

Source code summarization using Natural Language Processing
in C#



Author

Muhammad Qasim Khan

Regn Number

FALL 2016-MS-16(CSE) 00000171740

MS-16 (CSE)

Thesis Supervisor:

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
ISLAMABAD
February, 2020

Source Code Summarization Using Natural Language
Processing in C#

Author

Muhammad Qasim Khan

FALL 2016-MS-16(CSE) 00000171740

A thesis submitted in partial fulfillment of the requirements for the degree of
MS Software Engineering

Thesis Supervisor:

Dr. Wasi Haider Butt

Thesis Supervisor's Signature: _____

DEPARTMENT OF COMPUTE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD
MAY, 2020

DECLARATION

I certify that this research work titled “Source Code Summarization Using Natural Language Processing in C# “is my own work under the supervision of Dr. Wasi Haider Butt. This 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

Muhammad Qasim Khan

FALL 2016-MS-16(CSE) 00000171740

LANGUAGE CORRECTNESS CERTIFICATE

This thesis is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University for MS thesis work.

Signature of Student

Muhammad Qasim Khan

FALL 2016-MS-16(CSE) 00000171740

Signature of Supervisor

COPYRIGHT STATEMENT

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made online accordance with instructions given by the author and lodged in the Library of NUST College of E&ME. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST College of E&ME, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the College of E&ME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST College of E&ME, Rawalpindi.

ACKNOWLEDGEMENTS

I am highly thankful to ALLAH Almighty for his blessings and constant help throughout my thesis. Indeed this would not have been possible without his bountiful and gracious help in each and every step. I am thankful to HIM for putting me on a path where people helped me graciously to achieve my goal. Indeed none is worthy of praise but ALLAH Almighty.

I am profusely grateful to my beloved parents for their constant love, support, prayers and sacrifices. I would like to pay my special thanks to my beloved father **Sher Zaman**, for encouraging me to avail best opportunities in my life. I am also thankful to my sisters and family for all the support and prayers throughout my time of research.

I would also like to express my gratitude to my supervisor, **Dr. Wasi Haider Butt** for his constant motivation, patience, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my MS study.

I would also like to pay special thanks to my Guidance Committee Members **Dr. Arsalan Shaukat and Dr. Urooj Fatima**. Their recommendations are very valued for improvement of the work. I would like to pay special thanks to **Muhammad Waseem Anwar** for his tremendous support and cooperation. Each time I got stuck in something, he came up with the solution. Without his help, I wouldn't have been able to complete my thesis. I appreciate his patience and guidance throughout the whole thesis.

I would also like to thank my family, my brothers, my class fellows and my seniors for their support and cooperation. Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Thanks for all your encouragement!

Dedicated to my beloved parents whose tremendous motivation and support helped me to accomplish this achievement.

ABSTRACT

TABLE OF CONTENTS

COPYRIGHT STATEMENT	iii
ACKNOWLEDGEMENTS	iv
ABSTRACT	vi
TABLE OF CONTENTS	vi
LIST OF FIGURES	viii
LIST OF TABLES	ix
CHAPTER 1: INTRODUCTION	11
1.1. Background Study	11
1.1.1. Code Comment Generation	11
1.1.2. Source code summarization	12
1.1.3. Natural Language Processing	13
1.2. Problem Statement	16
1.3. Proposed Methodology	17
1.4. Research Contribution	18
1.5. Thesis Organization	19
CHAPTER 2: LITERATURE REVIEW	22
2.1. Introduction	22
2.2. Literature Review Methodology	23
2.1.1. Inclusion and Exclusion Criteria.....	23
2.1.2. Search Process	23
2.1.3. Quality Checking	25
2.3. Results	26
2.4. Discussion	29
2.5. Analysis	31
CHAPTER 3: PROPOSED METHODOLOGY	34
3.1. Tools and Techniques Architecture	44
3.2. UML Details	45
CHAPTER 4: IMPLEMENTATION	50
4.1. Architecture of Transformation Engine (MUTE)	50
4.1.1. User Interface.....	51
4.1.2. User interface for output	53
4.2. Transformation Rules	54
4.2.1. Thermal engine example implementation.....	57
4.2.2. Generated Summary	59

CHAPTER 5: VALIDATION	65
5.1. Case Study 1	66
5.1.1. Human Evaluation	72
5.1.1.1. Evaluation of TCS	73
5.1.1.2. Evaluation team	73
5.1.1.3. Evaluation Questions	74
5.1.1.4. Response Discussion	77
CHAPTER 6: DISCUSSION AND LIMITATIONS	84
6.1. Discussion	84
6.2. Limitations	86
CHAPTER 7: CONCLUSION AND FUTURE WORK	88

LIST OF FIGURES

Figure 1.1: Process for generation of source code comments.....	12
Figure 1.2 : Source Code Summarization	13
Figure 1.3: Components of NLP.....	14
Figure 1.4: Flow of Research.....	18
Figure 1.5: Thesis Outline	20
Figure 2.1 : Search Process.....	24
Figure 3.1: Translating Code to a Simplified Summary (TCSS) Architecture	35
Figure 3.2: Translating Code In To Simplified Summary (TCSS) Architecture	36
Figure 3.3: Tokenization of if conditional statement.....	37
Figure 3.4: Programming constructs structuring for Natural Language Processing	39
Figure 3.5: Translating Structure in to Simplified Summary using templates	42
Figure 3.6 : Tools and Techniques Architecture.....	45
Figure 3.8: Class diagram of Description Area	47
Figure 4.1: Transformation Code System.....	50
Figure 4.2: Main Screen of TCS	52
Figure 4.3: Open Interface for TCS	53
Figure 4.4: User Interface for generated summary and code once submit button is clicked	54
Figure 4.5: TCS Output	58
Figure 4.6: Using TCS for SCS	58
Figure 4.7: Input screen for construct with in a construct	61
Figure 4.8: Output screen	62
Figure 4.9: Diagram showing different sections of summary	63
Figure 5.1: Quality Assessment Process of our Generated Source Code Summary	66
Figure 5.2: Class used in the case study	67
Figure 5.3: Screenshot while submitting the code of first case study to our Tool	68
Figure 5.4: Screenshot showing generated summary of case study 1	69
Figure 5.5: Survey	75
Figure 5.6: Responses of the users that participated in survey	77
Figure 5.7: Responses of the users that participated in survey	78
Figure 5.8: Responses of the users that participated in survey	78
Figure 5.9: Responses of the users that participated in survey	79
Figure 5.10: Responses of the users that participated in survey	79
Figure 5.11: Responses of the users that participated in survey	80
Figure 5.12: Responses of the users that participated in survey	80
Figure 5.13: Responses of the users that participated in survey	81

LIST OF TABLES

Table 2.1: Repositories, associated papers and their reference.....	25
Table 2.2: Data Extraction.....	25
Table 2.3: Frameworks/Tools proposed	26
Table 2.4: Language Constructs targeted	27
Table 2.5: Target Programming Languages.....	27
Table 2.6: Input of Code / Comments for Source Code Summarization	28
Table 2.7: Targeted programming constructs	28
Table 2.8: Tools.....	31
Table 2.9: Evaluation Metrics.....	31
Table 3.1: Tokens	38
Table 4.1: Transformation Rules	55
Table 4.2: Templates for constructs and nested constructs of “If” and “For”	56
Table 4.3: Different type of class summaries	60

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

In order to find error in software code, software engineer must investigate and find exact entities of software code which are needed to be changed [1]. Existing studies found that developers spend more time in searching the code rather than editing it[2]. However, there are other activities that are important along with the searching of code like: browsing and reading. Furthermore, these activities are essential for the developer to understand the code in order to maintain or fix it. Moreover, if we want to improve the maintainability of a software system then we have to improve its readability of source code [3]. Sometimes, developers take a quick look to understand the code rather than studying it in detail. Furthermore, if the code is commented well, then developers find it easy to understand the code. Unfortunately, comments are either missing or not good enough to help developers during maintenance. To solve this problem we have presented a solution to this problem but before moving further lets discuss some of the concepts that would be used in this paper.

This chapter includes the introduction of the research study and the background study in **Section 1.1**, whereas, problem has been described in **Section 1.2**. Moreover, proposed methodology has been described in **Section 1.3**, our contribution to research is included in **Section 1.4** and organization of thesis is comprehended in **Section 1.5**.

1.1. Background Study

In this section, we have included the description of some concepts used in this paper. These concepts include:

- Code comment generation
- Source code summarization

1.1.1. Code Comment Generation

Code comment generation is to generate human readable comments for source code. Code comments are also called as Program Annotation that is human readable explanations of the written source code. Furthermore, they are very useful for developers during maintenance phase [4]. Code comments are very important for software maintenance as good comments can improve readability of a program [5]. Therefore, the fact that code comments play an important role in software maintenance and development has been widely acknowledged by the researchers[6][7][8]. Although, code comments not only comprehend a source code but also inform about the intension and goals of programmer behind

developed software system [9], writing good comments is a time-consuming activity. Hence, many activities have been carried out to generate good comments automatically. Automatic code comment generation is really helpful in facilitating program comprehension [10, 11]. Automatic Code comment generation consists of three parts. In first part, data is prepared for the commenting system. Second part represents source code. Furthermore, it also covers structure and semantics of source code which contains information of structure, lexis, grammar, semantics, contexts, invocation relation and data dependency of source code. Finally, as shown Figure 1.1 in third part, text is generated which are natural language sentences which are extracted from information from source code.

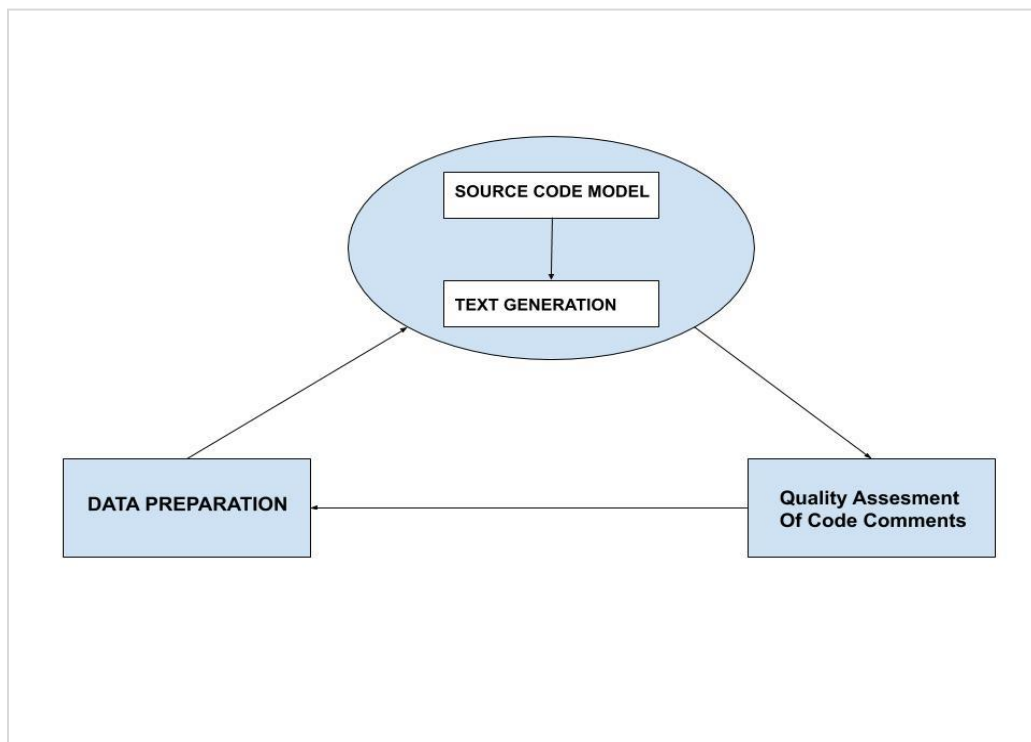


Figure 1.1: Process for generation of source code comments

1.1.2. Source code summarization

Maintaining code and re usage of source code is challenging now a days[12]. There are many practices used to solve this problem. One of them is to generate short summaries of source code .Source code summarization (SCS) is a task to create summaries that are readable and they describe functionality of software .Moreover, it is an important component of software documentation generation; for example, Java docs attaches short description of each method in Java program [13].Furthermore, source code summarization could be done by targeting different code constructs. Researchers have targeted different constructs to get their desired results which have been discussed in the Literature review section.

However, in the end, the target is to provide summary of the code in a way which would help programmers in understanding of the code. Source code summarization is measured by measuring complexity, conciseness and correctness of the summary generated by the SCS tool. Furthermore, researchers have focused on automatic generation of the source code summary.

The main task of Source Code Summarization is to provide readable summaries which can help in understanding of the source code for efficient maintenance. Furthermore, the input of SCS is source code which upon usage of different kinds of techniques that is as following used in the last ten years:

- Natural Language Processing
- Machine Learning
- Neural networks
- Attention mechanism

Moreover, combinations of these techniques are also used for SCS which later is discussed in the literature review section. As shown in [Figure 1.2](#) the overall process in generalized form consists of three sections totally. Which from starting, source code is taken as input in start and processing has been done to generate the summary in final step.

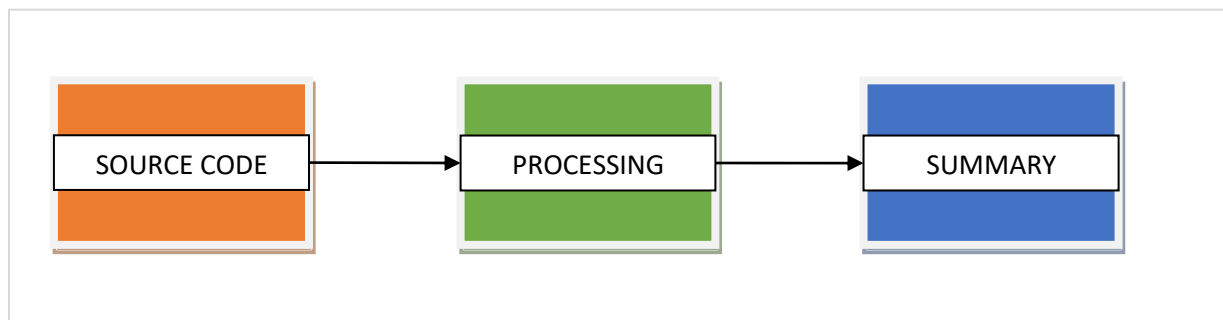


Figure 1.2 : Source Code Summarization

1.1.3. Natural Language Processing

Human beings are the most advanced species on earth and there is no doubt in that .Moreover, our success as human beings is because of our ability to communicate and share information .That is where the concept of developing a language comes in. Furthermore, when we talk about the human language, it is one of the most diverse and complex part of us considering a total of six thousand and five hundred languages that exists. So, in 21st century, according to the industry estimates, only 21% of the available

data is present in the structured form. Moreover, data has being generated as we speak over calls, send messages and write tweets or statuses on different social networks. Majority of this data is present in textual form which is highly unstructured in nature. In order to produce significant and actionable insights from this data it is important to understand with the techniques of text analysis and natural language processing. Natural Language Processing is a major area of artificial intelligence [14] and it is a process to derive meaningful information from natural text[15]. Natural language processing is carried out with the assistance of other knowledge areas too like : Deep learning [16], attention mechanism [17],neural networks [18] and machine learning [19]. Moreover, this technique brings closer machine to human being because the machine becomes more human by using this technique [20]. It has various applications like: usage in fields of machine translation, email spam detection, information extraction, summarization, medical, and question answering etc.[21].Generally, it usually involves the process of structuring text, deriving patterns within the structured data and finally evaluating and interpreting the output.

The major components of NLP shown in [Figure 1.3](#)as follows:

1. Natural Language Understanding
2. Natural Language Generation

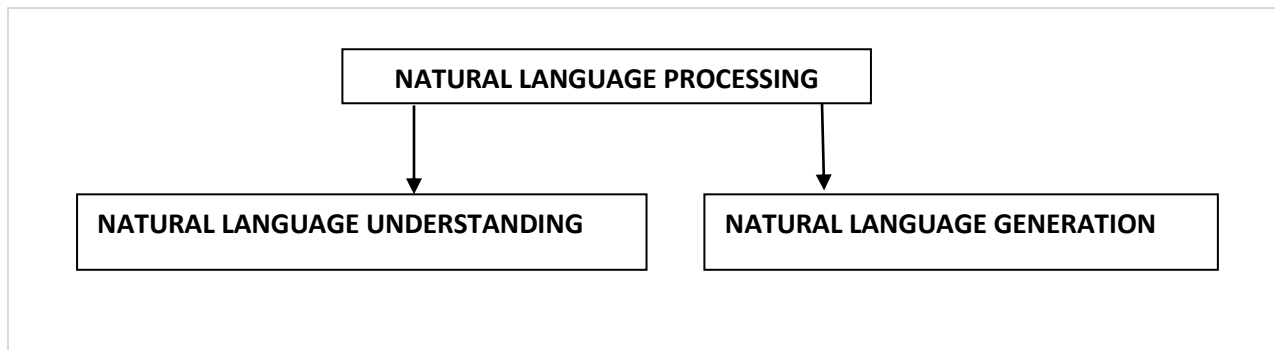


Figure 1.3: Components of NLP

Natural Language Understanding generally refers to mapping the given input into language, performing useful representation of that language and analyzing those aspects of language. On the other hand, Natural Language Generation of producing meaningful phrases and sentences in the form of Natural Language from some internal representation . Generally, Natural Language Understanding is more complex than Natural Language Generation.

There are various steps involved in Natural Language Processing which are as follows:

1. Tokenization
2. Stemming
3. Lemmatization
4. Generating POS Tags
5. Named entity recognition
6. Chunking

Tokenization is the first step of NLP. It is the process operating strings into tokens which are small structures or units that can be used for tokenization. The second step of NLP is stemming. It includes normalization of the words into its base form or root form. The third step of NLP is Lemmatization. It groups together different inflected forms of a word called lemma. It is somehow similar to stemming which maps several words into one common root. However, the major difference between stemming and lemmatization is that the out of lemmatization is a proper word. Fourth step of NLP is POS Tags. POS tags show the grammatical type of the word or it separates the parts of speech and identifies its grammar. A word can have more than one part of speech based on the context it is used. The next step of NLP is Named Entity Recognition in which it identifies the names of entities such as person names, company names etc. The final step of NLP is Chunking. In this step, all the extracted information is gathered into bigger pieces and meaningful statements or information. These bigger pieces are also known as chunks. In the context of NLP, grouping of tokens into chunks is called chunking. Furthermore, it also refers to the artificial intelligence of communicating with an intelligent system using the natural language. Additionally; its goal is to find high quality information from the given text. Natural Language Processing and Text mining go hand in hand as parallel to each other sometimes.

One of the goals of the big data analytics is to make the most out of the textual data with access to standard Natural Languages Processing are good but deep learning techniques can improve the text learning and text generation. These methods typically rely on a “language model”. A model that estimates the likelihood of seeing a particular component in a body of natural language. One example is the trigram model which attempts to calculate probability of seeing a pacific sequence of three words in a natural language corpus. Furthermore, all these methods perform well in practice but each one have some kind of limitation. Language is subjective and ambiguous i.e. sometimes the same word can mean different depending upon the context. Moreover, sometimes synonyms can have subtly different meanings depending on the way they are used. It can also be difficult to add new words to an existing language model. So, NLP often requires a lot of manual curation. This added labor comes at the cost of

variable quality and consistency. However, there are some ways like deep learning to overcome the limitations of NLP. For example, Google 1T Corpus has a vocabulary of over 13 million words.

The first and the most important application of natural language processing is sentiment analysis. Another natural language processing application is on speech recognition software to find the meaning from conversations of people or audio data of videos. Moreover, it has been also applied in translation of finding relevant and high quality information from source code by providing meaningful summaries.

Machine translation is also another use case of NLP and the most common example is Google Translate which in real time uses NLP to translate data from one language into another one. Another application of NLP is spell checking keyword and also extracting information from any document or any website. Information extraction is heavily used in Source code summarization by using NLP.

Today, NLP is used in many applications to solve some business problems. One of them is “Spam detection”. Spam detection is to detect a spam comment or spam like advertisements in website. When a spam is detected the user is given options to hide in order to focus on the part where he is interested. Spam detection using NLP helps improving the user experience on websites.

Another application of NLP is autocorrect feature which is largely used in computer software’s and mobile software. So, NLP has been used in different applications today and is a significant area of research for research during the past decade.

