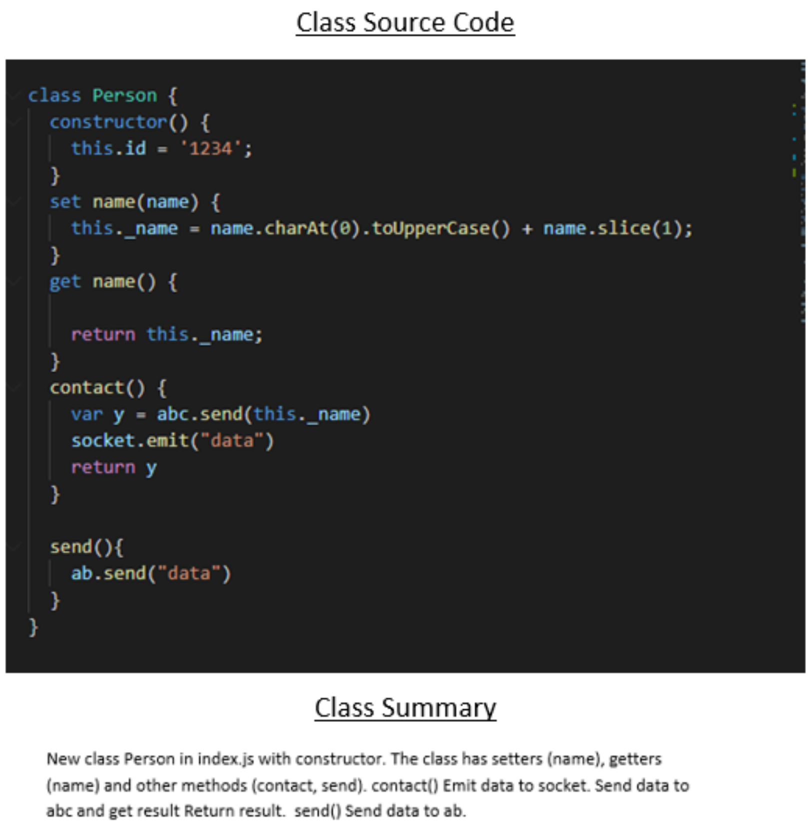
Figure 7. Change Extractor Code

4.2.2. Summarizer

We have different sub-modules that are necessary for this step. We first parse every artifact (class, library change, method, and endpoint) and select the basic information about it. The basic information contains name of class, its methods, constructor etc. After that we feed this data to another sub-module *w-statements.* The job of this module is to extract the weighted statements according to the rules explained in previous section.

These selected statements are then provide to *extract-phrases,* which utilize the summery generator module and create summaries for the selected statements, rearrange these phrases in their original order (order in which they appear in source code) and combine simple phrases. The last task of this module is to combine basic information about the artifact and its summary and return the summary.

We have three main source code items i.e. classes, methods and endpoints. The source code along with their summary are shown in **figure 8,9 and 10**:



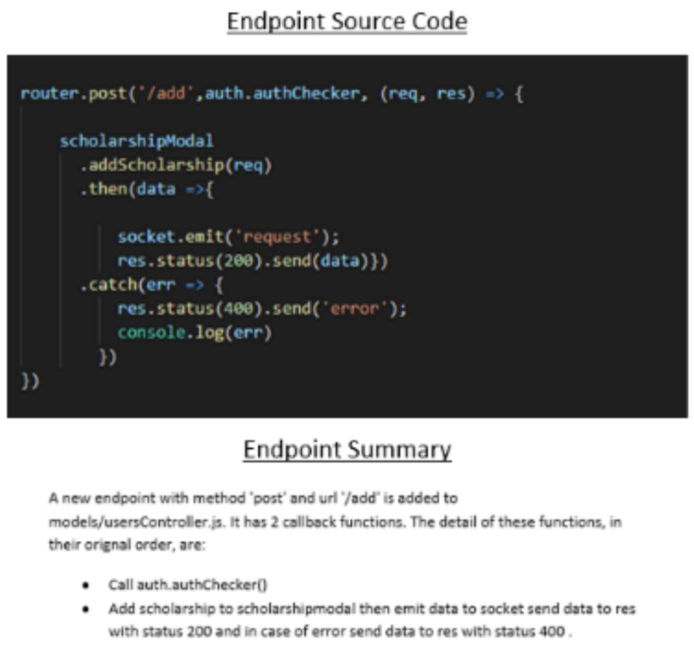
Class Source Code

```
class Person {
  constructor() {
    this.id = '1234';
  }
  set name(name) {
    this._name = name.charAt(0).toUpperCase() + name.slice(1);
  }
  get name() {

    return this._name;
  }
  contact() {
    var y = abc.send(this._name)
    socket.emit("data")
    return y
  }

  send(){
    ab.send("data")
  }
}
```

Class Summary

New class Person in index.js with constructor. The class has setters (name), getters (name) and other methods (contact, send). contact() Emit data to socket. Send data to abc and get result Return result. send() Send data to ab.