1.2. Problem Statement

As discussed earlier in this chapter that Source code summarization have significantly gained the attention of researchers and practitioners around the world. Software systems are used in many fields of life now a days; for example PE teaching and human movement science [22] not only this but the fact is that software are used in every digital device and real world departments i.e. Health Care, Automotive Systems etc., have increased their maintenance. Maintenance of software systems has become more significant task in business world too and now it is very important for a software to be maintainable [23]. As time is referred as money in business [24]; therefore, source code summarization can help programmers to focus on the real task rather than spending more time in understanding of the code. Due to complex design of software system, maintenance involves difficult technical complications if the code is not well commented. Many researchers have worked on analyzing the techniques used on generation of source code comments [5, 25]. Importance of code comment urges us to perform it automatically to help engineers as much as we can. Automatic source code summarization will save enormous amounts

of cost, development time and maintenance time. Automatic SCS provides us the capability to understand code in lesser time, therefore, it has been researched frequently during the last decade in domain of software maintenance system.

As discussed in **Section 2.2 and 2.3**, in the domain of software maintenance and SCS system researchers have proposed solutions to automatically generate source code summary of source at higher abstraction level. SCS provides the ability to create summaries to represent source code functionality which can later be used for software maintenance with minimum effort. Hence it concludes that the source code summary, which includes information regarding classes, methods and other constructs can help programmer to maintain program faster. Some researchers have generated summaries for classes and some have generated summaries for methods. Consequently, researchers have been trying to explain the functionality of software in by targeting different constructs. This creates a gap in which researchers haven't been targeting the constructs in such a way that the logic of the method could be understood, as summaries generate automatically with different methodologies used by them. However, Researchers haven't been targeting nested constructs which are a part of source code and can't be ignored. Therefore, there is a dire need to provide a better and detailed summary of source code level that is generated by keeping in view important constructs like classes and methods in such a way that nested constructs like if statements and for loops are not ignored. So, programmers can have a better understanding of the source code to fill this research gap.

1.3. Proposed Methodology

Our research has been done in systematic way. We have shown each step of our research in [Figure 1.4](#). Problem has been identified in the first step. In second step we have identified the ideal solution of the problem shown in first step of research. Furthermore, comprehensive and detail literature review was carried out by us which helped us to identify the optimal solution for the problem. Review of the research related to our proposed solution was carried out by us, analyzed and compared it. We also performed a comparative analysis of source code summarization techniques and frameworks.

The proposed solution includes an automatic source code summarization in c# language targeting constructs like loops, if statements, functions. It also includes detailed summary which identifies the functions of classes and explains it in simple language. Furthermore, it explains the logic of function by explaining the usage of loops and conditional statement. The summary has been generated using natural language processing. Three case studies has validated the proposed methodology.

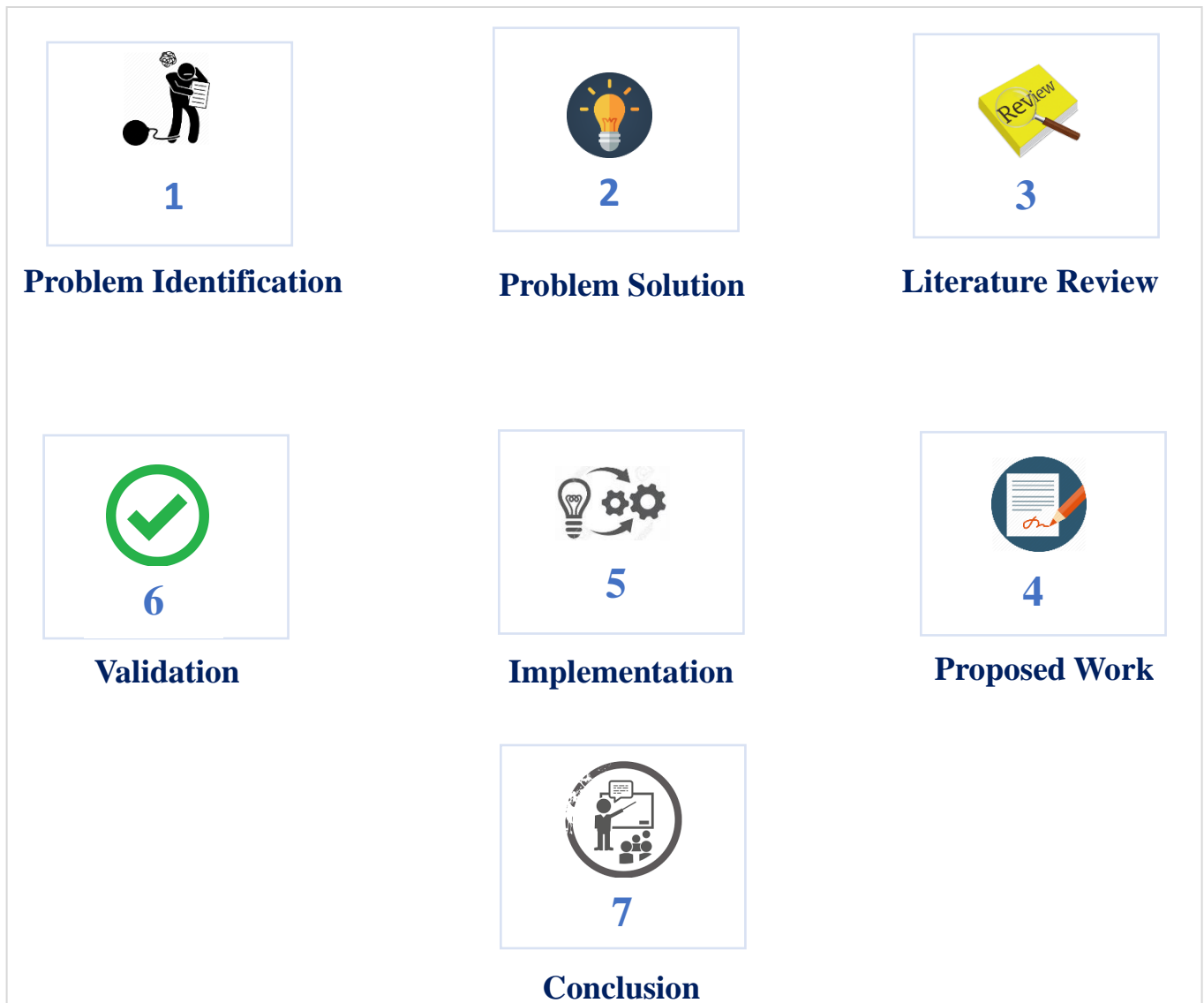


Figure 1.4: Flow of Research

1.4. Research Contribution

Detailed set of contributions of the proposed approach are as follows:

- We have presented new methodology for Natural Language Processing of Code which can generate source code summary for source code. It helps to provide a higher-level description of the source code and reduces the time in understanding of code.
- We have used the core concepts of Natural Language Processing to propose new methodology for SCS in C#.
- We have validated our proposed methodology by using three different benchmark case studies

1.5. Thesis Organization

Thesis organization can be viewed in [Figure 1.5](#). In **Chapter 1: INTRODUCTION**, the introduction consists of the background study of the concepts used in this research, problem statement, and proposed methodology, contribution of research and organization of thesis. **Chapter 2: LITERATURE REVIEW** discusses the previous contributions in domain of embedded systems in the context of Source code summarization. We went through two steps to perform Literature Review. Firstly, the research articles are evaluated to find SCS-based research in domain of Source code summarization. Secondly, multiple research articles from the last decade were studied in detail to extract data from each research article. Finally, research gap has been identified from the literature review. **Chapter 3: PROPOSED METHODOLOGY** narrates the proposed methodology for the problem identified where the summary of source code using natural language processing is performed and the results are shown. **Chapter 4: IMPLEMENTATION** covers the detailed description of our methodology using Natural Language Processing. **Chapter 5: VALIDATION** ensures authentication of proposed framework by usage of 3 case studies, i.e. 1) Car Engine 2) Automatic Lift 3) Social media application. During validation, Source Code Summary of these case studies is generated, then these summaries are checked by different programmers via survey. **Chapter 6: DISCUSSION AND LIMITATION** consists of a brief discussion on the work done by us and its limitations. **Chapter 7: CONCLUSION AND FUTURE WORK** concludes our research and recommends a future work for the researchers which could play a vital role in advancement of science.

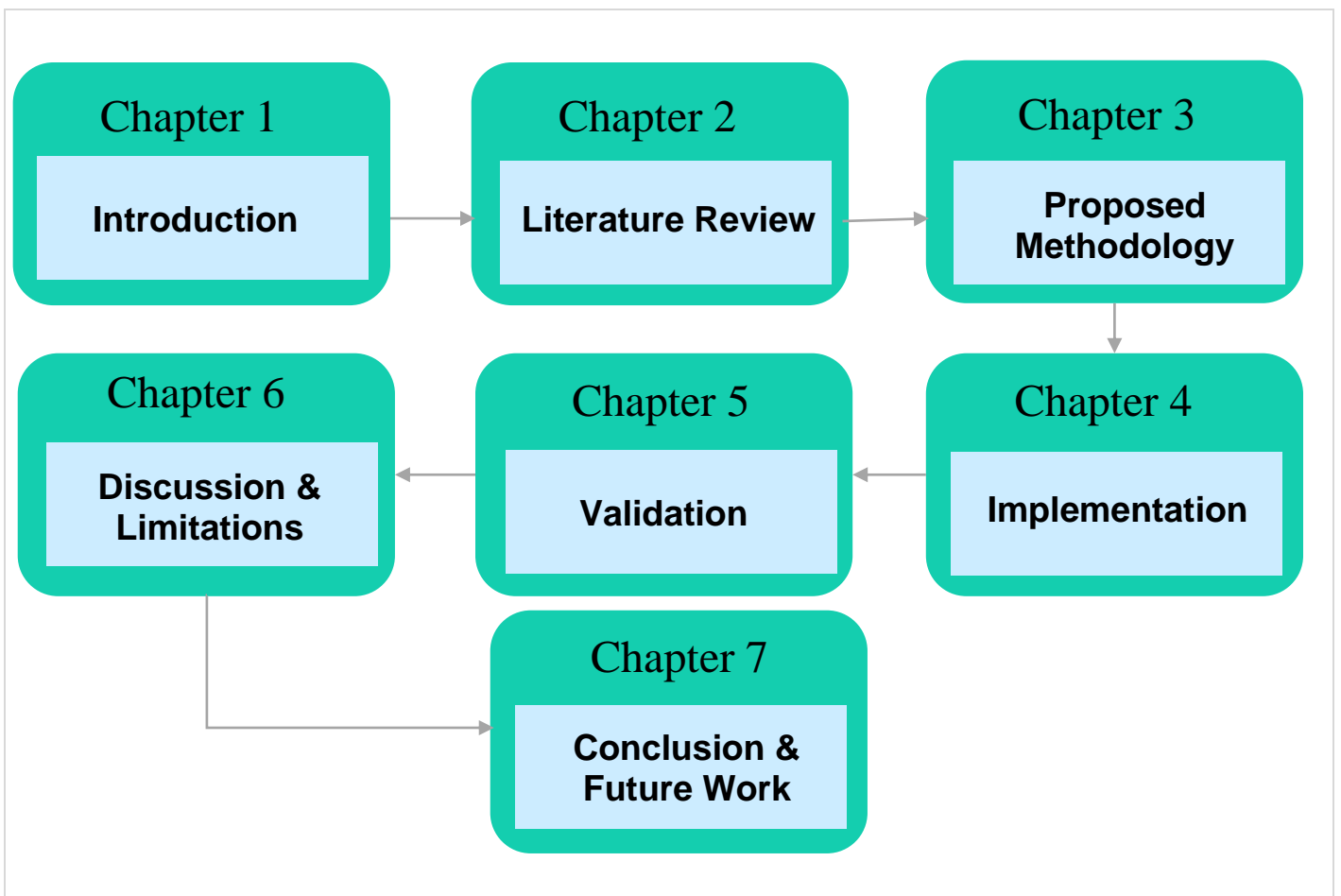


Figure 1.5: Thesis Outline

Chapter 2

Literature Review

CHAPTER 2: LITERATURE REVIEW

In software development and software maintenance phase, the key to success is comprehension of code. Moreover, source code is the primary source for finding information about systems[26]. Even open source software systems maintenance is also a vital research topic for researchers [27]. Not only this but software maintenance is also important for software evolution too [28]. Source code summary is crucial to enhance the understanding of code. Therefore, Source Code Summarization (SCS) is getting recognition among researchers and programmers during the past few years due to its ability to improve understandability of code. So, many researchers have proposed and developed different tools in past decade for helping maintenance programmers and maintenance engineers for improvement of maintenance phase of Software Engineering. Automatic Source code summarization has been performed using different techniques and has been applied on various programming languages. So, there is strong need to summarize the latest advancements in SCS. Therefore, this article contains a Systematic Literature Review (SLR) which identifies 22 research papers (i.e.2010-2020) either performing source code summarization or suggesting state-of-the-art methodologies. Furthermore, 16 implemented methodologies and 5 proposed ones are identified. Moreover, some advantages of source code summarization have been identified. Finally, a comparative analysis of tools developed, is performed. In a nutshell, SCS provides a facility for programmers to understand the written code well.

2.1. Introduction

Developer analyses the code for software evolution process, for doing so, they must understand the code faster[29]. It takes more time if it is done manually than by using automated source code summarization techniques [30]. Source code summarization is a technique used to generate natural language summaries of source code. Although, software maintenance could be performed faster if developer pays more attention towards editing the code, they spend more time reading the code[31]. Furthermore, this happens because good comments are missing in the code. Source code summaries mostly depend on high quality identifiers and method's identifiers due dependency on natural language processing [32]. Early researches have been using techniques Natural language processing, machine learning, neural networks etc. for the development of code summaries.

Code comments generation is very important and challenging topic among researchers now a days due to its importance in maintenance of software[5]. Automatic source code summarization is a process in which human readable code comments or code summaries are generated automatically using various

techniques. Meanwhile, such a description should not only be covering the description of the program code but also the intent of the developer behind code. For Automatic SCS, many tools have been proposed or developed by the researchers in the past decade (2010-2020).

Following are the research questions suggested for this article:

- What are the significant frameworks/methodologies developed / proposed for SCS from 2010 to 2020?
- What are the approaches used by researchers for SCS from 2010 to 2020?
- Which programming constructs were targeted by the researchers for SCS during 2010 to 2020?
- Which programming languages were targeted for the generation of SCS?
- What are the significant metrics required to measure the quality of Summaries?

We have performed Systematic literature review to give quantitative answers of the above research questions. For this purpose, 22 research papers are selected, which are published from 2010 to 2020 in 4 significant libraries that are: IEEE, SPRINGER, ACM AND ELSEVIER .Along with that, analysis has been performed on the tools developed by researchers. It is concluded that SCS helps in reducing efforts of software maintenance engineers by giving them understanding of the code in reduced time.

2.2. Literature Review Methodology

2.1.1. Inclusion and Exclusion Criteria

In order to optimize paper selection procedure, six constraints are defined. On the basis of these constraints, papers will be either selected or rejected. (1) A research paper will only be selected if it is in the scope of research perspective. Only those papers will be selected which will either be suggesting a tool / methodology or a framework to automatically generate the summary of source code. (2) Research papers published between 2010 and 2020 would be selected only. (3) Research papers published in IEEE, ACM, ELSEVIER and SPRINGER will be selected only. (4) Research papers must provide solid results in context of source code summary generation.

2.1.2. Search Process

For finding research papers we searched SPRINGER, ACM, IEEE and ELSEVIER because these scientific libraries are widely accepted. The different search items like Source code summarization, code

summary, comment generation and automatic source code summarization system etc. were given to these libraries to find conference proceedings and journal papers. Furthermore, we used AND/OR operators on all searches where search terms were contained more than one word i.e. (((“comment summary”: generation) AND “source code summarization”: generation) AND “All Metadata”: Methodology). On the other hand, AND/OR operator does not make it sure to give right results. Hence, to get research articles that were published from 2010 to 2020, a year filter “2010-2020” was also applied during all searches. [Figure 2.1](#) explains the search process.

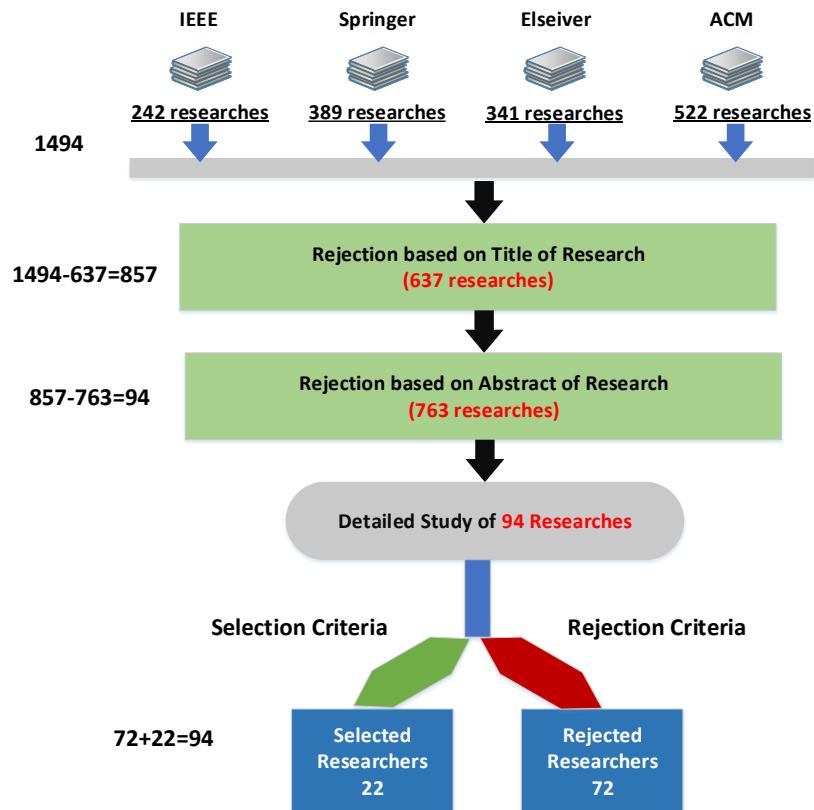


Figure 2.1 : Search Process

(1) We used different keywords like “source code summarization”, “automated comment generation” etc. during our research in each of the online scientific libraries, 1494 search results were generated upon using the above mentioned keywords. (2) 637 search results are dropped because their titles were irrelevant to our research area. (3) We rejected 763 research papers on the basis of abstract analysis. (4) We studied 94 research papers in detail. (5) Finally, we selected 22 research papers on the basis of our inclusion and exclusion criteria and rejected 73 search results

2.1.3. Quality Checking

As a matter of fact, assessment of Quality in research articles is necessary because result of our research is highly dependent on the results of nominated research papers. Therefore, we strictly followed developed principles, to make sure that the impact of our study leaves a better mark on scientific society. These quality principles consist of the following points.

- 1) To achieve good results; we selected most recognized scientific databases. As, they ensure the selected research papers are reliable
- 2) We try to select latest research articles, to ensure the quality of nominated research articles.

Table 2.1 provides summary of number of selected research papers from associated libraries and their reference numbers. We extracted information from finalized research articles on the basis of our exclusion and inclusion criteria. For better study, we performed Data Synthesis on our selected research papers. An overview of data extraction and data synthesis of selected research articles is shown in Table 2.2

Table 2.1: Repositories, associated papers and their reference

Sr#	Library	Type	Ref	Total
1.	ELSEVIER	Journal	[33]	1
		Conference		
2.	ACM	Journal		2
		Conference	[34, 35]	
3.	SPRINGER	Journal	[36, 37]	2
		Conference		
4.	IEEE	Journal	[13, 38]	2
		Conference	[2, 30, 32, 39-50]	

Table 2.2: Data Extraction

Sr#	Parameters	Details
1.	Bibliographic Information	Extracted information includes Author, title, publishing year, details of publisher, research type
Data Extraction		
2.	Overview	General overview of research
3.	Results	Found results during research
Data Synthesis		
4.	Approaches	Identified techniques used by researchers
5.	Tools	Found different tools implemented by researchers
6.	Benefits	Identify Benefits of SCS
7.	Languages	SCS of different programming languages

2.3. Results

Table 2.3: Frameworks/Tools proposed

Sr#	Frameworks /Methodology	Approaches	Data Set		Ref
			Description	Availability	
1.	CloCom	NLP(Code clone techniques)	1005 Open source java Projects , 42 million LOC	Open Source	[42]
2.	JSummarizer	NLP	N/A	N/A	[45]
3.	Portfolio	NLP and Spreading activation network (SAN) algorithms	N/A	N/A	[46]
4.	CodeAttention	Attention mechanism	1600 Java Projects	Open source	[36]
5.	Minipar	latent semantic indexing (LSI) ,Clustering , NLP	NA	NA	[40]
6.	CrowdSummariser	NLP, Crowd sourcing	11 Java applications	Open source	[38]
7.	Using abstract syntax tree and Actor-critic network	Deep reinforcement learning	N/A	Open source	[34]
8.	Comment’s generation using natural language processing and CCGs	NLP , CCGs	Self-created program	Proprietary	[35]
9.	ContextCC	Neural networks, NLP	N/A	Open source	[33]
10.	Source code summarization of java methods	NLP	Six java projects	Open source	[13]
11.	Hybrid Deep-com	NLP , Deep neural network, Attention mechanism	588,108 (methods and comments)	Open source	[37]
12.	Ast -attendgru	Neural models, Neural networks	2.1 m Java methods	Open source	[48]
13.	MethodMan	Pre-defined templates	N/A	Open Source	[44]
14.	Functional topic summary extractor	Topic Mining ,Relevance Calculation, LexRank summarization algorithm	N/A	Open source	[47]
15.	Method level text summarization of java code using nano-patterns	Nano-patterns	Six Java projects	Open Source	[43]
16.	TASSAL (Tree-based Autofolding Software Summarization Algorithm)	NLP	6 Java Projects	Open source	[50]
17.	EBSCS (Entity based source code summarization)	NLP	N/A	N/A	[30]
18.	Tree2Seq (Tree-to-sequence)	AST-based encoder	N/A	Open source	[39]
19.	Combination of LSI (Latent Semantic Indexing) and (Vector Space Model)	Latent Semantic Indexing Vector Space Model	Two Projects From Source Forge	Open Source	[49]

This section explains the results extracted from selected research articles according to research methodology defined in Section 2.