Figure 8. Class Source code and our system generated Summary



Endpoint Source Code

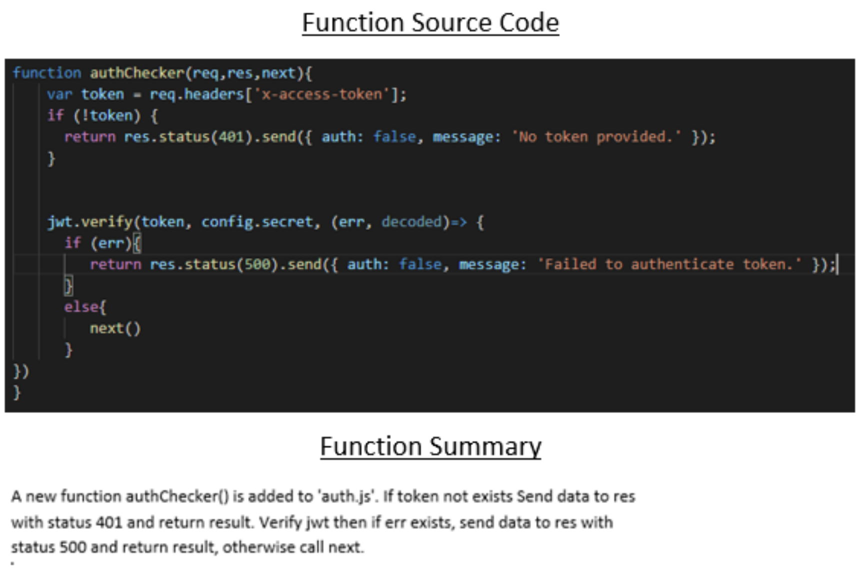
```
router.post('/add',auth.authChecker, (req, res) => {

    scholarshipModal
      .addScholarship(req)
      .then(data =>{

          socket.emit('request');
          res.status(200).send(data)})
      .catch(err => {
          res.status(400).send('error');
          console.log(err)
        })
})
```

Endpoint Summary

A new endpoint with method 'post' and url '/add' is added to models/usersController.js. It has 2 callback functions. The detail of these functions, in their orignal order, are:

- Call auth.authChecker()
- Add scholarship to scholarshipmodal then emit data to socket send data to res with status 200 and in case of error send data to res with status 400 .

Figure 9. Endpoint Source Code and Our System Generated Summary


Function Source Code

```
function authChecker(req,res,next){
    var token = req.headers['x-access-token'];
    if (!token) {
      return res.status(401).send({ auth: false, message: 'No token provided.' });
    }

    jwt.verify(token, config.secret, (err, decoded)=> {
      if (err){
         return res.status(500).send({ auth: false, message: 'Failed to authenticate token.' });
      }
      else{
         next()
      }
})
}
```

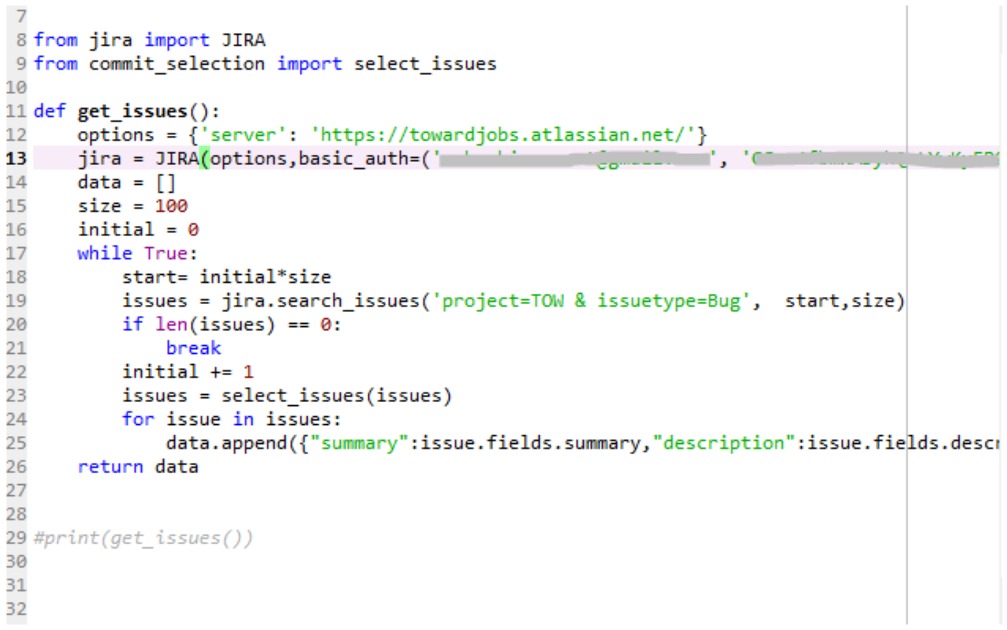
Function Summary

A new function authChecker() is added to 'auth.js'. If token not exists Send data to res with status 401 and return result. Verify jwt then if err exists, send data to res with status 500 and return result, otherwise call next.

Figure 10. Function Source Code and Our System Generated Summary

4.2.3. Issues Extractor

JIRA client library for python i.e. *JIRA* is used to extract the issues registerd in the selected time perios. **Figure 11** shows the code developed for extracting issues.

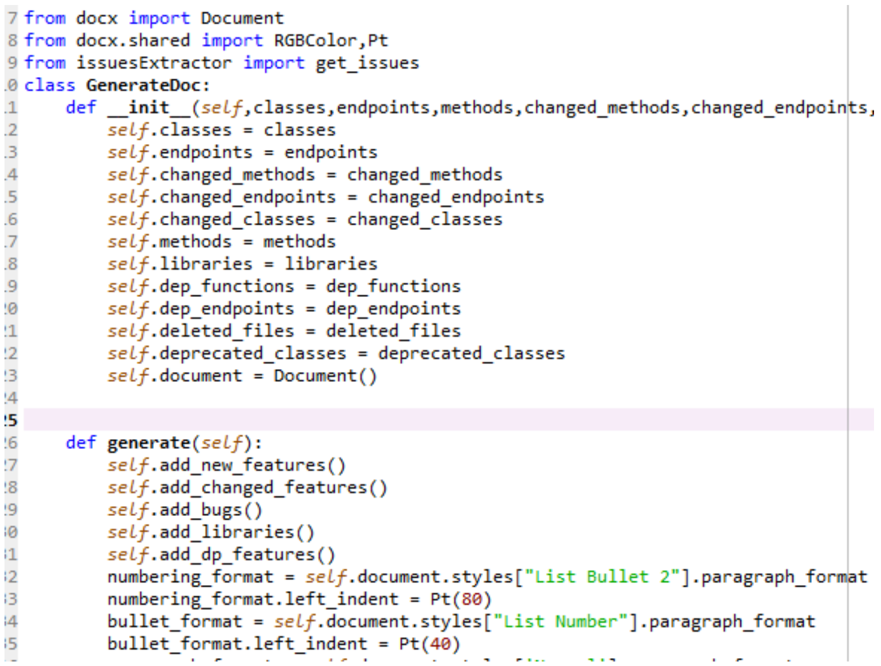


```

from jira import JIRA
from commit_selection import select_issues

def get_issues():
    options = {'server': 'https://towardjobs.atlassian.net/'}
    jira = JIRA(options,basic_auth=('[redacted]', '[redacted]
    data = []
    size = 100
    initial = 0
    while True:
        start= initial*size
        issues = jira.search_issues('project=TOW & issuetype=Bug',  start,size)
        if len(issues) == 0:
            break
        initial += 1
        issues = select_issues(issues)
        for issue in issues:
            data.append({"summary":issue.fields.summary,"description":issue.fields.descr
    return data


#print(get_issues())