22 research articles written in past 10 years are analyzed to identify 19 different techniques to automatically generate the summary of source code. Table 2.3 describes the tools/methodologies/frameworks either developed or proposed for generating SCS. It also describes the approaches used by researchers in their methodology.

Different data sets were used by researchers for experimentation of their proposed methodologies. However, few articles have not mentioned their validation data sets. So, details of data sets and their availability are also mentioned in Table 2.3. These data sets mostly contained java methods. Some researchers used more than 1000 Java methods in their data sets. Also, some Data sets included comments and code as input to their proposed tool for SCS. However, one of the proposed methodology generated summary from summaries written by people on internet.

Table 2.4: Language Constructs targeted

Sr#	Language Construct	Ref
1.	Method	[13, 30, 32-35, 37-40, 42-44, 46-48, 50]
2.	Conditional Statements	[30, 32, 34, 35, 37-39, 42, 43, 45, 50]
3.	Classes	[2, 30, 37, 39-42, 45, 47]
4.	Iterative statements	[35, 37, 38, 43]

Table 2.5: Target Programming Languages

Sr#	Language	Ref
1.	Java	[2, 13, 32, 33, 35-39, 41-43, 45-48, 50]
2.	Python	[34, 35]
3.	General	[34, 36, 40, 44]

Table 2.4 describes different constructs of programming languages that were targeted for SCS. “Methods” are the most targeted construct in process of summary generation. Among 22 research articles, 18 of them have targeted “Java” for generating summary of source code, by using different data sources. Two of the researchers have targeted Python and 3 of them mentioned generic methodologies for SCS. 13 out of 22 researchers chose NLP as a part of their methodology and highlighted benefits of its usage in their research articles. 3 out of 22 used neural networks in their proposed methodologies. Furthermore, SCS methodology takes code as an input and gives simple language summary of the code as output. The input may vary i.e. some methodologies used comments as input along with code and some used only code as input. Most commonly used language for testing the methodologies was “Java” in past decade as shown in Table 2.5.

In Table 2.6, we have taken six research papers as an example and showed whether they have used code and comments for the generation of natural language summary or not. Our research paper title is

“Generation of Summary using natural language processing in c#”. We used source code as an input for source code summarization but not comments. Furthermore, in the [Table 2.6](#) we have shown which of the researchers has implemented their proposed methodology and has shown results in their articles. “Implemented” means they have implemented and shown in results in their research article. However, “Suggested” means that the researcher has not shown the results in their research article.

Table 2.6: Input of Code / Comments for Source Code Summarization

Ref	Code Comments	Source Code	Nature
[33]	No	Yes	Suggested
[38]	Yes	No	Implemented
[45]	No	Yes	Suggested
[30]	No	Yes	Suggested
[2]	No	Yes	Implemented
TCS	No	Yes	Implemented
[13]	No	Yes	Implemented

Table 2.7: Targeted programming constructs

Ref	Inheritance	Structure	Member functions	Member variables	Classes #	Nature
[33]	No	No	Yes	No	None	Suggested
[50]	No	No	Yes	No	None	Implemented
	No	No	Yes	Yes	None	Suggested
[30]	No	No	Yes	No	Unlimited	Suggested
[2]	No	No	Yes	No	None	Implemented
TCS	No	No	Yes	No	Unlimited	Implemented
[13]	No	No	Yes	No	None	Implemented

Use of different constructs by selected researchers

In [Table 2.7](#), we have discussed various constructs used by the researchers during their research. We found out that most of them have targeted methods for their generating Source code summaries. However, very few have targeted classes or more than one class. Furthermore, our methodology is using

classes and methods for the generation of Source code summary. As it is a vital part of a source code. Therefore, we used them in our methodology for natural language processing. In a research, they have targeted specifically methods and briefly explained their usage by processing on method's identifier, parameter's identifier and return statement's identifier for generation of natural language summary.

On the other hand, we are using the combination of classes, methods, conditional statements, iterative statements and return statements to generate natural language summary. As you can see in [Table 2.7](#), very few researchers targeted constructs other than classes and methods. Therefore, there was a dire need to work and improve the methodology used for generation of natural language summary.

2.4. Discussion

Different techniques were used by researchers for source code summarization. Clocom”, automatically generated comments by mining the existing code [42]. “JSummarizer” , is a suggested Eclipse plug in automatically generated summaries for Java Classes [45]. “Portfolio”, a tool suggested by researcher can find relevant functions and their usage in the source code [46]. An approach named “Code Attention” Translated of code into comments by focusing on symbols, keywords and critical statements using new attention technique [36]. Furthermore, “Minipar” , did Comprehension of Source code using LSI (Latent Semantics Indexing) , clustering and natural language processing [40]. Source code summaries were generated by crowd sourcing in “Crowd summarizer” [38]. One of the researcher used deep reinforcement learning to improve SCS [34]. Moreover, a researcher generated Comments from source code of python using NLP [35]. Neural networks along with NLP were used in a methodology developed by a researcher for SCS [33] . Automatic generation of Source Code Summaries of Java methods was done by using NLP too [13]. Comment generation may also help programmer during maintenance phase. Therefore,” Hybrid-Deep-Com”, was created by researchers using NLP , Deep Neural Network and Attention mechanism in their proposed methodology [37]. A research methodology generated natural language summaries by proposing a new neural model and used neural networks for getting required results [48]. “Methodman” , a proposed methodology, used pre-defined template to generate source code summaries [44]. A similar approach proposed a methodology to understand the function of the software code [47]. Moreover, a researcher used “Nano-Patterns” to improve the natural language summaries of java code [43]. “TASSAL”, a proposed tool , automatically creates code summary by folding code regions that are less informative [50]. A proposed methodology proposes to generate description of entities is generated using comments of the program in “EBSCS” [30]. Another methodology uses proposed model named “Tree2seq” for Neural Comment Generation for source code

by using auxiliary code classification task [39]. Moreover, a methodology generated code summary using combination of two techniques LSI (Latent Semantics Indexing) and VSM (Vector Space Model) [49]. Furthermore, Some researchers did generation of source code summary of java classes using NLP [2]and micro patterns [41].

During the maintenance phase, the developer doesn't have time to read the whole source code. Therefore, they need a quick understanding of the code to perform modifications efficiently. As, each entity in programming is somehow linked with other entities. Therefore, dependency between entities must be understood clearly for modifying the source code. "Automated-Summaries "of classes, are of great help in this regard. Therefore, a concise, correct and relevant summary of the source can help developers during maintenance.

As discussed in Section 3, 81.18% research articles have focused on generating the summaries of only one programming language "java". So, the summaries being generated are language dependent. However, to improve the scalability of SCS, we need to work on generating more "generic" summaries i.e. language independent summaries. Syntactical and logical difference in different programming languages make it quite challenging, to derive a general SCS approach.

Another challenge in SCS is to include maximum programming constructs, to generate a complete and comprehensive summary. As discussed in Table 2.7, majority of researchers are targeting classes, methods, iterative statements and conditional statements. However, more complex constructs are yet to incorporate in SCS techniques i.e. inheritance and polymorphism etc. Incorporating maximum constructs is significant for maintenance programmers. As, if a maintenance programmer has to understand the complete working of the code in short time, they won't be able to grip the complete concept of the relevant code , from the summaries which are currently being generated.

Moreover, researchers have been facing two important problems while generating meaningful summaries for source codes.1) determining what should be included in those summaries. 2) How to generate summaries automatically without help of developers. Quality of the summary is highly dependent on the good programming practices used by programmers. As, programmers are expected to use meaningful identifiers and method names. Hence, good summary is dependent on programmers. It is a huge challenge to generate meaningful description of methods and classes, when the identifiers are not meaningful.

Another important challenge in SCS is "quality-evaluation" of generated summaries. Literature lacks automatic quality evaluation tools and algorithms for source code summaries.

2.5. Analysis

This section provide details of the analysis performed between 9 tools identified from research articles. Name of tool, license type, purpose and language support of the tools are mentioned in Table 2.8. In SCS one of the biggest challenge is to evaluate the validity of generated summary. Our literature does not have solid ways to verify the quality of summaries. Hence, we have summarized different evaluation metrics, used by researchers to validate their summaries either subjectively or objectively.

Most commonly used quality evaluation metrics in last 10 years are summarized in Table 2.9. Most commonly used validation techniques are Human Evaluation (Questionnaire, Survey), Intrinsic online evaluation [49] and IR metrics [50, 51]. It is evidently clear that human evaluation is the most preferred evaluation choice. However, evaluation based on human response is error prone, consequently creates a huge validity threat.

Table 2.8: Tools

Ref	Tool	License-Type	Purpose	Language	
				Developed	Support
[38]	CrowdSummariser	Proprietary	Generate high Level Summaries	Eclipse Plugin	Java
[50]	TASSAL	Open-Source	Autofolding in SCS	Java	Java
[45]	JSummarizer	Open-Source	Automated Summary Generation	Eclipse Plugin	Java
[42]	CloCom	Open-Source	Mine software repositories for automatic comment generation	N/A	Java
[33]	ContextCC	Proprietary	Generate high Level Summaries	N/A	Java
[36]	CodeAttention	Proprietary	Generate high Level Summaries	N/A	General
[43]	Method Level Text Summarization	Proprietary	Generate Summaries	N/A	Java
[44]	MethodMan	Proprietary	Generate Summaries	srcML	General
[2]	JStereoCode	Open-Source	Identify Java code stereotypes	Eclipse Plugin	Java

Table 2.9: Evaluation Metrics

Ref	Validation Parameters	Validation
[38]	Conciseness, content adequacy, accuracy, comprehensibility	IR metrics
[50]	BLEU, METEOR	IR metrics
[49]	Number of relevant words, Length of Summary, Presence of	Intrinsic online evaluation

	method, class and attribute names in summary, Number of leading comments, Presence of verb and object	
[37]	BLEU, METEOR , Accurate Comment (Survey)	Machine translation metrics , Human Evaluation
[48]	BLEU score	Machine translation metrics
[47]	Semantic relevance	Human Evaluation
[13]	Accuracy, Content , Concise, Methods (what, why, how) , Orthogonality, Interpretation level	Human Evaluation
[46]	Relevance, Visualization of dependencies	Human Evaluation , IR metrics
[2]	Understandability, Readability, Content Adequacy, Conciseness, Expressiveness	Human Evaluation
[45]	Concise, complete, readable, understandable,	Human Evaluation
[43]	Correctness,completeness,conciseness, Non-Redundancy	Human Evaluation
[34]	BLEU, METEOR,ROUGE-L, CIDER	Machine translation metrics
[42]	context-sensitive text similarity measure (accurate,adequate,concise, useful)	Human Evaluation
[36]	BLEU, METEOR ,Understandability,Similarity, Interpretability, fluency, grammatical accuracy	IR metrics, Human Evaluation
[33]	BLEU-4 , METEOR, briefness, natural ,informative	IR metrics, Qualitative

Chapter 3

Proposed Methodology

CHAPTER 3: PROPOSED METHODOLOGY

Detailed description of concepts involved in our proposed methodology are present in this chapter. The recommended solution is derived using customized language processing method and general natural language processing that includes Tree Graph of stacks to identify the nested constructs up to two levels.

In our proposed methodology, first our tool will get source code written in C# as an input. However, there are some pre-requisites that must be followed while writing the code. These pre-requisite includes proper naming of identifiers and methods. Once code has been taken as input it will first be tokenized. The process of tokenization has been discussed earlier in 1st chapter of this paper. However, the tokenization we have used in our methodology is purely based on tokenizing the source code of C#. Therefore, it picks up tokens and saves it accordingly in an array. Next step, is to identify the constructs that are part of another construct ; For example, whether a function is a part of a class and whether the conditional statement identified is a part of that function or not. This is the most complex part of the processing. As the limit of nesting of constructs couldn't be limited. However, we limited the nesting of constructs up to two levels. Furthermore, each construct is considered as a stack. The top most stacks will be executed first and then next one for maintaining the sequence of execution of programming constructs.

Firstly, we will discuss the proposed methodology in general steps. Later on, we will be going into details of our proposed methodology. As you can see in [Figure 3.1](#) our methodology consists of majorly three steps. In first, Source code is taken as an input. In second step, customized processing is done on source code with natural language processing side by side. In this step all the processing is done. Furthermore, from tokenization to the generation of plain language summary of each construct is generated in this step. In the final step all those summaries are collected together and processed in to a single summary of the source code in plain language. The output is a summary which in plain language explains the functionality of the code.

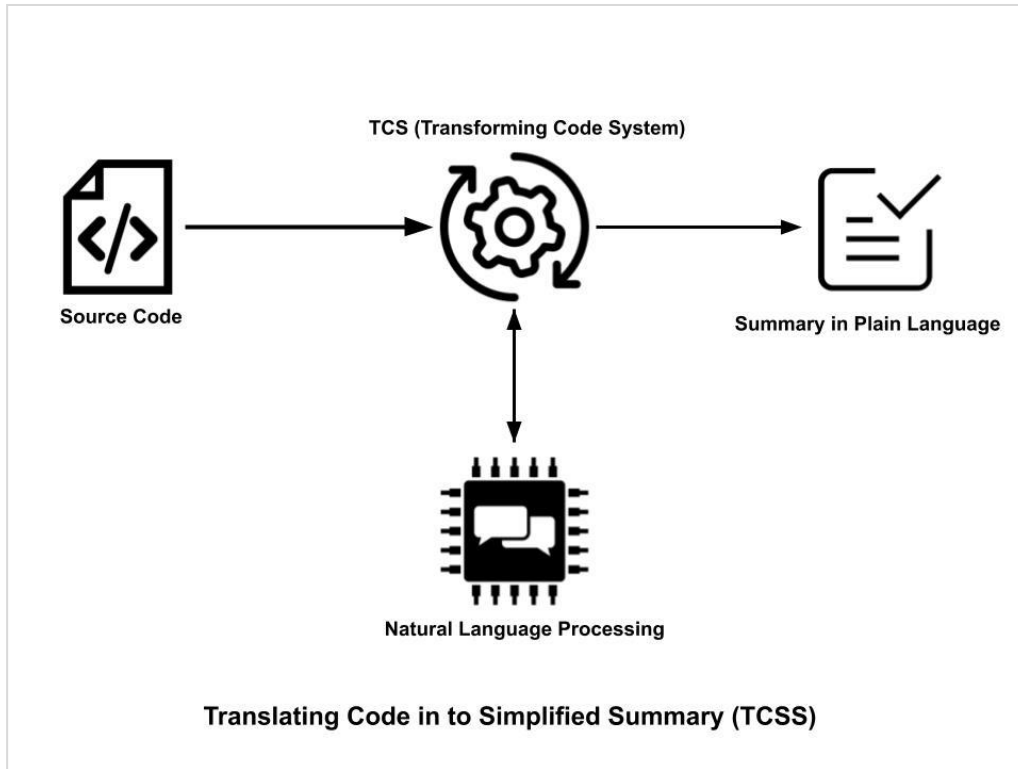


Figure 3.1: Translating Code to a Simplified Summary (TCSS) Architecture

Going into details lets discuss [Figure 3.2](#). In [Figure 3.2](#), the process of TCSS is explained in detail, especially the part where most of the processing is done. Furthermore, the part where we had to define a structure of program was the most difficult and complex. The solution of such a problem was solved by using a combination of complex structures like stacks and graphs. TCSS includes six major steps which are as following:

- Tokenization
- Construct Identification
- Structure of Constructs
- Putting constructs in relevant stacks
- Linking Stacks with other stacks
- Natural Language Processing

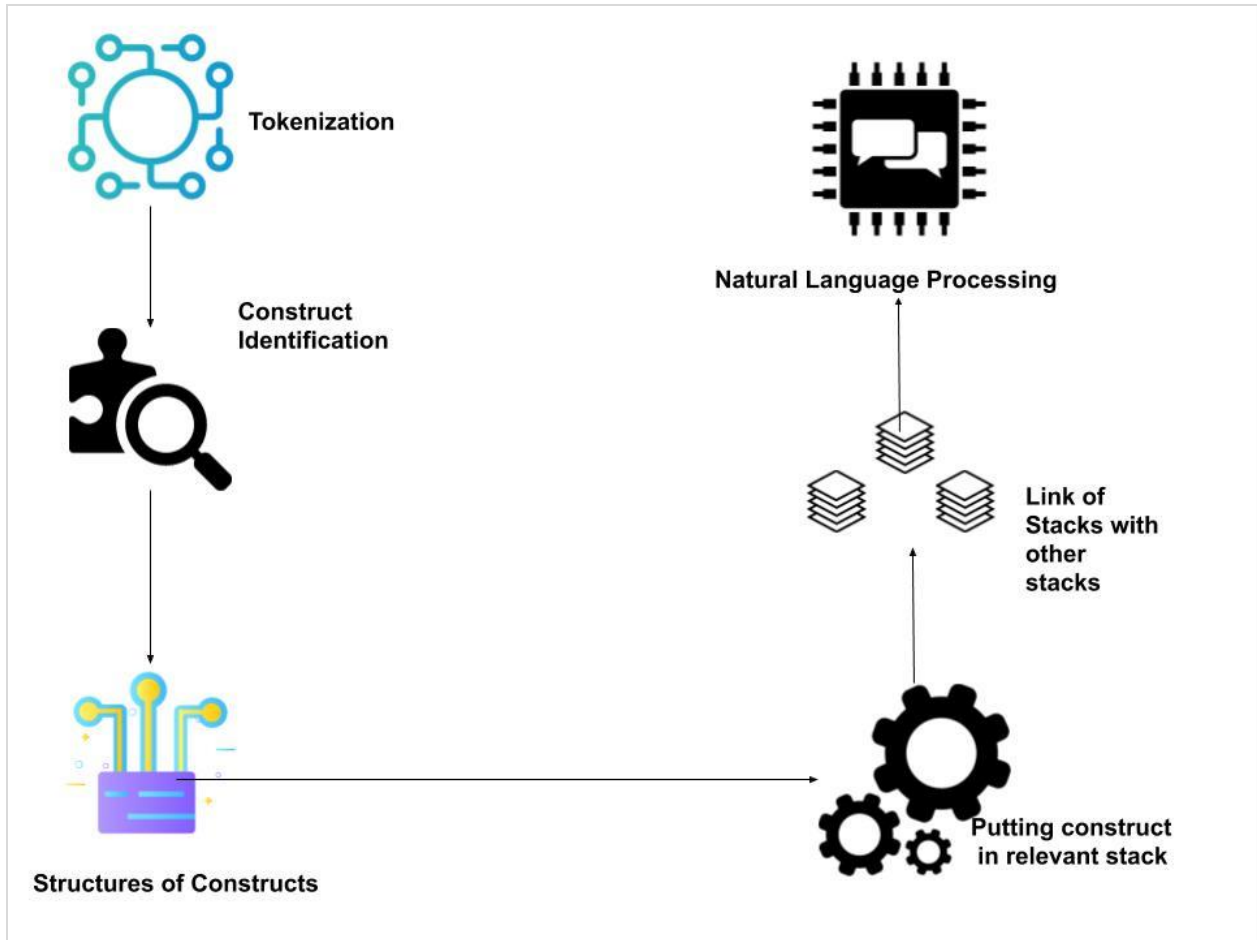


Figure 3.2: Translating Code In To Simplified Summary (TCSS) Architecture

To begin with, In [Figure 3.2](#), the first step of TCSS is tokenization, in this part the source code is read and converted into tokens. Each token is basically a part of programming language. As you can see in [Figure 3.3](#). A conditional statement has been tokenized.