```

Figure 11. Issue Extractor Source Code

4.2.4. Doc Generator

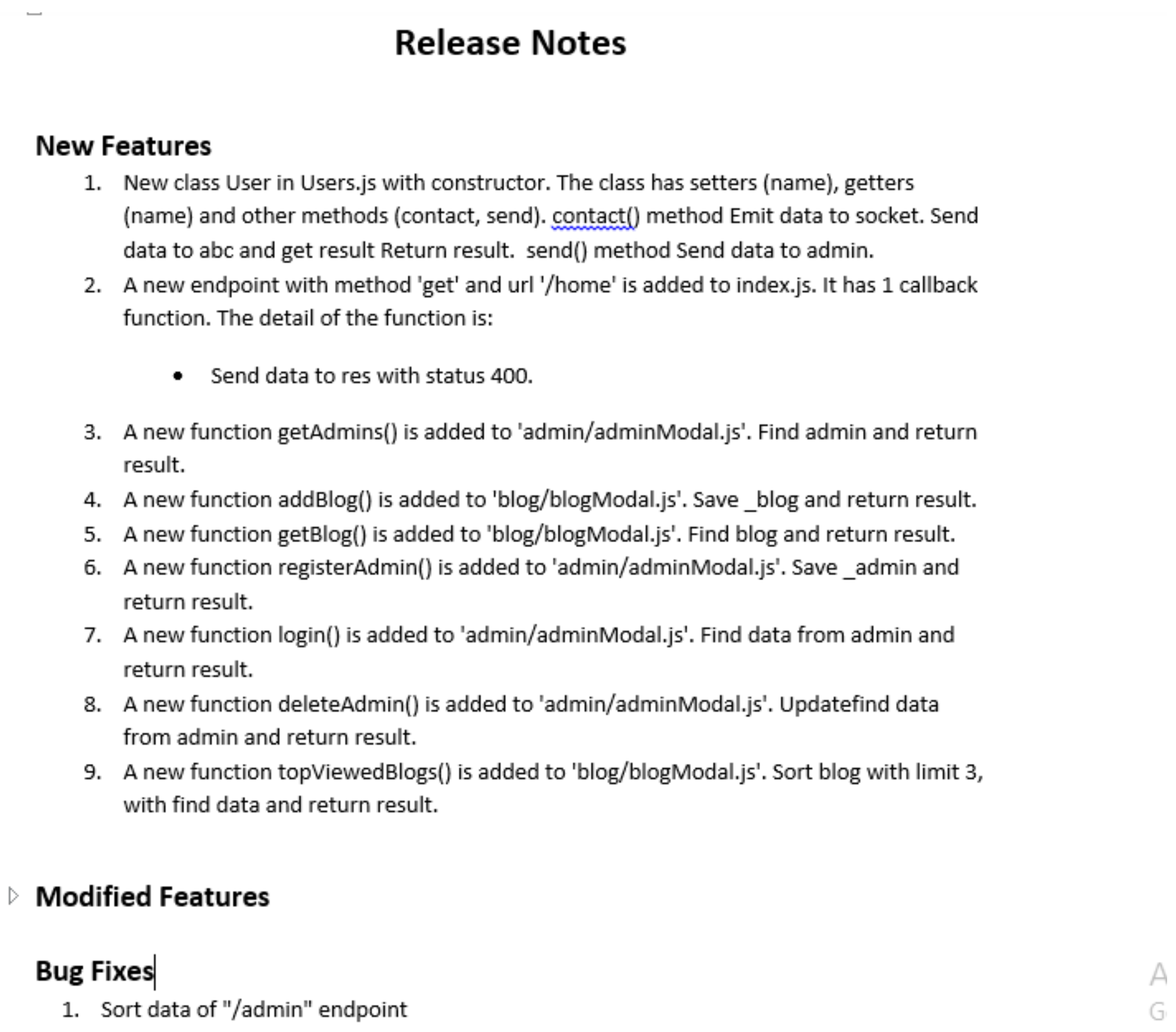
The last step in this method is doc generator. The output of this system is a Microsoft Word file with extension '.docx'. *Python-docx*library is used for generating output. The source code of this module is shown in **Figure 12**:



```
from docx import Document
from docx.shared import RGBColor,Pt
from issuesExtractor import get_issues
class GenerateDoc:
    def __init__(self,classes,endpoints,methods,changed_methods,changed_endpoints,
        self.classes = classes
        self.endpoints = endpoints
        self.changed_methods = changed_methods
        self.changed_endpoints = changed_endpoints
        self.changed_classes = changed_classes
        self.methods = methods
        self.libraries = libraries
        self.dep_functions = dep_functions
        self.dep_endpoints = dep_endpoints
        self.deleted_files = deleted_files
        self.deprecated_classes = deprecated_classes
        self.document = Document()


    def generate(self):
        self.add_new_features()
        self.add_changed_features()
        self.add_bugs()
        self.add_libraries()
        self.add_dp_features()
        numbering_format = self.document.styles["List Bullet 2"].paragraph_format
        numbering_format.left_indent = Pt(80)
        bullet_format = self.document.styles["List Number"].paragraph_format
        bullet_format.left_indent = Pt(40)
```

Figure 12. Doc Generator module Source Code

The output file generated by this system is shown in **Figure 13**:



Release Notes

New Features

1. New class User in Users.js with constructor. The class has setters (name), getters (name) and other methods (contact, send). contact() method Emit data to socket. Send data to abc and get result Return result. send() method Send data to admin.
2. A new endpoint with method 'get' and url '/home' is added to index.js. It has 1 callback function. The detail of the function is:
   - Send data to res with status 400.
3. A new function getAdmins() is added to 'admin/adminModal.js'. Find admin and return result.
4. A new function addBlog() is added to 'blog/blogModal.js'. Save _blog and return result.
5. A new function getBlog() is added to 'blog/blogModal.js'. Find blog and return result.
6. A new function registerAdmin() is added to 'admin/adminModal.js'. Save _admin and return result.
7. A new function login() is added to 'admin/adminModal.js'. Find data from admin and return result.
8. A new function deleteAdmin() is added to 'admin/adminModal.js'. Updatefind data from admin and return result.
9. A new function topViewedBlogs() is added to 'blog/blogModal.js'. Sort blog with limit 3, with find data and return result.

Modified Features

Bug Fixes

1. Sort data of "/admin" endpoint

Figure 13. Our System Output (Release Notes)

Chapter 5

Results Evaluation

CHAPTER 5: RESULTS EVALUATION

This chapter deals with the results and evaluation of our algorithm. **Section 5.1** discusses the evaluation metrics to be used for the evaluation of our algorithm. **Section 5.2** discusses the results and comparison with the previously used techniques. In **section 5.3** the results get from evaluating the results from industry developers.

5.1. Evaluation Metrics

The evaluation metrics for this problem is the same which was used by other researchers in their publications. The generated notes are evaluated manually by software developers in previous papers. So we also used the same method to evaluate our results.

5.2. Results and comparison

Limited people worked and researched on the automation of these notes. There are also few libraries and plug-ins which are used by engineer for this purpose. **Table 3** compares the proposed methodology with the existing algorithms and libraries. The comparison matrices are:

- **Target Language:**Language / Platform for which the corresponding algorithm/tool intended to generate RNs.
- **Category:** This matric states the how they generate RNs. Different categories are:
    - Manual
    - Automatic
    - Semi-Automatic
- **Release Notes Type:**It tells us about which type of information is extracted from the source code. It can be syntactical information (Structural information i.e. functions, classes, files, variables etc), semantically information (The intention of the source code) or both.
- **Release Notes Features:**It means which type of features RNs provided. The different release notes features are:
    - New Features
    - Improvements
    - Bug fixes
    - Deprecated Features

    - License changes
    - Library/Package changes
- **Output:** The output format of the generated notes.
- **Pattern Matching (Extraction):** Whether the algorithm/tool extract certain type of information like Endpoints, DB queries etc.
- **Evaluation:** The evaluation matrices of the generated output.

Table 3. Proposed approach comparison with previous approach

S. No.		1	2	3	4	5	6	7
Algorithm / Tool		ARENA	Semi-Automatic Approach	RENO	Attlassian Release notes Generator	Chroniclor	Modal Based Approach	Proposed
Target Language		JAVA	NA	NA	JAVA	Node JS	Satellite system	Node.JS
Category		Automatic	Semi-Automatic	Semi-Automatic	Automatic	Automatic	Semi-Automatic	Automatic
Release Notes Type		Semantic and Syntactic	Syntactic	Syntactic	Syntactic	Syntactic	Syntactic	Semantic and syntactic
Release Notes Features	**New Features**	✓	✓	✓	✓	✓	✓	✓
	Improvements	✓	✓	✓	✓	✓	✓	✓
	Bug Fixes	✓	✓	✓	×	×	×	✓
	Deprecated Features	✓	×	×	×	✓	×	✓
	License Changes	✓	×	×	×	×	×	✓
	Library/Package changes	×	×	×	×	×	×	✓
Output		HTML file	Plain Text ,Markdown or HTML	Standard format	HTML file	NA	NA	Word file(.docx)
Pattern Matching (Extraction)		No	NO	No	No	NO	No	Yes (End points)
Evaluation		Manually	NA	NA	NA	NA	NA	Manually

5.3. Evaluation

For evaluation we conducted a case study. "TowardJobs" is an android app, developed in React-Native, which shows a list of available jobs in Pakistan. It has different advanced filters for searching jobs. The admin panel for posting and controlling jobs in app is built in React JS.

The backend of both these android app and admin panel is built in node JS. It has 40+ commits at all and 28 new commits in the new version of system. We provide this project to our algorithm and generated the RNs for this version of application. The generate RNs are then evaluated by the industry developers. The main and difficult part in this approach is human language sentence generation for classes, bugs, function and endpoints. We evaluated all these summaries generated by our system individually from different Software Engineers. The details of these engineers involve in study are shown in **Table 4**.