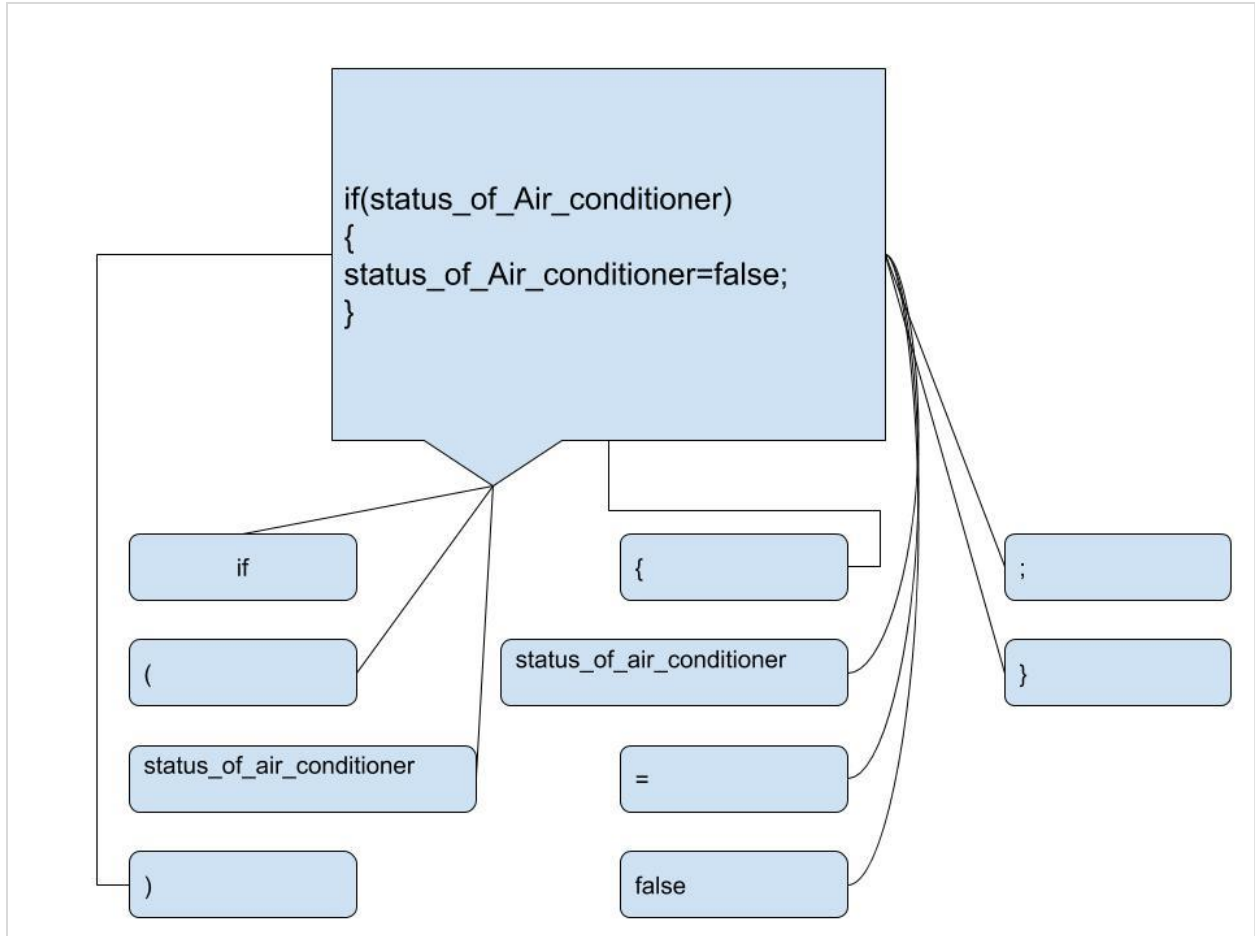


Figure 3.3: Tokenization of if conditional statement

In [Figure 3.3](#) we have shown how tokenization is performed of conditional statement. This tokenization would be done by using a customized function which will consider the conditional statement as a single token. Moreover, the example used in the [Figure 3.3](#) has considered “status_of_air_conditioner” as a single token. This has been done for future processing for generation of summary of conditional statements. Furthermore, we tokenized the input code. Each token represents a word of the code; for example, in the given example:

```
for (inti =0 ; i<3 ; i++)
```

There are 12 tokens.

Furthermore, In [Figure 3.2](#), the second step of **TCSS** is identification of construct and the third one formation of structure of construct. We have divided the constructs into methods, classes, conditional statements (if for implementation purposes) and iterative statements (for only). It is because, we want to identify the logic used in the methods which could be covered only if we

include the brief explanation of conditional statements and iterative statements in our summary. As discussed in the **Chapter 3**, inclusion of logic in source code summary can play a vital role in helping programmers for improving efficiency of program. So, we considered briefly describing the function of conditional and iterative statements in our methodology.

Table 3.1: Tokens

Word	Array index	Token
for	0	1
(1	2
int	2	3
i	3	4
=	4	5
0	5	6
;	6	7
<	7	8
3	8	9
i	9	10
++	10	11
)	11	12

Moreover, each structure has been designed specifically for the usage of source code summary which will be discussed in the next chapter. Once a structure has been identified it is added to it is added into its relevant construct. If it is not a part of a construct then it is not added. For example, member functions of a class will be added in the structure of class and constructs of a function will be added in the structure of that relevant function which has been shown in [Figure 3.4](#).

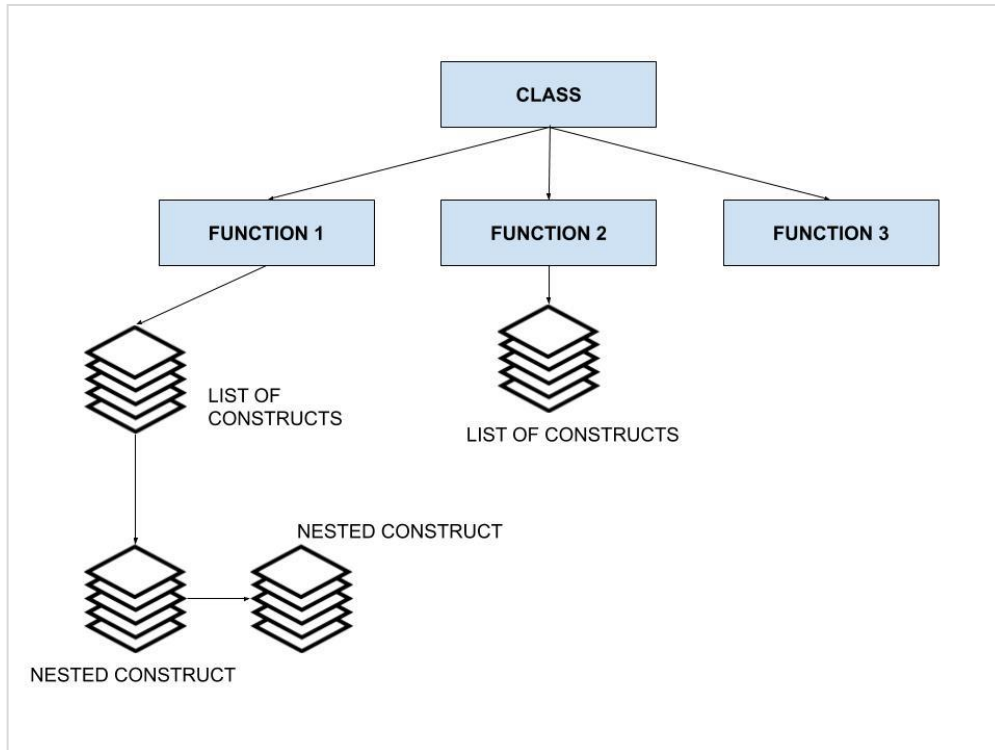


Figure 3.4: Programming constructs structuring for Natural Language Processing

As you can see in [Figure 3.4](#), we have presented each node of the graph as a construct of program. Furthermore, these nodes can be stacks of more constructs inside that node. The reason of doing so is that for brief description of function we would need to know the order of constructs and the information in which we can know which constructs are part of another construct. This will play a vital role in further processing of generation of natural language summary. Nesting is commonly used in programming codes. Furthermore, it is very important for us to identify the hierarchy of the constructs within a function to identify its logic. Therefore, we had to identify the hierarchy. For doing so we made an algorithm by using stacks.

To solve the nesting problem, we have used an algorithm as follows:

Firstly, we considered every construct's body in function as a stack. Let's name this stack as "S1". Suppose, "if" and "for" are two constructs. Consider the following code:

```
if(speed ==0) // i1
{
for(inti =0;i<3;i++) //f1
{
```

```

if (acceleration<3) //i2
{
    Acceleration++;
}}

```

In the above code, there are three constructs i.e. two if and one for. Additionally, in comments we have named them as “i1”, i2, f1. Let’s define the hierarchy of this code, f1 is defined inside the body of i1 and i2 is defined in body of f1. So, to identify the hierarchy in the code constructs we needed an algorithm. According to that algorithm, we defined a stack. In that stack each construct will be pushed unless we found a “}” in the given code. Once we found it, we will consider the construct present at the top position a part of construct that is present below it. From the above code, we can make the following stack:

i2
f1
i1

Form the above stack we can identify that i2 is a part of f1. So, we will add i2 in the instance of f1. By doing so, we would be able to identify the logic. Let’s say, we get a description “acceleration>3 will be checked several times”

f1
i1

Once this is identified we will remove i2 from the top and then f1 would be at the top. This process will repeat until there is only one element left in the stack. By doing so, we can identify the hierarchy of the code constructs. So, at the end of this effort, we will get a description “When speed==0, acceleration>3 condition will be checked several times. Each time if it is true, “acceleration++” will be executed.”

Usage of Stacks

Constructs like for, if etc. can have multiple constructs within their body. Along with that, for understanding the nominal logic of that body, the arrangement of the construct must also be known. Therefore, for knowing the pattern, we choose to consider a stack of constructs that were used in the body of any construct. For generation of the natural language we need to know the order of the constructs too. So, stack can maintain the order of execution of the statement.

Natural Language Processing

Next step of our proposed methodology is Natural Language Processing. This process is done on different occasions during the methodology where it is required. This step in parallel fixes the identifiers and conditional statements so they can be understood easily by the reader of the summary. Furthermore, special characters between words are removed in this section. For example, an identifier of a function named as “stop-engine” will be processed as stop engine. Moreover, verbs and nouns are separated in this process. With verbs “s” or “es” are attached with verbs according to their nature. For example, “stop-engine” will be changed to “stops engine”. The reason of doing so is that we would be using present indefinite tense in our generated summaries. Additionally, conditional statement’s conditions are also processed for better readability. For example, if there is a statement “length > array” would be converted into “length is greater than array”. This will improve the readability of the summary.

Once, the structure of constructs is completed, Custom language processing is also done to convert different chunks of English phrases into complete sentences using different templates. Furthermore, these sentences are connected with connectors to convert them in a paragraph so they are easily readable for readers. Different elements of sentences are used during the processing. Sentence connectors like: furthermore, moreover etc. are used to fulfill the purpose of paragraphing. For example, Consider there is a function that is returning an output by taking some parameters as in input.

Let's consider the following code:

```
Class car
{
Void forward (string accelerator)
{
Return speed;
```

```
}  
}
```

Our description in such case would be “The instance of car can forward. When the car forwards it gives speed by taking accelerator as an input.” However, if the function is not taking any input, our description will change accordingly. It will be “The instance of car can forward. When the car forwards it gives speed without taking any input.”

On the other hand, if there isn't any input or out of the function then our description would be “The instance of car can forward. When the car forwards it performs some required actions without taking any input or giving an output.” However, There are some pre-conditions that must be followed else our proposed methodology won't give promising results. These pre-conditions are similar to earlier pre-conditions set by earlier researches. One of them is that Function names must be kept in an order of verb followed by noun. If it doesn't the coder must mention noun and verb in the identifier of the function. Majorly, the only that the programmer or the coder must remember is to give proper names according to the coding standards that are variables must be identified by nouns and methods must be identified by verbal names.

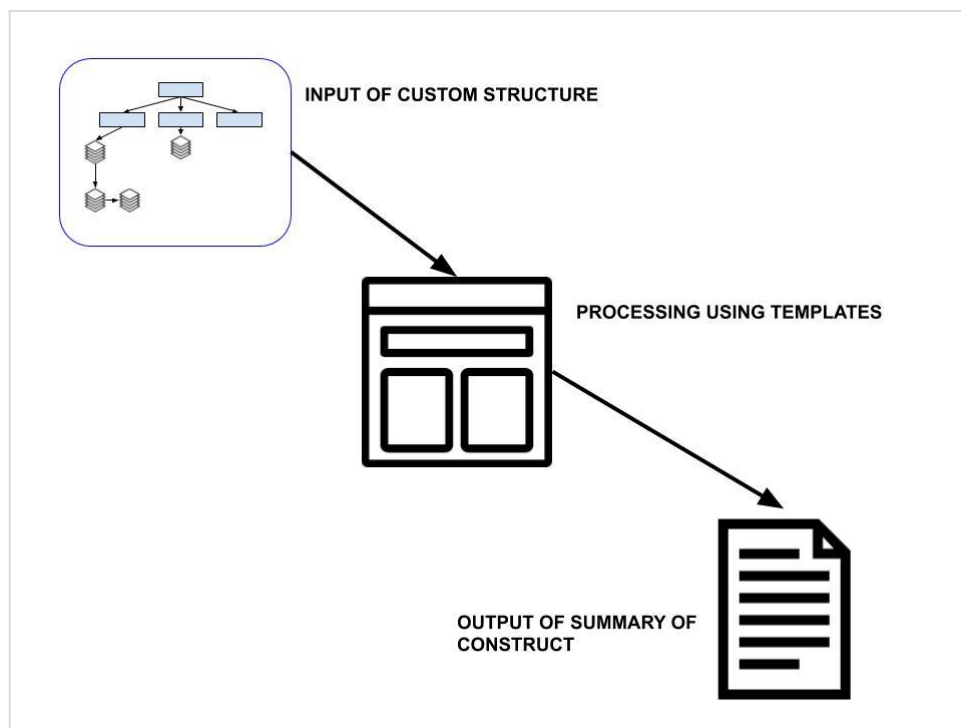


Figure 3.5: Translating Structure in to Simplified Summary using templates

In [Figure 3.5](#), the procedure for translation of structure of construct stack has been shown.

Comparison

There are many comparisons between old techniques and the one used by us. Previous researchers have been solving specifically different problems related to source code summarization. Therefore, the characteristics may vary from research article to other one. For example, P.W. MCBurney's work [13] in the paper in which they generated source code summaries for the Java methods has many factors that are different from our methodology. Like we are targeting different combination of constructs then they as well as we are also targeting a different programming language. Furthermore, our summary is lengthier than theirs and includes explanation of Classes too. Moreover, we didn't work on the importance of methods among other methods because during M.C Burney's survey most of the programmers rejected the concept that one method can be better than the other one on the basis of its frequent use. Rather than that, we focus on identifying and briefly explain the logic of different methods by generating summaries from constructs like conditional statements and iterative statements. As these both , together lay a strong foundation of programmer's logic. Furthermore, we have also improved the presentation of the summary as anything written with an appropriate level of presentation is better in terms of readability.

In Table 3.51, the comparison of our methodology has been show by showing different characteristics. For example, ASCJM stands for Automatic Source Code Summarization of Context for Java Methods and is work of P. W. McBurney C. McMillan and ASCCM stands for Automatic generation of natural language summaries for Java classes a paper published in 2013 and AJMNN stands for Augmenting Java method comments generation with context information based on neural networks, a paper published recently in 2019. As shown in Table 3.51, internal logic description was missing in the generated summaries by the below three researchers. The reason of doing so was their goal, which was to generate brief and concise summaries that can only give an eagle view of source code for the readers. Furthermore, the combination of constructs is different than others because in some methodologies their target was only to explain the reason of existence and importance of a method. Whereas, some only focus on class's description while generating summary. We have not only combined these two but also highlighted the working of iterative statements and conditional statements in our work.

Characteristics	TCS	ASCJM	ASCCM	AJMNN
Internal logic in summary	Yes	No	No	No
Methodology	Tree of Stacks and NLP	SWUM	Tokenization, NLP	Neural networks and NLP
Constructs	Class, member functions, loops, if condition, return statements, function parameters	Methods, method's parameters, method's return statement	Classes	Methods,for,if,try-catch
Input	Code	Code	Code	Code and Comments
Programming Language	C#	Java	Java	Java

Table 3.51: Comparison with old methodologies

3.1. Tools and Techniques Architecture

In order to increase the process of software research and development, various tools were used during our research and development. “End note” [55] was used for research purposes whereas “Visual Studio” [56] was used for the development of tool which will implement our proposed methodology. Furthermore, “Google forms” [57] was used for validation purposes. “Google drawings” [58] was used to making drawings in the thesis. Natural language processing libraries[59] were used in order to perform various language processing requirements.

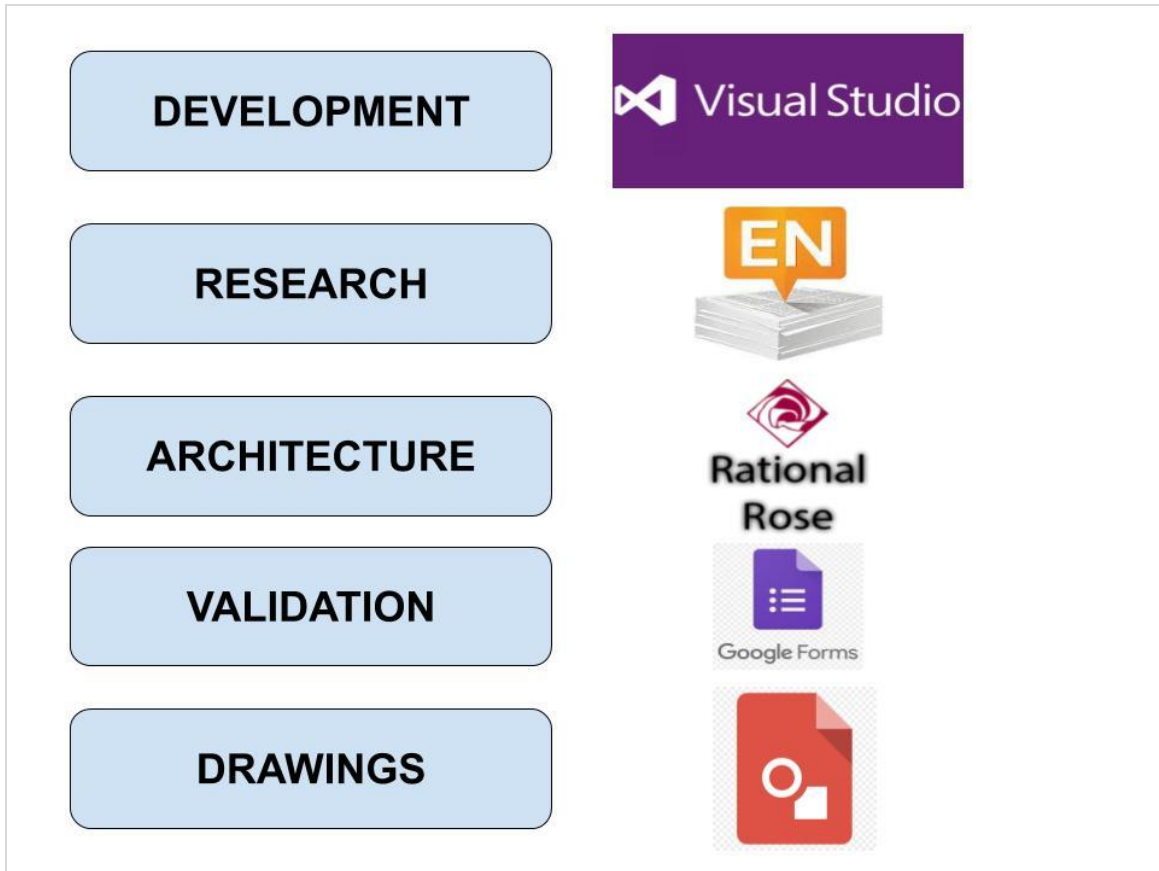


Figure 3.6 : Tools and Techniques Architecture

3.2. UML Details

Uml diagrams are important for software development as well as software maintenance [52]. Therefore, we have developed some uml diagrams in order to explain our implemented software in a better way.

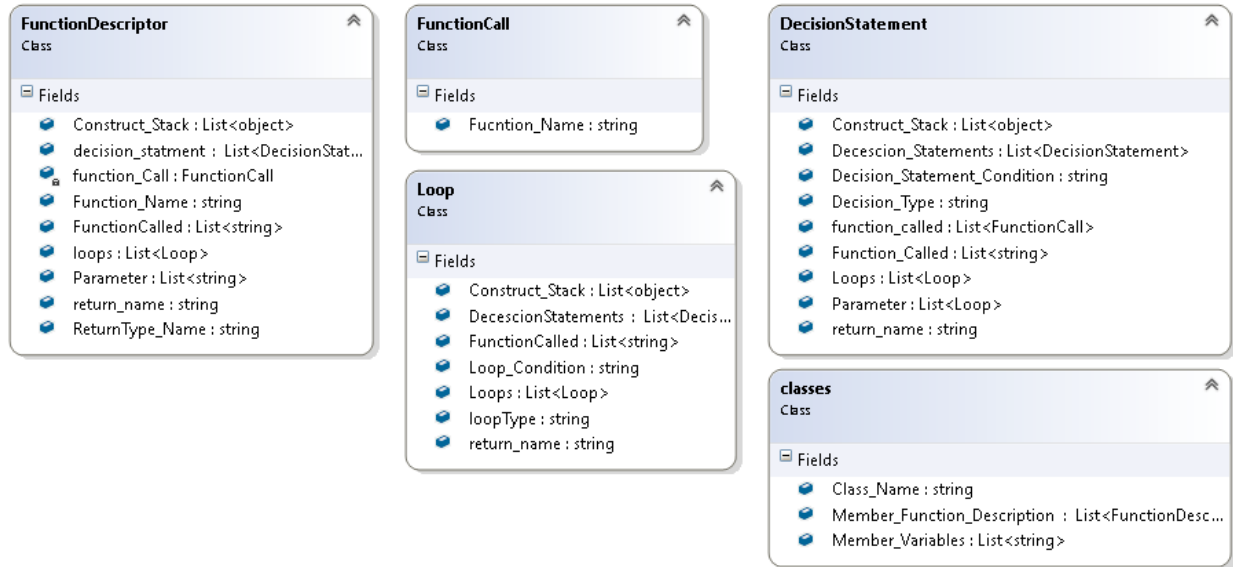


Figure 3.7: Programming constructs details

Figure 3.7 presents the details of the programming constructs which would be covered in order to generate natural language summary. To begin with, “Classes” is representing the class construct in which we are saving identifiers, details of member functions and member variables. Later, during processing they will be processed to gather valuable information for our generated summary. Secondly, “loop” represents the iterative statements in which the information regarding an iterative statement will be saved. Furthermore, as you can see in Figure 3.7, the class of “loop” has various member variables. These variables have usage of their own. Moreover, “Construct_Stack” is basically a stack of constructs that would be a part of this iterative statement. This could be a conditional statement or any of the structure mentioned above in the Figure 3.7. The reason of doing so has been explained earlier above in Figure 3.4. Moreover, number of loops with in this loop and number of decisions statements are also recorded. Similarly, conditional statements are also saving relevant information in “Decision_statement” class. It has member variables to save information specifically of an “if statement”. Furthermore, “Function_call” covers the information of the functions called within other constructs.

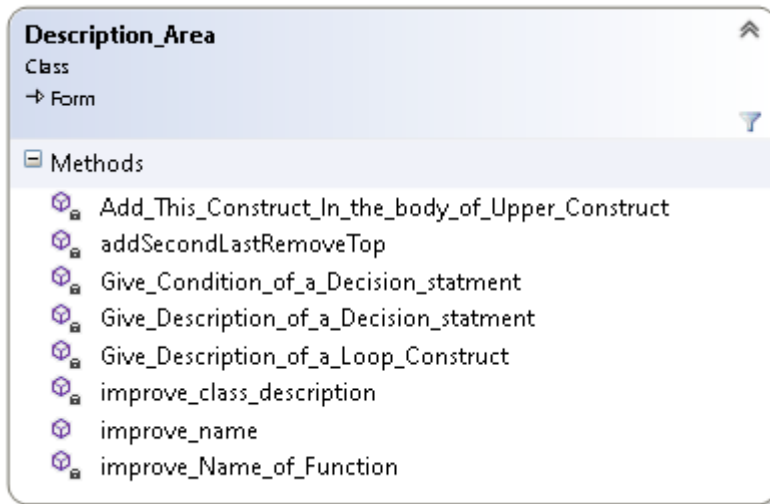


Figure 3.8: Class diagram of Description Area

In [Figure3.8](#), not all the methods of this class has been shown. Only major activities carried out in TCSS are shown. We will explain each of the method briefly for a better understanding of the processing.

Add_This_Construct_In_The_Body_of_Upper_Construct

The functionality of this method basically to add a construct in the construct of which it is part of. By doing so, we can get an idea which construct is part of which construct and this will help us in generating summary of source code.

AddSecondLastRemoveTop

Once a construct has been used for natural language processing it is removed from the stack. The method “addSecondLastRemoveTop” does removes the construct from stack once it is used for natural language processing.

Give_Condition_Of_Decision_Statement

We need to understand the condition of a decision statement before describing its need in the function. Therefore, the above function “GiveConditionOfDecisionStatemen” extracts the conditional statement. Furthermore, it also removes special characters used in the conditional statement and replaces it with natural language. For example, if a condition of a conditional statement is “array_length>number” will be converted into array length is greater than number.

Give_Description_Of_a_Loop_Construct

Similarly, just like conditional statements, iterative statement like “for loops” also need to be identified and briefly described. The method “Give_Description_Of_a_Loop_Construct” is doing that task for us. This one is vital as there can be many conditional statements part of an iterative statement in source code.

Furthermore, the remaining methods improve the identifiers of classes and description. Moreover, they also improve the description of classes by performing some customized natural language processing to improve its readability and understanding.

Chapter 4

Implementation

CHAPTER 4: IMPLEMENTATION

This chapter majorly focuses on the implementation of our proposed methodology. Furthermore, we share the details of the **TCSS (Translating Code in to Simplified Summary)** design and working for generation of source code summary. Moreover, this chapter provides details about the architecture of transformation engine used for generation of source code summary in **Section 4.1** and the transformation rules that we applied are discussed in **Section 4.2**.

4.1. Architecture of Transformation Engine (MUTE)

The overview of architecture of TCSS is presented in [Figure 4.1](#). There are two important components of this system i.e. User Interface (UI) and Source Code. The transformation is carried out using the methodology we have shown in earlier. Moreover, the methods carrying out the functionality explained earlier are performing different processes TCSS. Furthermore, the program was developed in Visual Studio for testing the methodology. The languages used for development is C# and the source code which we tested on this program was written in C#.

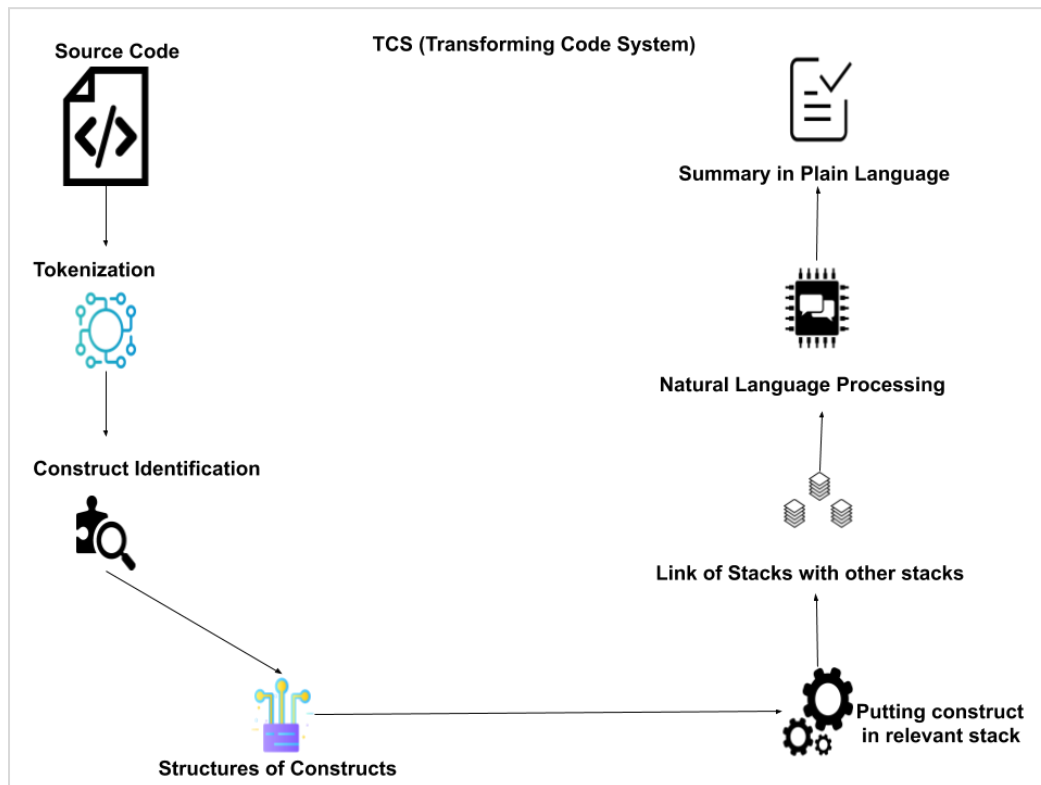


Figure 4.1: Transformation Code System

4.1.1. User Interface

A good Human computer interface is vital for better interaction of humans and computers [53]. The interface in [Figure 4.2](#) shows the main screen of TCS. It consists of following features that are as following:

- *File*
- *Edit*
- *Font*
- *Color Picker*
- *Help*
- *Exit*
- *Code input area*
- *Submit Button*

To Begin with, *File* lets you write a new code by pressing “*New*” button that can be seen in [Figure 4.1](#). *New* allows you to discard all the screens and write or paste a new code for generation of summary. Moreover, *Edit* button lets you select, cut a chunk of code, copy and paste the code from code input area to another screen. *Color Picker* lets you change the color of the code. *Code input area* takes code as an input. In this area, you can either write a code or paste it. For now we have implemented the methodology on programming language of C# whereas, for future work, we will implement more languages. *Submit Button*, when pressed, lets you submit your code for further processing for the generation of natural language summary. Moreover, number of lines can also be seen in the *Code Input Area*. This helps the user to understand how much lines of code he has yet written. However, once user writes the code in the *Code Input Area*, the interface changes.

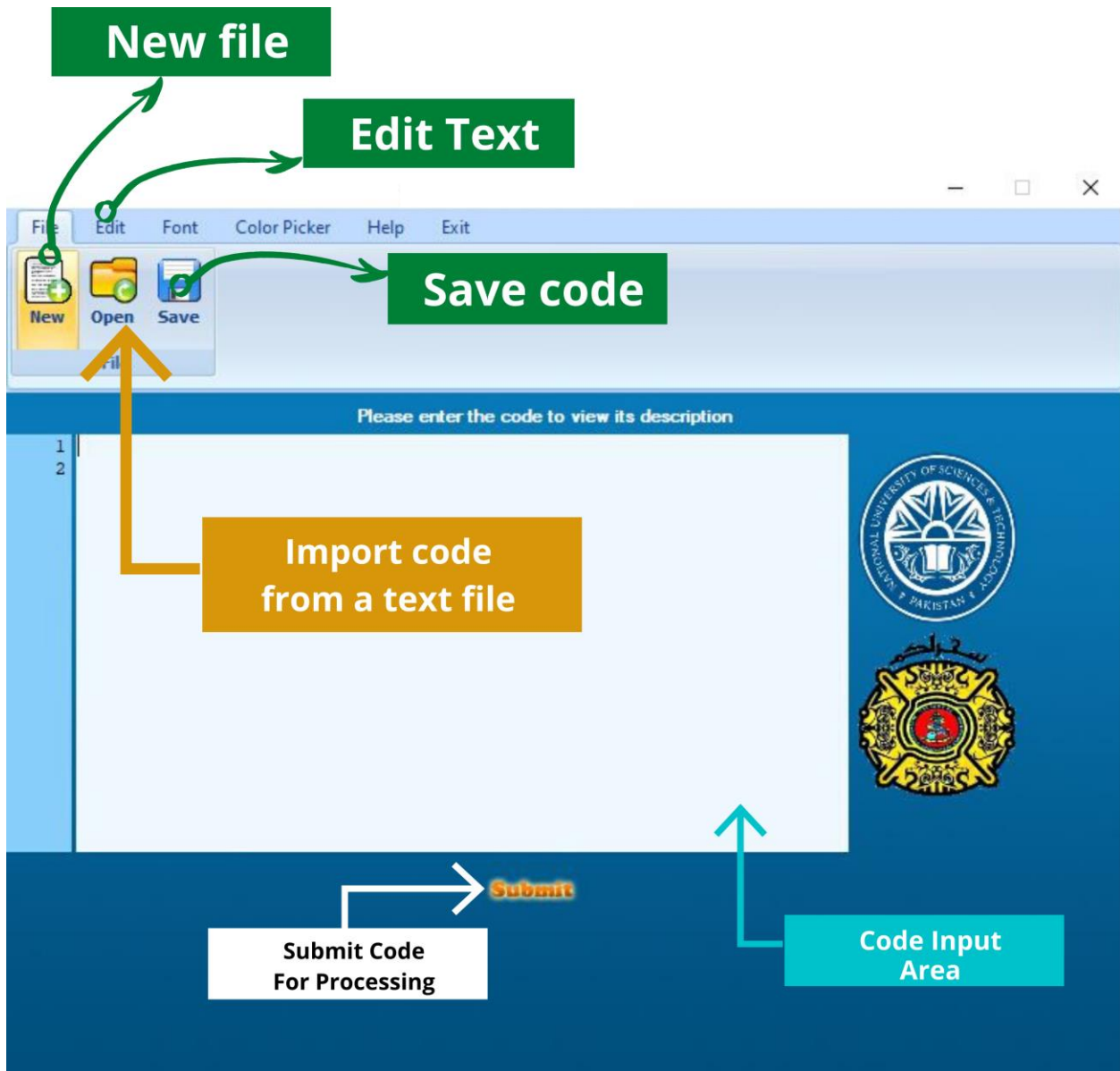


Figure 4.2: Main Screen of TCS

Moreover, [Figure 4.3](#) shows the user interface when he/she clicks on *Open*. It opens a file box from where user can select a text file from where code can be pasted into *Code input area*.

Similarly, when you press the *Save* button, it opens a dialogue box where one can select the location where they want to save the source code written in the *Code Input Area*. This helps user to define the destination location for the source code. *New* button allows user to reset *Code Input Area* and clear the summary generation interface. *Submit* button starts the transformation process. *Open* button option directs the user to destination folder. *Cancel* button cancels the operation and brings back user to the interface shown in [Figure 4.1](#).

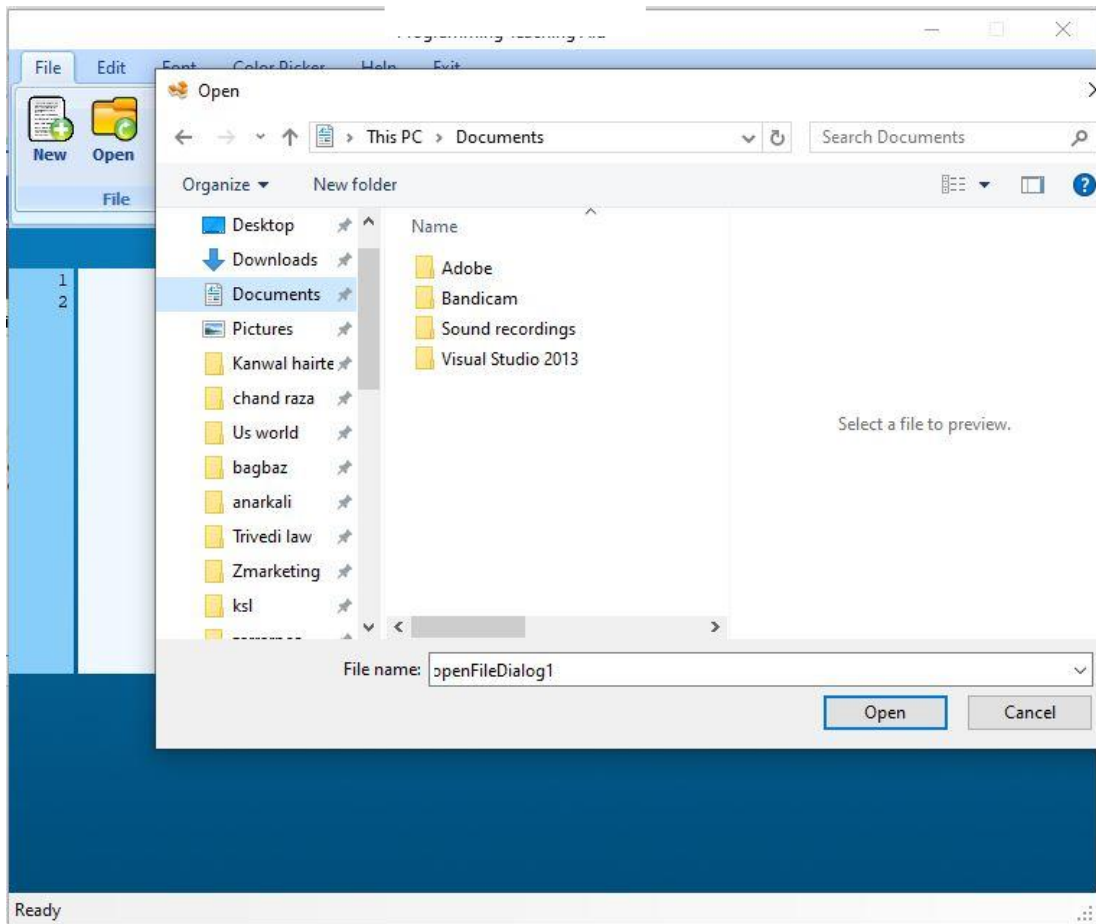


Figure 4.3: Open Interface for TCS

4.1.2. User interface for output

When user will input source code the screen will which is show in [Figure 4.4](#). Furthermore, this screen includes one more window in which the generated summary has been shown. For an example we have shown an example of a simple class. The code consists of single class with a single member function named as reverse. The return type is “void”. There are some changes more to user interface shown in [Figure 4.4](#) .Firstly, one is that the buttons of *Save* and *Open* has been eliminated. Secondly, user cannot edit any of the text box included. There are two text boxes included in this user interface. First one is *Code Input Area* and second one on the right shown in [Figure 4.4](#) summary area. Summary area shows the generated summary that was generated by using the Transformation system on the basis on our proposed methodology. The summary has some parts. To begin with, the first part consists of explanation of the functions consisted by a class. The second part consists of brief description each function one by one. Furthermore, each function description has been given a “Heading” before the description itself.

This will improve the readability of the summary. Because, in the end, all what matters is the readability and ease of the generated summary. Therefore, we tried our best to facilitate the reader by giving different options like: presenting the functions in a list and adding connectives before starting another sentence. Moreover, we tried our best to present the information as natural as we can.

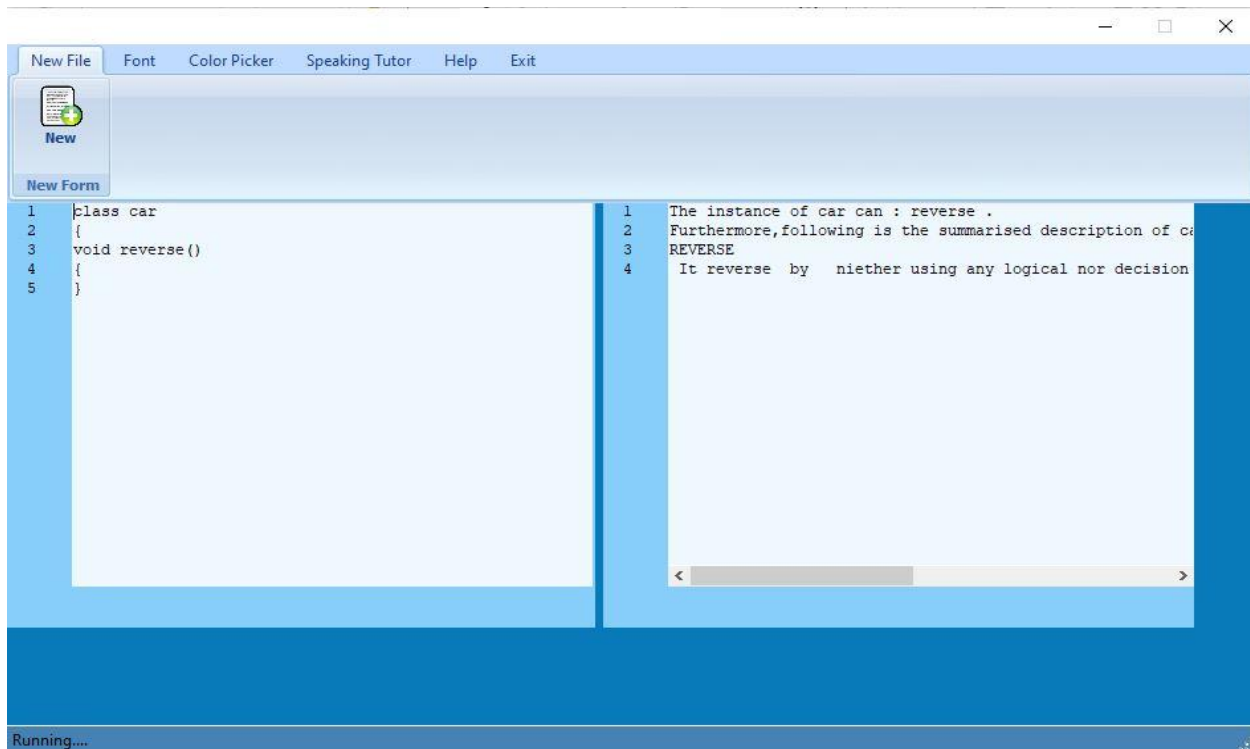


Figure 4.4: User Interface for generated summary and code once submit button is clicked

4.2. Transformation Rules

Furthremore, in the [Table 4.1](#) we have shown the templates used in our procedure. The possibilites that would be covered during implementation has been shown in the [Table 4.2](#).