Table 4. Developer's Background

No. of Years	Development experience
0 – 3	9
4 – 6	3
7 – 10	2
Total	**14**

We provide our test project to our system and generate result. After that we develop a questionnaire that contains the source code of different artifacts and their corresponding summaries and ask them to rate their correctness, conciseness and accuracy. The questions are:

1. Your full name
2. Total Experience in software Development
3. Experience in JavaScript or Node JS development
4. Your current or Last Organization

The next three questions are very important and repeated for different types of code artifact (Source code and their natural language summaries). The different code artifacts are Functions, classes and endpoints. These questions are:

5. Accuracy: How much the summary generated by the system is accurate
    - Accurate
    - Slightly Accurate
    - Very Inaccurate
6. Content Adequacy: Check the summary content independent of the way it is presented, what you say it is:

- Adequate
- Misses Some
- Misses Important

7. Conciseness: Check the summary content independent of the way it is presented, what you say it is:
    - Concise
    - Slightly Verbose
    - Very Verbose

The source code of a class is given below:

```
class User {
constructor() {
    this.id = '1234';
  }
set name(name) {
this._name = name.charAt(0).toUpperCase() + name.slice(1);
  }
get name() {
returnthis._name;
  }
contact() {
var y = abc.send(this._name)
socket.emit("data")
return y
  }
send(){
admin.send("data")
  }
 }
```

The summary of the above code is evaluated from developers and the results of their opinion is show in **Table 5**.

Table 5. Class Summary Evaluation

Accuracy		Content Adequacy		Conciseness	
Options	**Result**	**Options**	**Result**	**Options**	**Result**
Accurate	71.4%	Adequate	64.3%	Concise	71.4%
Slightly Inaccurate	28.6%	Misses Some	35.7%	Slightly Verbose	21.4%
Very Inaccurate	0%	Misses Important	0 %	Very Verbose	7.1%

The notes generated by our system has many methods. One of the methods (source code) is given below:

```
functionauthChecker(req,res,next){
var token = req.headers['x-access-token'];
if (!token) {
returnres.status(401).send({ auth: false, message: 'No token provided.' });
  }
jwt.verify(token, config.secret, (err, decoded)=> {
if (err){
returnres.status(500).send({ auth: false, message: 'Failed to authenticate token.' });
    }
else{
next()
    }
})
}
```

The summary of this method is:

"*A new function authChecker() is added to to 'auth.js'. If Token not exists sends data to res with status 401 and return result. Verify jwt then if err exists, send data to res with status 500 and return result, otherwise call next.*"

The evaluation of the summary is shown in **Table 5**:

Table 6. Function's Summary Evaluation

Accuracy		Content Adequacy		Conciseness	
Options	**Result**	**Options**	**Result**	**Options**	**Result**
Accurate	71.4%	Adequate	85.7%	Concise	57.1%
Slightly Inaccurate	28.6%	Misses Some	14.3%	Slightly Verbose	42.9%
Very Inaccurate	0%	Misses Important	0 %	Very Verbose	0%

The source code of an endpoint added to project in new version is given below:

```
router.post('/add',auth.authChecker, (req, res) => {


scholarshipModal
    .addScholarship(req)
    .then(data =>{
res.status(200).send(data)})
    .catch(err => { res.status(400).send('error'); console.log(err) })
})
```

The summary evaluation of the above endpoint is shown in **Table 6**:

Table 7. Endpoint's Summary Evaluation

Accuracy		Content Adequacy		Conciseness	
Options	**Result**	**Options**	**Result**	**Options**	**Result**
Accurate	78.6%	Adequate	64.3%	Concise	57.1%
Slightly Inaccurate	14.3%	Misses Some	28.6%	Slightly Verbose	28.6%
Very Inaccurate	7.3%	Misses Important	7.1 %	Very Verbose	14.3%

The above tables shows that the result generated by our approach are very concise and accurate.

Chapter 6

Discussion and Limitations

CHAPTER 6: DISCUSSION AND LIMITATIONS

This chapter discusses the overview of proposed research and limitations of this research. Section 6.1 contains the Discussion whereas Limitations to research are mentioned in section 6.2.

6.1. Discussion

Automation in software development has revolutionized the process by reducing time and enhancing quality. Natural language processing and programming language processing has helped software engineer in every stage of software development and deployment. Software RNs are important document in software deployment and helps in maintenance and testing. Most of the time these notes are generated manually. The purpose of this research is to automate the generation of these RNs. These notes are very important for a software developer, manager and other stakeholders.