Consider we have a program that has a “for” construct. Furthermore, in it, suppose there is a nested loop. In this case, the description of the function would be “this function would perform required actions repetitively.” The description has been kept simpler in order to understand the usage of the loop constructs. However, the statements used in these constructs could also be specified but for now that have been left for future work. Similarly, the working and the scenario of other possibilities has been shown in [Table 4.2](#).

Table 4.1: Transformation Rules

Construct	1 st NestedConstruct	2 nd Nested Construct	Template + Example
for	If	None	This “class name” can “function name”. It does it by repetitively checking “if condition”; For example, The instance of car can accelerate. It does so by repetitively checking speed is greater than 0 whether it is true or not to perform some required actions.
for	For	None	This “class name” can “function name”. It does so by repetitively performing specific actions; For example, The instance of car can accelerate. It does so by repetitively performing required actions over and over again.
for	None	None	This “class name” can “function name”. It only performs some actions repetitively. ; For example, The instance of car can accelerate. It only performs some actions repetitively.
if	For	None	This “class name” can “function name”. It checks “condition of decision statement” and if it is true it performs some required actions repetitively. ; For example, The instance of car can accelerate. It only performs some actions repetitively.
if	If	None	The instance of “class name” can “function name”. It checks “condition of decision statement” and if it is true then it checks “nested decision statement condition” to perform required actions. ; For example, The instance of car can accelerate. It checks weather speed is equal to 0 or not and if it is true then it checks if break is equal to true to perform required actions
if	None	None	The instance of “class name” can “function name”. It checks “condition of decision statement” and if it is true then it checks then performs required actions. ; For example, The instance of car can accelerate. It checks if speed is equal to 0 or not. Furthermore, if it is true perform required actions
for	If	If	It repeatedly checks whether speed is equal to 0 or not and stop is equal to 1 or not to perform some required actions on the basis of whether they are true or not.
for	For	If	It repeatedly performs some actions over and over again and checks whether speed is equal to 0 or not.

for	If	For	It repeatedly checks weather speed is equal to 0 or not and performs required actions over and over again.
for	For	For	It repeatedly performs required actions over and over again.
if	If	If (Second decision statement)	It checks “condition of first decision statement” and “condition of second decision statement” to perform actions accordingly on the basis of their truthfulness.
if	For	If (nested in for)	It checks “condition of first decision statement” and repeatedly checks “condition of the nested statement” to perform required actions.
if	If	For (nested in the previous if)	It checks “condition of first decision statement”, if it is true then checks “condition of the nested statement” and if this one is true then repeatedly perform required actions.
if	For	For	It checks “condition of first decision statement” , if it is true Then repeatedly perform some actions over and over again.
if	function call	None	It checks “condition of the decision statement” , if it is true then it performs the function of “function name”
If	For	Function call	It checks “condition of decision statement” and if it is true it performs some required actions repetitively and performs the function of “function name”
if	If	Function call	It checks “condition of the decision statement”, if it is true then “condition of the decision statement”. If it is true then it performs the function of “function name”
For	If	“If” has a value to return	It repeatedly checks “condition of if” and if it is true then returns “return value name” as an output. For example: It repeatedly checks weather speed is equal to 0 or not and if it is true then returns the speeds value.

Table 4.2: Templates for constructs and nested constructs of “If” and “For”

Class name	Number of Functions and Their Name/Names	Template /Description
Car	One and the function name is accelerate.	The instance of the “class name” can only “function name”. For example, the instance of a car can only accelerate.

Car	Two and the function names are accelerate and stop.	The instance of the “class name” can “function name” and “function name”. For example, the instance of car can accelerate and stop.
Car	In this case there may be Three or more functions. For example function names are accelerate,stop,reverse and go_forward.	The instance of the “class name” can “function name”,“function name”... and “function name”. For example, the instance of car can accelerate, stop, reverse and go forward.

In the beginning, suppose we have a class named “car” and it has three functions named as “accelerate, stop and reverse”. So our program will give the description of the code as “the instance of the car can accelerate, stop and reverse. “However, if there are only two functions then the description will be “The instance of the car can accelerate and stop. “On the other hand, if there is only one function in the class then the description will be “The instance of the car can only accelerate.” However, we changed the style of the summary later on for improving the readability. For doing so, we introduced bulleted information as well as we introduced some information with headings. The reason of doing so was that if there were more than 10 member functions of the class then readability would had been compromised and poor in quality.

4.2.1. Thermal engine example implementation

Before going further, we tried a code in our tool to implement our proposed methodology. First we tried a simple class with two types of functions one with a parameter and the other without one. In [Figure 4.5](#),we took a class of “Thermal_Engine” having two member functions. Moreover, the generated summary is show in the [Figure 4.5](#). The headings of each function name were given in Capital Letters to improve the readability for the user. The reason of using technical word like “output”was, because these summaries would be read by programmers and maintenance engineers. Therefore, such words are common for them in daily routine and will help them to understand the crux of the program in a better way.

```

class Thermal_Engine
{
    int Give_Engine_Temperature ()|
    {
        int temperature=0;
        return temperature;
    }
    int Give_Pressure(int atmospheric_pressure)      {
        return atmospheric_pressure;
    }
}

```



```

1 The instance of thermal engine can : give engine temperature and give pressure .
2 Furthermore, following is the summarised description of thermal engine functions.
3 GIVE ENGINE TEMPERATURE
4 It gives engine temperature by giving temperature as an output.
5 GIVE PRESSURE
6 It gives pressure by giving atmospheric pressure as an output.

```

Figure 4.5: TCS Output

Moreover we used another set of member functions within a class. It was impossible to show the whole code in the [Figure 4.6](#), that is why we showed most of the output. However, the output and the source code are mentioned separately here.

<pre> 1 ss Thermal_Engine 2 { 3 1 check_stability_of_engine 4 { 5 bool stability_stat 6 if (temperature>50) 7 { 8 stability_statu 9 return stabilit 10 } 11 else 12 { 13 stability_statu 14 return stabilit 15 } 16 string caclulat 17 { 18 string life_of_engi 19 int life=5; 20 21 if (years_used<5) 22 { 23 life -= years_u 24 life_of_engine 25 } 26 else 27 28 29 </pre>	<pre> 1 The instance of thermal engine can : check stability of engine and caclulate engine life . 2 Furthermore, following is the summarised description of thermal engine functions. 3 CHECK STABILITY OF ENGINE 4 It check stability engine by checking temperature is greater than 50 and if it is tru 5 and giving stability status as an output. 6 CACLULATE ENGINE LIFE 7 It engine life by checking years_used is lesser than 5 and giving life of engine as ar </pre>
--	--

Figure 4.6: Using TCS for SCS

Source Code:

```
class Thermal_Engine
{
boolcheck_stability_of_engine(int temperature)
{
boolstability_status;
if (temperature>50)
{
stability_status = false;
returnstability_status;
}
else
{
stability_status = true;
returnstability_status;
}
}
stringcaclulate_engine_life(intyears_used)
{
stringlife_of_engine;
int life=5;

if (years_used<5)
{
life -= years_used;
life_of_engine = life.ToString() ;
}
else
{
life_of_engine = "expired";
}
returnlife_of_engine;
}
}
```

4.2.2. Generated Summary

The instance of thermal engine can: check stability of engine and calculate engine life.

Furthermore, following is the summarized description of thermal engine functions.

CHECK STABLILITY OF ENGINE

It check stability engine by checking temperature is greater than 50 and if it is true then it gives stability status as an output.

And giving stability status as an output.

CACLULATE ENGINE LIFE

It engine life by checking years used is lesser than 5 and giving life of engine as an output.

There are different sections of our generated summary. The first part which is highlighted in the *generated summary* section with blue ink is the one which defines the classes. The second part is the connector which is highlighted with Pink color. Furthermore, this part joins the two sections

of the summary that is the first one with the third one. The third part of the summary is the brief summary of methods in which we describe the logic of each method. Furthermore, this part consists of capital headings. The heading defines each method and is used to improve readability of the user.

Table 4.3: Different type of class summaries

Class name	Number of Methods	Method's identifier	Summaries
Thermal_Engine	One	The method's identifier is check_stablility_of_engine.	The instance of the "class name" can only "function name". For example, the instance of a thermal engine can only check stability of engine.
Thermal_Engine	Two	The function names are check_stablility_of_engine and caclulate_engine_life.	The instance of the "class name" can "function name" and "function name". For example, the instance of car can check stability of engine. And calculate engine life.
Thermal_Engine	More than Two	In this case there may be Three or more functions. For example function names are accelerate,stop,reverse and go_forward.	The instance of the "class name" can "function name", "function name"... and "function name". For example, the instance of thermal engine can accelerate, stop, reverse and go_forward.

In [Table 4.3](#), we have shown the three different types of class explanations which can be used during the generation of the summary. Moreover, the first row shows that if a class a single function then the generated summary of the class will be different than the one having two or more methods. Furthermore, we have shown the expected summaries of classes with more than one methods too. As the templates suggest, there is one pre-condition of this methodology that the naming the identifiers of variables, methods, classes and other constructs must be done in proper way. Else we won't be able to generate a valuable summary for the maintenance engineer.



Figure 4.7: Input screen for construct with in a construct

In [Figure 4.7](#), we have taken a code as an input in which there is conditional statement with in an iterative statement. Furthermore, iterative statement is within a conditional statement. Moreover, the method consisting of all these constructs has two parameters. One with data type of Boolean and the other data type is integer. A class with identifier “thermal_engine” has also been declared.

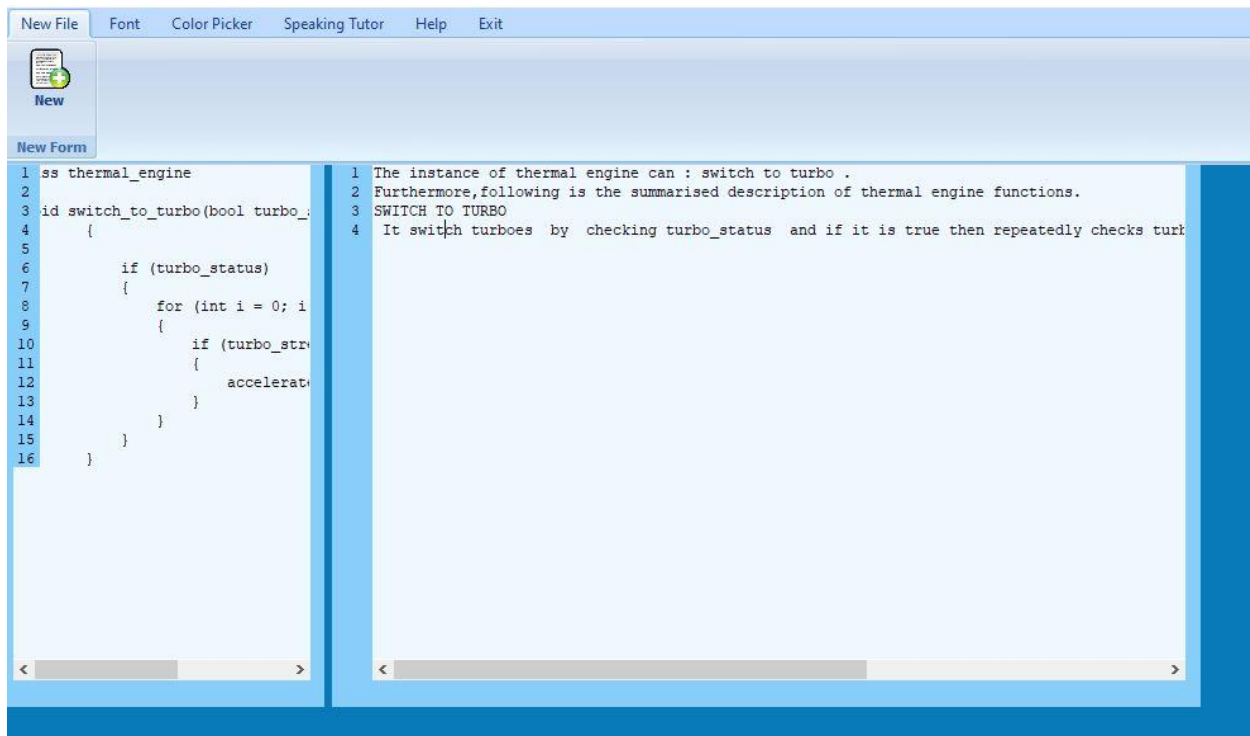


Figure 4.8: Output screen

The output screen has been show in [Figure 4.8](#). We have shown the generated summary below in different colors to identify the sections that were defined earlier.

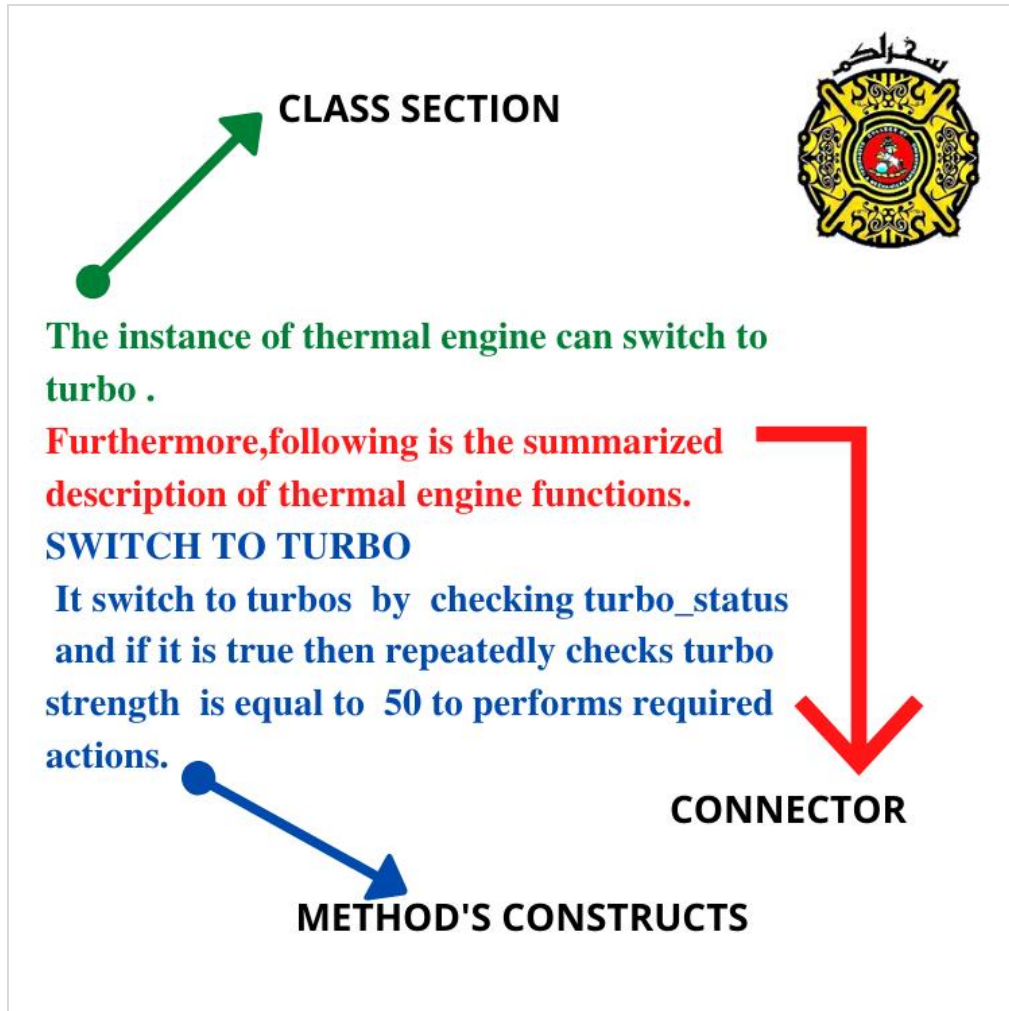


Figure 4.9: Diagram showing different sections of summary

We have mentioned the sections of the descriptions of the description in Figure 4.9. Although, the summary generated is helpful, it has some grammatical errors. The reason is the limitation of the natural language processing library we have used for generation of this summary. The accuracy of finding verbs and nouns is not 100%. Due to which, some of the nouns and verbs are misread and not identified properly. Furthermore, false identification of verbs and nouns creates problems for adding “s” or “es” with verbs. As in the example above show in Figure 4.9, in the method’s construct section, the sentence must start with “it switches”; however, it starts with “it switch” which degrades the quality of the statement for the readers. It is because the library we used for identifying the parts of the speech has misread switch as a noun whereas in the current context it’s a verb. Moreover, verbs in present indefinite tense must be used with “s” or “es”. Therefore, the generated summary has some errors.

Chapter 5

Validation

CHAPTER 5: VALIDATION

This chapter focuses on the validation of TCS by implementing it on three different case studies. The study of researchers have Analyzed that the quality assessment and validation of code comments is another important research problem which must be solved, as the quality of generated natural language summary or comments is an important indicator for evaluating whether a commenting methodology is effective or not. Furthermore, Generated natural language summary may describe not only the functions, but also the design intents of developers behind source code by mentioning the logic used the developer. Therefore, validation of generated summary is a process itself as programming languages are different from natural languages in nature. Moreover, source code consists of a lot of information about classes, member functions, member variables, parameters of methods, and at the same time has many nested structures or may contain nested structure with in a nested structure and complex relations; meanwhile, generated natural language summaries is written in natural languages are unstructured, and expressed freely inform.

Appropriate quality assessment metrics will lead to the abundance of a quantitative comparison that highlights the pros and cons of each source code summary generation algorithm[5]. So far, the criteria of quality assessment of Natural language summary are not same depending on the types of comments. For example, from the perspective of methods, summaries can be divided into different categories and placed in the category of descriptive comments generation, summary comments, conditional comments or description, comments for debugging and metadata comments, etc. In each category, each technique adopts different comment assessment criteria. Therefore, it is vital to design and formulate correct and relevant quality assessment metrics for generated comments or summary, which will help the study of automatic code comment generation.

In [Figure 5.1](#), we have shown the process used for validation and quality assessment of our developed tool and methodology. It has three parts, first, data is first one is data preparation which prepares data for the TCS as an input. This data is a source code, we are using three case studies / different source codes written in C# to be tested for validation. Secondly, the representation of source code, which aims at analyzing and making the structure and semantics of source code, such as information of structure and source code, lexis, grammar, semantics,

contexts, invocation relation and data dependency of source code. Thirdly, generation of natural language sentences or summary based on the information extracted from source code by using the process which we described in the last chapter. These studies will test and validate functionalities and results on the basis of different types of code used. Firstly, the first case study was tested on our implemented tool. This case study basically is a class with 17 member functions. Each member function has different functionality and requirement. Furthermore, let's implement the first case and validate the results via survey of reviews given by 27 technical people. Among these technical people, 15 are currently working as a software engineer in various identified I.T companies. We will be showing them results in the form of generated summaries and will ask them try to understand the code by reading our summaries. First, we will ask them to understand the code by using our summary, the provided code and measure their experience by their given responses.

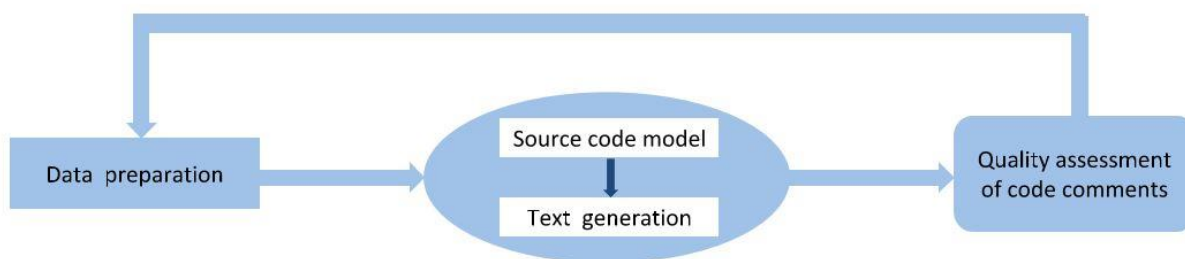


Figure 5.1: Quality Assessment Process of our Generated Source Code Summary

5.1. Case Study 1

To begin with, we wanted to test our code on a given source code. For doing so, we have selected a source code consisting of a single class and several member functions. The reason of doing so was to prove and identify the potential of implementation of TCS. Therefore, we added a number of 18 functions in the class to make it sure they are tested well by using our developed tool. Furthermore, let's identify the code which we are going to use to test the generated summary from that code. We tested a class with a total of 17 member functions. Moreover, each member function has different structure the other. This case study has been taken from online website source code repository named as "GitHub" [link]. Furthermore, in [Figure 5.2](#), the structure of a single class containing 17 member functions. As you can see in the [Figure 5.2](#),

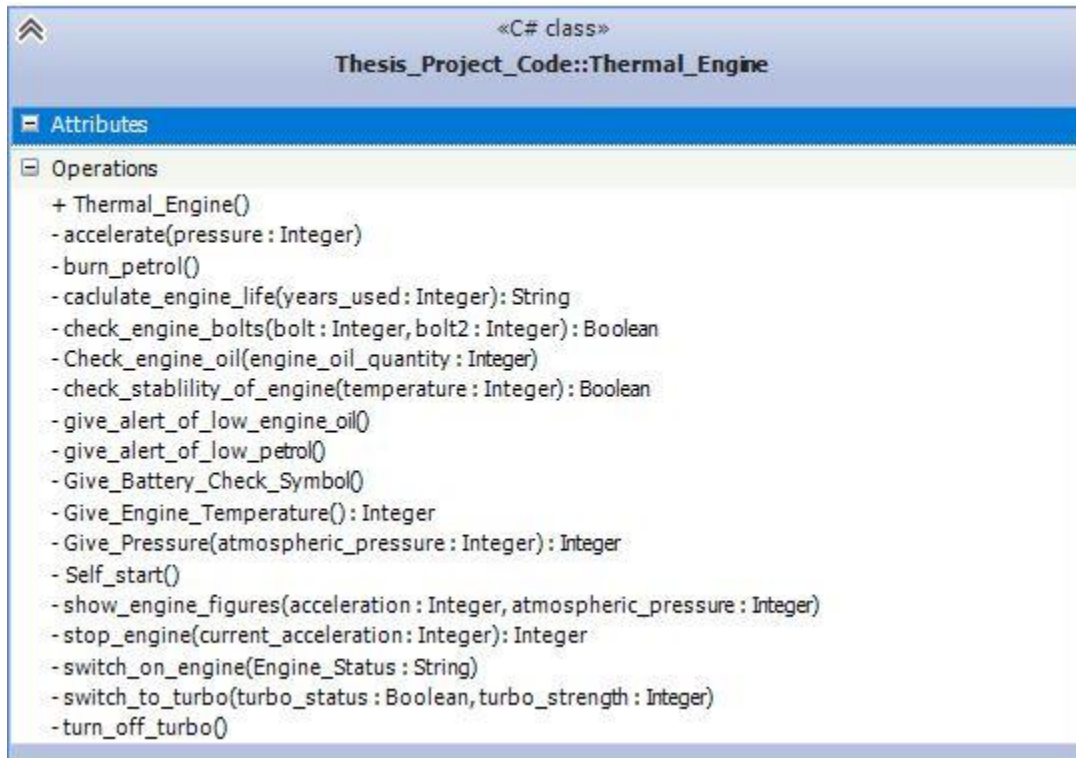


Figure 5.2: Class used in the case study

Some functions have parameters and some don't have. For example, the function “-accelerate(pressure:Integer)” is a private member function indicated by “-“and the identifier of function is “accelerate” and identifier of parameter is “pressure” of type integer.



Figure 5 3: Screenshot while submitting the code of first case study to our Tool

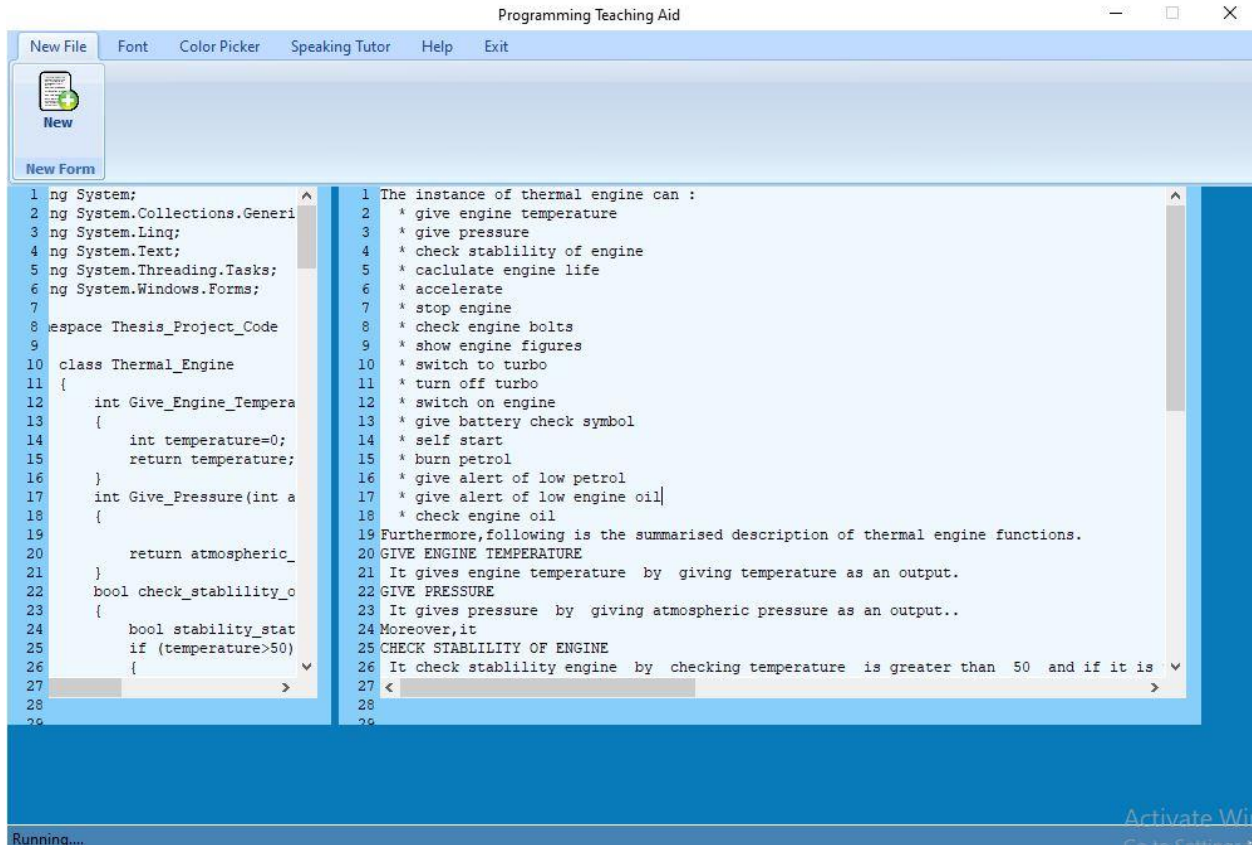


Figure 5.4: Screenshot showing generated summary of case study 1

The summary couldn't be shown in the output screen that is why we are going to show the generated summary here. Following is the generated summary shown by the system of whole source code. The blue part is basically the brief description of class "thermal_engine" whereas yellow is the connector line between first part of summary and the third part. The third part, in green, is brief description of member functions. The summary generated is as following:

"The instance of thermal engine can:

- * give engine temperature
- * give pressure
- * check stability of engine
- * calculate engine life
- * accelerate
- * stop engine

- * check engine bolts
- * show engine figures
- * switch to turbo
- * turn off turbo
- * switch on engine
- * give battery check symbol
- * self start
- * burn petrol
- * give alert of low petrol
- * give alert of low engine oil
- * check engine oil

Furthermore, following is the summarized description of thermal engine functions.

GIVE ENGINE TEMPERATURE

It gives engine temperature by giving temperature as an output.

GIVE PRESSURE

It gives pressure by giving atmospheric pressure as an output.

CHECK STABLILITY OF ENGINE

It check stability engine by checking temperature is greater than 50 and if it is true then it gives stability status as an output.

And giving stability status as an output.

CACLULATE ENGINE LIFE

It engine life by checking years used is lesser than 5 and giving life of engine as an output.

ACCELERATE

It accelerates by repetitively checking pressure is greater than 50 whether it is true or not to perform actions accordingly.

STOP ENGINE

It stop engine by repetitively checking acceleration is equal to 0 or not whether it is true or not to perform actions accordingly. And giving acceleration as an output.

CHECK ENGINE BOLTS

It check engine bolts by checking bolt is equal to 0 or not and if it is true then it gives engine bolt status as an output and giving engine bolt status as an output.

SHOW ENGINE FIGURES

It show engine figures by

SWITCH TO TURBO

It switch turboes by checking turbo status and if it is true then repeatedly checks turbo strength is equal to 50 to performs required actions.

TURN OFF TURBO

It turns turbo by repetitively checking leather pressure is greater than 50 whether it is true or not to perform actions accordingly.

SWITCH ON ENGINE

It switch engine by repetitively checking Engine Status is equal to (literal constant, "Off") whether it is true or not to perform actions accordingly. Checking Engine Status is equal to (literal constant, "Off")

GIVE BATTERY CHECK SYMBOL

It gives battery check symbol by checking battery color is equal to 0

SELF START

It self start by neither using any logical nor decision statements and performs some statements to fulfill its purpose.

BURN PETROL

It burns petrol by checking petrol quantity is greater than 100

GIVE ALERT OF LOW PETROL

It gives alert petrol by neither using any logical nor decision statements and performs some statements to fulfill its purpose .

GIVE ALERT OF LOW ENGINE OIL

It gives alert engine oil by

CHECK ENGINE OIL

It check engine oil by checking engine oil quantity is lesser than or equal to 50 and if it is true then it checks engine oil quantity is lesser than 30 to performs required actions.”

5.1.1. Human Evaluation

We have validated our code using human evaluation. Although, automatic evaluation techniques is more likely to give better results in terms of evaluation of generated summary , in our case the generated summary is collectively written and more lengthy than others. The reason is that we are briefly describing the use of iterative statements as well as conditional statements in our generated summary of source code.

Moreover, comment generation and summary generation are two different topics. As comment generation is generating individual statements and putting them after every code line where as in summary generation a whole paragraph is assembled and given in the top of the code as a comment. Therefore, summaries are lengthy than code comments as they are shown together.

Furthermore, researchers usually evaluate their generated summaries or code comments by human evaluators to evaluate the generated summaries / comments using assessment metrics.

Although, human evaluation's score is subjective and manual scoring has low efficiency, it is still one of the important ways for evaluating the performance of various commenting algorithms. However, there are many manual evaluation metrics for measuring and validating the quality of generated comments or summary. Although, quality assessment metrics are not identified with same names in various researches[5], we gather them into three different types according to their characteristics as follows. First, analyzing code comments / summary with their contents: evaluation of generated summary on contents, such as their adequacy, accuracy, conciseness, how much informative is the summary and interpretability. The meaning of various features in this group is described as follows.

- **Accuracy** is measurement of generated that how much it is closer to the semantics of the relevant source code.
- **Content adequacy** is used to measure how much there is difference between the generated summary meaning and the true meaning of the source code.
- **Conciseness** is used to measure how much unnecessary information is there in the generated summary or comments.

- **Informativeness** measures the how much information is generated summary is providing regarding source code.
- **Interpretability** measures to what extent the generated summary is interpretive and is conveying the meaning of source code.
- **Naturalness:** is used to measure the grammar and fluency of generated comments or generated summary.
- **Expressiveness** is used to measure how much the generated comments are expressive in their nature. Furthermore, the readability of the comments or summary is also measured under the umbrella of this feature.
- **Understandability** Is used to measure whether the generated comments are understandable by the reader or not.
- **Code understandability** is used to evaluate how much the reader understood the code after reading generated comments and generated summary.
- **Necessity** is used to evaluate and measure how much the generated summary is relevant and the information express was necessary.
- **Utility** defines how much the generated summary was helpful.

5.1.1.1. Evaluation of TCS

Surveys are important for analyzing a software system [54]. For the evaluation of TCS, we arranged a survey. In this survey we reached out software developers from software market. Each developer was given the summary to read and then the code. Later, they would be giving their opinion over different metrics we selected above for human evaluation.

5.1.1.2. Evaluation team

42 people participated in for the evaluation of our study. Among them, six were graduate students and remaining where post graduate students form National University of science and technology in Pakistan. Computer Science and Engineering Department of various university graduates participated in the evaluation.

. Among them, 32 are professionals and graduate students from three different universities, not listed due to our privacy policy. Two participants failed to respond enough of the study and we

had to decline their responses. These two participants were from N.U.S.T. (National university of science and technology) of Department at the University of Computer science. Furthermore, one of them was undergraduate and the other graduated student. Another participant only completed the survey on two summaries before leaving the survey. The remaining participants replied the questions on the survey in full.

5.1.1.3. Evaluation Questions

There were six question put in the survey. Furthermore, the answer of this question could be Strong agree, agree, disagree or strong disagree except for the last two questions that are Q7 and Q8. The participant has to answer for each statement by selecting one of the given choices. Following are the questions that were asked in the survey:

Q1: Independent of other factors, I feel that the summary is accurate.

Q2: The summary is missing necessary information, and that can help the understanding of the source code.

Q3: The summary consists of a lot of unwanted information.

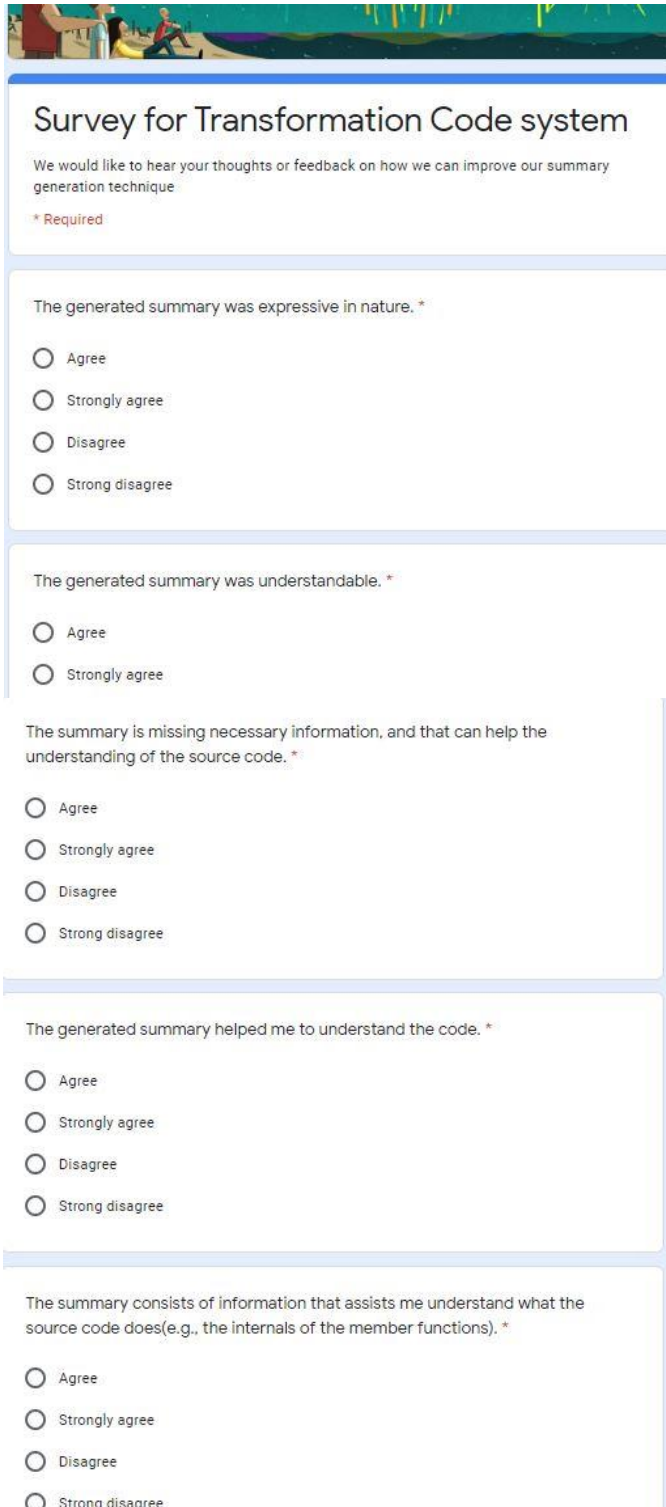
Q4: The summary consists of information that assists me understand what the source code does (e.g., the internals of the member functions).

Q5: The summary consists of information that assists me understand why the method exists in the source code (e.g., the results of changing or removing the method).

Q6: The summary consists of information that assists me understand how the code was written.

Q7: Please write a summary of code in your own words.

Q8: Do you have any comments about the auto-generated summary?



The image shows a survey form titled "Survey for Transformation Code system". At the top, there is a decorative banner with an illustration of people in a boat on water. Below the banner, the title "Survey for Transformation Code system" is displayed in a large, bold font. Underneath the title, a subtitle reads "We would like to hear your thoughts or feedback on how we can improve our summary generation technique". A red asterisk followed by the word "Required" indicates that the following questions are mandatory. The survey consists of five questions, each with four radio button options: "Agree", "Strongly agree", "Disagree", and "Strong". The questions are: 1. "The generated summary was expressive in nature. *", 2. "The generated summary was understandable. *", 3. "The summary is missing necessary information, and that can help the understanding of the source code. *", 4. "The generated summary helped me to understand the code. *", and 5. "The summary consists of information that assists me understand what the source code does(e.g., the internals of the member functions). *".

Survey for Transformation Code system

We would like to hear your thoughts or feedback on how we can improve our summary generation technique

* Required

The generated summary was expressive in nature. *

Agree

Strongly agree

Disagree

Strong disagree

The generated summary was understandable. *

Agree

Strongly agree

The summary is missing necessary information, and that can help the understanding of the source code. *

Agree

Strongly agree

Disagree

Strong disagree

The generated summary helped me to understand the code. *

Agree

Strongly agree

Disagree

Strong disagree

The summary consists of information that assists me understand what the source code does(e.g., the internals of the member functions). *

Agree

Strongly agree

Disagree

Strong disagree

Figure 5.5: Survey

In [Figure 5.5](#), we have shown some images of survey that was recorded in order to get responses from various users of our methodology. This survey was made in Google Forms. Furthermore, the participants of the survey have been mentioned above. We sent source of the first case study and the generated study in order to note their responses. So, we can have a better idea and evaluation of our used methodology. Moreover, the participant's responses were noted on Google forms too. These responses later on were formulated to find the compiled results. Each participants was asked to tell about their personal review too. In order to improve the methodology for future work. While having that survey we came to know some limitations of our methodology which are discussed below and will be improved in our future work.

Metrics and Statistical Tests

Each of the multiple choice questions provided by us which are mentioned above could be replied as "Agree", "Strongly agree", "Disagree", or "Strongly Disagree". We gave values to these replies as 4 for "Strongly Agree", similarly 3 for "Agree", 2 for "Disagree", and 1 for "Strongly Disagree." Moreover, for questions 1, 4, 5, and 6, bigger values show stronger performance. Furthermore, for questions 2 and 3, lower values were suggested. Moreover, we analyzed and summarized the responses for each statement by following a process. For example, all replies to question 1 for summaries of source code.

Response discussion

In the survey given to users we asked them to give comments about the generated summary and there were different comments and responses from the survey. One of them who was working as an android application developer as a freelancer and was student of Master's Degree in software engineering from NUST that "It was good enough to understand the flow of code". The previous response mentioned shows that the user thinks that it were good enough to make him understand the code. However, one of the participants (a software engineer graduated from National University of science and technology) gave comment that "I think your method should also provide a brief summary of overall program." It means that the user thinks that given methodology should also provide a brief summary of overall program too. Furthermore, one participant, a software engineer who currently working as an asp.net developer in Pakistan Revenue Automation (Pvt.) Ltd in Islamabad said "The summary has to be generated in

more detail for better understanding of functionalities and identify functional requirements”. So, this user thinks that there should be more details present in the generated summary to help user understand the code better. Another participant, a post graduate student from National University of Sciences and technology gave response that, “It used technical terms. It would have been better if it used terms for non-technical person too”. He means to say that in the generated summary there were too much technical words that were creating an obstacle in understanding of the code. However, we created the summary to make it sure that the maintenance engineers understand the code better. On the other hand, remaining participants gave responses which showed that they were helped by the generated summary in order to understand the code. One of the participant, an I.T. teacher of under-graduate students, gave comment, “The summary enhanced the understanding of code”, which shows that the generated summary helped the user in understanding the code.

5.1.1.4. Response Discussion

In [Figure 5.6](#), response of participants during survey are noted in a Pie chart. This pie charts show the results of each asked question from the user. Furthermore, the [Figure 5.6](#) also includes the responses given to each question by users. So, let’s discuss each response in detail. To begin with, in [Figure 5.6](#), responses has shown that 71.4% people disagree that the generated summary is missing necessary information. Furthermore, 19% agreed that there is some missing information in the generated summary.