Very few researchers had worked in automation of RNs. All available algorithms automate the generation for java projects. Node.js is widely used for the development of Rest APIs, Micro services, SOAP APIs and other purposes. This approach is a first one to work on Node JS project. We have compared our approach to previous algorithms and achieved very good results. Results are also evaluated from developers.

6.2. Limitations

This is a novel approach for Node JS project. It still needs improvements in its results as Node JS is very complex and divers in terms of its syntax. The approach used heuristic rules to generate natural language summaries for source code items. This step can be done with Deep Learning algorithms. We can do it with neural attention model of machine translation, but the problem is that the data set of node JS source code is not available which limited us to used heuristic approach. To create dataset by own will take a lot of time and more node JS developers.

As we discussed in the above paragraph that the node.js is very diverse and complex. It has different frameworks and libraries. We were unable to cover all the frameworks and libraries/packages at this time. The only framework which is supported by our system is Express.js. It’s only tested on Express Framework.

Chapter 7

Conclusion and Future Work

CHAPTER 7: CONCLUSION AND FUTURE WORK

This chapter concluded the research and discuss the future work. Section 7.1 shows colclusion and 7.3 contains future work.

6.1. Conclusion

Generating release notes manually is a main hurdle in software development now a days. It takes a lot of time and understanding of the code. In continuous delivery its generation is even more complex. Automatically generating release notes save time, improves quality and enhances software development process. Very little literature is available in this context.

This approach is a state of the art technique for Node JS projects. There is no other tool or technique available for Node JS except Chroniclor [27]. The evaluation and case study shows that the proposed approach is very good in its result for node JS projects. It is more accurate, complete and concise than the release notes generated manually. The output of the system is a Word document which is quite easy for a release manager to edit (update/deleted/add) it. This technique is beneficial to software engineers in saving their time and other resources.

6.2. Future Work

In future work we will develop a proper GUI tool to generate release notes. We want to generate output in multiple formats like Microsoft Word, pdf and html. We also want to improve the quality of sentences generated from source code. The summarization is done with heuristic rules. We want to automate the summary generation by using deep learning models like Neural Attention model of Machine Translation and Tree based Convolutional Neural Network.

We can also add different new features to our existing approach like changes made to deployment configuration, information about the authors etc.

References

1. Kagdi, H., M.L. Collard, and J.I. Maletic, *A survey and taxonomy of approaches for mining software repositories in the context of software evolution %J J. Softw. Maint. Evol.* 2007. **19**(2): p. 77-131.
2. Farley, J.H.D., *Continuous delivery: reliable software releases through build, test, and deployment automation. .* 2010.
3. Chen, L., *Continuous Delivery: Huge Benefits, but Challenges Too.* IEEE Software, 2015. **32**.
4. Moreno, L., et al., *ARENA: An Approach for the Automated Generation of Release Notes.* IEEE Transactions on Software Engineering, 2016. **43**: p. 1-1.
5. Abebe, S., N. Ali, and A.E. Hassan, *An empirical study of software release notes.* Empirical Software Engineering, 2015.
6. Khomh, F., et al., *Understanding the impact of rapid releases on software quality: The case of firefox.* Empirical Software Engineering, 2014. **20**.
7. Kemerer, C. and S. Slaughter, *An empirical approach to studying software evolution.* Software Engineering, IEEE Transactions on, 1999. **25**: p. 493-509.
8. Moreno, L., et al., *ARENA: An Approach for the Automated Generation of Release Notes.* IEEE Transactions on Software Engineering, 2017. **43**(2): p. 106-127.
9. Hassan, S., W. Shang, and A.E. Hassan, *An empirical study of emergency updates for top android mobile apps.* Empirical Software Engineering, 2017. **22**(1): p. 505-546.
10. Farley, J.H.D., *Continuous delivery: reliable software releases through build, test, and deployment automation.* 2010.
11. Rodríguez, P., et al., *Continuous deployment of software intensive products and services: A systematic mapping study.* Journal of Systems and Software, 2017. **123**: p. 263-291.
12. Klepper, S., S. Krusche, and B. Brügge. *Semi-automatic Generation of Audience-Specific Release Notes*. in *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*. 2016.
13. Abebe, S.L., N. Ali, and A.E. Hassan, *An empirical study of software release notes %J Empirical Softw. Engg.* 2016. **21**(3): p. 1107-1142.
14. Ba, K. and S. Charters, *Guidelines for performing Systematic Literature Reviews in Software Engineering.* 2007. **2**.

15. Kitchenham, B., et al., *Systematic literature reviews in software engineering – A systematic literature review.* Information and Software Technology, 2009. **51**(1): p. 7-15.

16. Collard, M.L., H.H. Kagdi, and J.I. Maletic. *An XML-based lightweight C++ fact extractor*. in *11th IEEE International Workshop on Program Comprehension, 2003.* 2003.

17. Moreno, L., et al. *Automatic generation of natural language summaries for Java classes*. in *2013 21st International Conference on Program Comprehension (ICPC)*. 2013.

18. Teixeira, J. and H. Karsten, *Managing to release early, often and on time in the OpenStack software ecosystem.* Journal of Internet Services and Applications, 2019. **10**: p. 7.

19. Khalfallah, M., *Generation and Visualization of Release Notes for Systems Engineering Software: Proceedings of the Ninth International Conference on Complex Systems Design & Management, CSD&M Paris 2018*. 2019. p. 133-144.

20. Chandrasekara, C., *Effective Release Notes with TFS Release*. 2017. p. 433-478.

21. Fischer, M., M. Pinzger, and H. Gall. *Populating a Release History Database from version control and bug tracking systems*. in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* 2003.

22. Gall, H., K. Hajek, and M. Jazayeri, *Detection of Logical Coupling Based on Product Release History*. 1998.

23. Iyer, S., et al., *Summarizing Source Code using a Neural Attention Model*. 2016. 2073-2083.

24. Haiduc, S., et al., *On the Use of Automated Text Summarization Techniques for Summarizing Source Code.* 17th Working Conference on Reverse Engineering, 2010.

25. Hu, X., et al., *Summarizing Source Code with Transferred API Knowledge*. 2018. 2269-2275.

26. Ugubi.io. *Release notes generator*. Available from: https://marketplace.atlassian.com/apps/1214154/release-notes-generator?hosting=server&tab=overview.

27. Michael Strickland, A.C., Andrew Fischer, Andrew Canaday. *Chronicler*. Available from: https://github.com/NYTimes/Chronicler.

28. Fennell, R. *Generate Release Notes Build Task (PowerShell).* Available from: https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BMVSTS-GenerateReleaseNotes-Task.

29. Mmentele. *release-notes-generator*. Available from: https://pypi.org/project/release-notes-generator/.

30. Stephan B¨onnemann-Walenta, C.W., Kim Brandwijk, Kepler Sticka-Jones, Pierre Vanduynslager, Matt Travi. *release-notes-generator*. Available from: https://github.com/semantic-release/release-notes-generator.

31. Ltd, A.T.P. *Automated Release Notes for Jira*. Available from: https://marketplace.atlassian.com/apps/1215431/automated-releasenotes-for-jira?hosting=cloud&tab=overview.

32. Kft, M.G.C. *Better PDF Exporter for Jira*. Available from: https://marketplace.atlassian.com/apps/5167/better-pdf-exporter-forjira-pdf-view?hosting=cloud&tab=overview.

33. Zhang, J. *Gitlab Release Note Generator*. Available from: https://github.com/jk1z/gitlab-release-note-generator.

34. Khurana, D., et al., *Natural Language Processing: State of The Art, Current Trends and Challenges.* 2017.

35. Gelbukh, A. *Natural language processing*. in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*. 2005.

36. Kim, J. and Y. Lee, *A Study on Abstract Syntax Tree for Development of a JavaScript Compiler.* International Journal of Grid and Distributed Computing, 2018. **11**: p. 37-48.

37. Gold, R., *Control flow graphs and code coverage.* Applied Mathematics and Computer Science, 2010. **20**: p. 739-749.

38. Kennedy, K., *Use-definition chains with applications.* Computer Languages, 1978. **3**(3): p. 163-179.

39. Sridhara, G., et al., *Towards automatically generating summary comments for Java methods*. 2010. 43-52.

40. Erickson, T., *An automated FORTRAN documenter*. 1982. 40-45.

41. Roach, D., H. Berghel, and J. Talburt, *An interactive source commenter for Prolog programs*. Vol. 14. 1990. 141-145.

42. Moreno, L., et al., *JSummarizer: An automatic generator of natural language summaries for Java classes*. 2013. 230-232.

43. Robillard, P., *Schematic pseudocode for program constructs and its computer automation by SCHEMACODE.* Commun. ACM, 1986. **29**: p. 1072-1089.

44. Maskeri, G., S. Sarkar, and K. Heafield, *Mining Business Topics in Source Code using Latent Dirichlet Allocation*. 2008. 113-120.

45. Ohba, M. and K. Gondow, *Toward mining "concept keywords" from identifiers in large software projects*. Vol. 30. 2005.