The summary is missing necessary information, and that can help the understanding of the source code.
42 responses

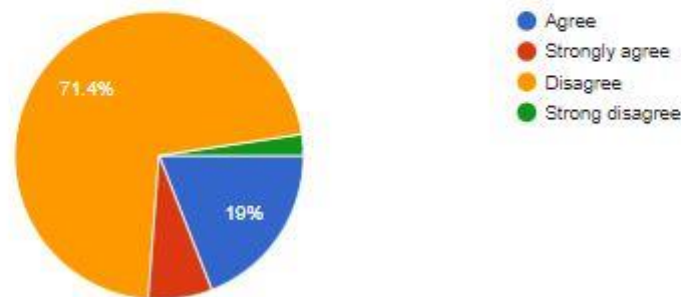


Figure 5.6: Responses of the users that participated in survey

In [Figure 5.7](#), we have shown the responses of the people who have participated in the survey. Furthermore, 64.3% strongly agree that the generated summary has helped them to understand the code. On the other hand, some the users have disagreed that the generated summary hasn't helped them in understanding the code.

The generated summary helped me to understand the code.

42 responses

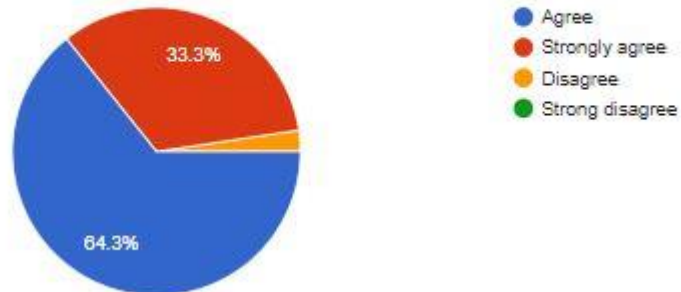


Figure 5.7: Responses of the users that participated in survey

The summary consists of information that assists me understand what the source code does(e.g., the internals of the member functions).

42 responses

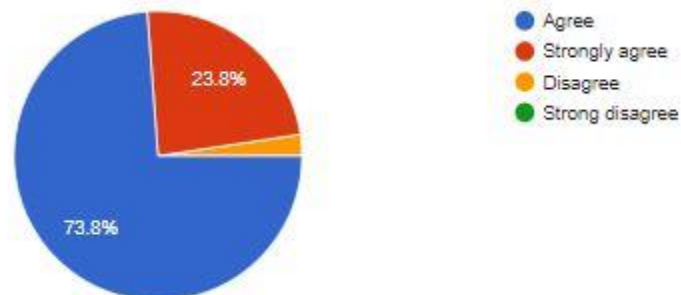


Figure 5.8: Responses of the users that participated in survey

The summary consists of information that assists me understand why the method exists in the source code (e.g., the results of changing or removing the method).

42 responses

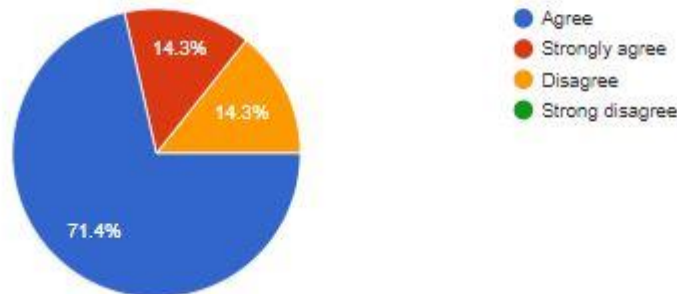


Figure 5.9: Responses of the users that participated in survey

As shown in Figure 5.9 14.3% People has disagreed the question that the generated summary has helped them in understanding why the method exists. It means most of them has got an understanding that why a specific method exists in the code by reading our generated summary.

The summary consists of information that assists me understand how the code was written.

42 responses

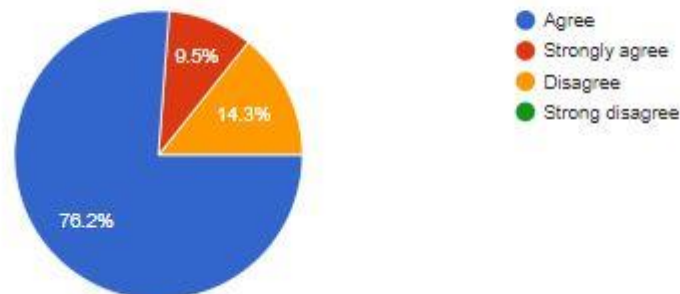


Figure 5.10: Responses of the users that participated in survey

In Figure 5.10, 14.3% disagree that they couldn't understand how the code was written from the generated summary. This provides us with a valuable information that there is need to improve the generated summary such that it is closer to plain natural language as much as it can. On the other hand, 76.2% understood how the code was written from the generated summary.

The generated summary was interpret-able .

42 responses

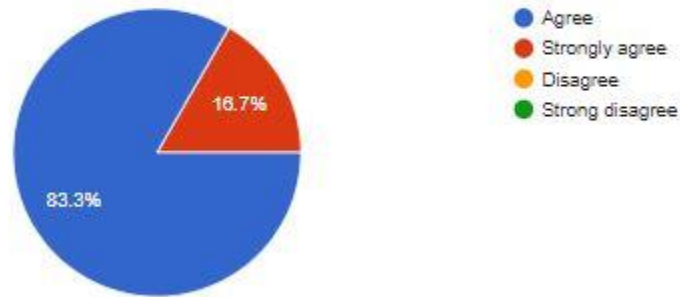


Figure 5.11: Responses of the users that participated in survey

Independent of other factors, I feel that the summary is accurate.

42 responses

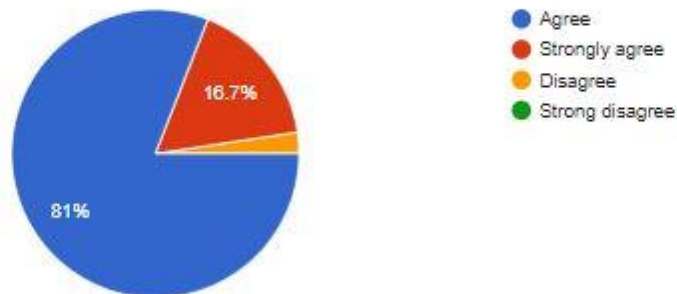


Figure 5.12: Responses of the users that participated in survey

The reason of any generated summary is that it must be readable. Therefore, as seen in [Figure 5.12](#), 81% thought that the generated summary was interpretable and it was easy for them to grasp the meaning of the generated summary. Furthermore, this is only possible if the generated summary is closer to natural language and the code is written well that is, the identifiers of the variables are named well enough.

The generated summary was concise.

42 responses

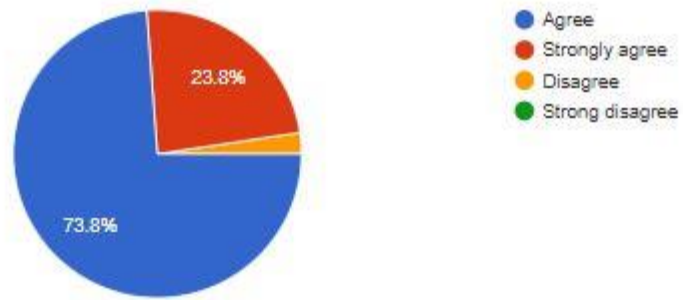


Figure 5.13: Responses of the users that participated in survey

In [Figure 5.13](#), 73% people think that the generated summary was concise. Moreover, 23.8% strongly agreed that the summary was concise. However, remaining thought that the summary wasn't concise at all. Therefore, we conclude from the above generated response that the generated summary is not perfect but it can be improved. Moreover, we have taken comments from the participants that how we can improve the generated summary. There were different responses, some of them thought it was perfect; however, most of them gave their critical review and enlightened us with many important points.

To begin with, some of the participants thought that the generated summary must have been less technical so non-technical people can also understand the summary. This was not possible in our case because we wrote the summary for maintenance engineers in order to ease them understand the code and understand the code faster than usual.

Secondly, some participants thought the summary should have been more explanatory and some other points of the loops should had been discussed.

Thirdly, some thought that the idea of using a single paragraph as a summary is not helpful and would be hectic. According to them, each line of code musts be commented separately which would be easier for them to understand the code. But there is a problem here, we wanted to generate a summary which would either make the maintenance engineer understand the code faster or help them understand the code. So, therefore these things are to be taken in mind for future work.

Chapter 6

Discussion and Limitation

CHAPTER 6: DISCUSSION AND LIMITATIONS

The **Section 6.1** consists of detailed discussion of proposed research work done by us; however, **Section 6.2** deals with the limitation of the research and of our methodology.

6.1. Discussion

In this research, we have proposed a methodology for generating source code summary for assistance in software maintenance of software maintenance engineers. Firstly, tokenization is done, which is complaint to C# syntax only in the current implementation. Secondly, data structures are produced of different tokens with the combination of graphs and stacks for identification of hierarchies in program constructs. Thirdly, a source code summary is generated by reading those data structures using templates and natural language processing side by side. This approach “TCS” is a huge achievement that provides brief detail of source code which can help in software development and maintenance both.

Literature provides strong evidence of various SCS (Source code summarization) frameworks and methodologies in software development. Moreover, researchers have given numerous higher abstraction level source code summaries to incorporate semantics of various programming languages i.e. *Java*, *Python* etc. For example, SWUM model was proposed for generation of natural language summary of Java Methods[44]. Similarly, Swan Rai, et al.[43] have presented Nano-patterns based approach to generate source code summary at higher abstraction level. They are successful in generating concise summaries. Furthermore, Yao Wan, , et al. [34], was successful in bringing the summaries of Java at higher abstraction level. Also, Xing Hu, et al.[37], Laura Moreno , et al.[2], Edmund Wong, et al.[42], Wenhao ZHENG., et al[36] have contributed in providing a source code summary generating methodologies. Moreover, many others have put their contribution in this field.

In aforementioned studies, researchers have given numerous methodologies to generate natural language summaries of different programming languages. In fact, most of them targeted Java. However, they have briefly and concisely given summaries but couldn't use the combination of constructs which can give a better overview of the source code. As a result, it leads to a summary or comment which either is not showing the complete understanding of the code or is so concise that it is missing necessary information. TCS completes this design flow, by generating summary

with a new methodology with the combination of various constructs like loops, conditional statements, methods, classes etc., with the help of them the summary is generated. Hence, research and academia can highly benefit from TCS, because it helps the programmer to understand the code in a better way.

It can be argued that why C# was targeted in our methodology and why were loops and conditional statements were used to generate the summary in order to understand the logic of code. We did so, because most of the researchers in past targeted Java and didn't include C# as their target language. Furthermore, addition of loops and condition statements in summary generation was done because they are key pillars to understand the flow of a method. Moreover, the usage of data structures such as graph of stacks was used to understand the hierarchy as well as the logic flow of the source code. Moreover, C# is highly used programming language and is also used now days in development of android applications and IOS application development too.

It can also be argued that Generated summary is highly dependent on the naming criteria of the programmer. Therefore, it only can generate a meaningful summary if the programmer names variables, methods and classes in proper manner. Similarly, conditional statement's conditions can only be interpreted if good programming practices are used. Because if naming is not done properly then the reader will face difficulty in understanding the code using the generated summary.

Source code summary generation is an important element in the modern's software development and software maintenance for creating ease in understanding of source of software systems. As discussed earlier, there is dire need to generate a meaningful, concise and understandable summary of code. Hence, TCS (Transformation code system) completes this process. On the other hand, this also gives us an opportunity to incorporate and combine different methodologies to gain the goals of generating a nice and readable summary.

Data structures we used for tokenization in our approach makes our proposed methodology able to generate more explainable natural language summary of source which briefly describes the programming constructs in the system but also enlightens the reader with the logic used in the methods. The model of our proposed work can easily be transformed and can be used to target any of the programming language like *Java*, *Python* etc. depending on its summary goals. Hence,

this methodology is highly scalable and configurable, which allows us to easily target new programming languages for summary generation. It also consists of qualities that leads to the high quality and less error-prone product.

In a nutshell, TCS is a Natural language based methodology which improves the summary generated by including the description of program flow along with various programming constructs in natural language. Moreover, this approach provides simplicity in the generated summary.

6.2. Limitations

This methodology leads to the automated generation of source code summary, but there are some limitations. The methodology is dependent on the programmer that his own grip of grammar of natural language is strong. That is, if he uses improperly names variables and identifiers then it won't be possible for this system to generate a meaningful summary for the reader. Moreover, the system is unable to generate summary for constructs that have more than 2 nested constructs. Although, the methodology supports up to unlimited number of constructs. Furthermore, the member variables of the class variables are not included in the generated summary which can also be used in future work for better explanatory summary.

Chapter 7

Conclusion and Future Work

CHAPTER 7: CONCLUSION AND FUTURE WORK

The proposed methodology provided by us gives a solution for automated source code summary generation of C# for better maintenance and understanding of source code in software system. It is based on natural language processing and stack-graphs to provide simple, readable, helpful, easy to understand summary of source code. The proposed summary generation approach **TCS (Transformation Code System)** provides brief and quality end summary with feature used for improving readability of generated summary. This reduces time and cost of development and maintenance of software systems. The results used during validation of our proposed methodology showed that the system worked in an expected way and was able to successfully generate natural language summary for source code.

In future we tend to extend TCS to add feature of understanding the code even if it isn't named properly. Moreover, we tend to add machine learning algorithms that can identify and give the functionality of overall software. Furthermore, we will adding more construct to generate more explanatory summary which can define the logic used by the programmer in a better way. We will be adding more features which will help. The summary has to be generated in more detail for better understanding of functionalities and for identifying functional requirements of software system too. Furthermore, we would be adding a feature which can detect and mention the list of libraries or packages that are used or imported. In future, we would be improving the template so they can also handle more complex source code.

REFERENCES

- [1]. Eddy, B.P., et al. *Evaluating source code summarization techniques: Replication and expansion*. in *2013 21st International Conference on Program Comprehension (ICPC)*. 2013.
- [2]. Moreno, L., et al. *Automatic generation of natural language summaries for Java classes*. in *2013 21st International Conference on Program Comprehension (ICPC)*. 2013.
- [3]. Sedano, T. *Code Readability Testing, an Empirical Study*. in *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. 2016.
- [4]. Haouari, D., H. Sahraoui, and P. Langlais. *How Good is Your Comment? A Study of Comments in Java Programs*. in *2011 International Symposium on Empirical Software Engineering and Measurement*. 2011.

- [5]. Song, X., et al., *A Survey of Automatic Generation of Source Code Comments: Algorithms and Techniques*. IEEE Access, 2019. **7**: p. 111411-111428.
- [6]. Pascarella, L., M. Bruntink, and A. Bacchelli, *Classifying code comments in Java software systems*. Empirical Software Engineering, 2019. **24**(3): p. 1499-1537.
- [7]. Yang, B., Z. Liping, and Z. Fengrong, *A Survey on Research of Code Comment*, in *Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences*. 2019, Association for Computing Machinery: Wuhan, China. p. 45–51.
- [8]. Steidl, D., B. Hummel, and E. Juergens. *Quality analysis of source code comments*. in *2013 21st International Conference on Program Comprehension (ICPC)*. 2013.
- [9]. Corazza, A., V. Maggio, and G. Scanniello. *On the Coherence between Comments and Implementations in Source Code*. in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*. 2015.
- [10]. Rahman, M.M., C.K. Roy, and I. Keivanloo. *Recommending insightful comments for source code using crowdsourced knowledge*. in *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2015.
- [11]. Zhao, L., L. Zhang, and S. Yan, *A Survey on Research of Code Comment Auto Generation*. Journal of Physics: Conference Series, 2019. 1345: p. 032010.
- [12]. Fernandes, E., et al., *How Clear is Your Code? An Empirical Study with Programming Challenges*. 2017.
- [13]. McBurney, P.W. and C. McMillan, *Automatic Source Code Summarization of Context for Java Methods*. IEEE Transactions on Software Engineering, 2016. 42(2): p. 103-119.
- [14]. Gelbukh, A. *Natural language processing*. in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*. 2005.
- [15]. Patten, T. and P. Jacobs, *Natural-language processing*. IEEE Expert, 1994. **9**(1): p. 35.
- [16]. Kłosowski, P. *Deep Learning for Natural Language Processing and Language Modelling*. in *2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*. 2018.
- [17]. Huang, L., S. Zhuang, and K. Wang, *A Text Normalization Method for Speech Synthesis Based on Local Attention Mechanism*. IEEE Access, 2020. **8**: p. 36202-36209.

- [18]. Ma, Q. *Natural language processing with neural networks*. in *Language Engineering Conference, 2002. Proceedings*. 2002.
- [19]. Kanan, T., et al. *A Review of Natural Language Processing and Machine Learning Tools Used to Analyze Arabic Social Media*. in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. 2019.
- [20]. Surabhi, M.C. *Natural language processing future*. in *2013 International Conference on Optical Imaging Sensor and Security (ICOSS)*. 2013.
- [21]. Khurana, D., et al., *Natural Language Processing: State of The Art, Current Trends and Challenges*. 2017.
- [22]. Li, T. and Q. Ren. *Research on development and application of computer supplementary software in PE teaching in Human Movement Science*. in *2010 2nd International Conference on Computer Engineering and Technology*. 2010.
- [23]. Chen, C., et al. *Why Is It Important to Measure Maintainability and What Are the Best Ways to Do It?* in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017.
- [24]. Liu, J., C. Zhou, and M. Tang. *Research on the Workday Time Management Model for Workflows in Business Service Grid Environment*. in *2006 Fifth International Conference on Grid and Cooperative Computing Workshops*. 2006.
- [25]. Rodeghero, P., et al., *Improving automated source code summarization via an eye-tracking study of programmers*, in *Proceedings of the 36th International Conference on Software Engineering*. 2014, Association for Computing Machinery: Hyderabad, India. p. 390–401.
- [26]. Singer, J. *Practices of software maintenance*. in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. 1998.
- [27]. Koponen, T. *Evaluation Framework for Open Source Software Maintenance*. in *2006 International Conference on Software Engineering Advances (ICSEA'06)*. 2006.
- [28]. Munro, M. *Software evolution research at the Centre for Software Maintenance*. in *IEE Colloquium on Managing Requirements Change: A Business Process Re-Engineering Perspective (Digest No. 1998/312)*. 1998.

- [29]. Sousa, M.J.C. and H.M. Moreira. *A survey on the Software Maintenance Process*. in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. 1998.
- [30]. Babu K, C., K. C, and S. N, *Entity based source code summarization (EBSCS)*. 2016. 1-5.
- [31]. Armaly, A., P. Rodeghero, and C. McMillan, *A Comparison of Program Comprehension Strategies by Blind and Sighted Programmers*. *IEEE Trans. Softw. Eng.*, 2018. **44**(8): p. 712–724.
- [32]. Yildiz, E. and E. Ekin. *Automatic comment generation using only source code*. in *2017 25th Signal Processing and Communications Applications Conference (SIU)*. 2017.
- [33]. Zhou, Y., et al., *Augmenting Java method comments generation with context information based on neural networks*. *Journal of Systems and Software*, 2019. **156**: p. 328-340.
- [34]. Wan, Y., et al., *Improving automatic source code summarization via deep reinforcement learning*, in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, Association for Computing Machinery: Montpellier, France. p. 397–407.
- [35]. Matskevich, S. and C.S. Gordon, *Generating comments from source code with CCGs*, in *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering*. 2018, Association for Computing Machinery: Lake Buena Vista, FL, USA. p. 26–29.
- [36]. Zheng, W., et al., *CodeAttention: translating source code to comments by exploiting the code constructs*. *Frontiers of Computer Science*, 2019. **13**(3): p. 565-578.
- [37]. Hu, X., et al., *Deep code comment generation with hybrid lexical and syntactical information*. *Empirical Software Engineering*, 2019.
- [38]. Badihi, S. and A. Heydarnoori, *CrowdSummarizer: Automated Generation of Code Summaries for Java Programs through Crowdsourcing*. *IEEE Software*, 2017. **34**(2): p. 71-80.
- [39]. Chen, M. and X. Wan. *Neural Comment Generation for Source Code with Auxiliary Code Classification Task*. in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. 2019.
- [40]. Liu, Y., et al. *Supporting program comprehension with program summarization*. in *2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS)*. 2014.

- [41]. Malhotra, M. and J.K. Chhabra. *Class Level Code Summarization Based on Dependencies and Micro Patterns*. in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. 2018.
- [42]. Wong, E., L. Taiyue, and L. Tan. *CloCom: Mining existing source code for automatic comment generation*. in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2015.
- [43]. Rai, S., et al. *Method Level Text Summarization for Java Code Using Nano-Patterns*. in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. 2017.
- [44]. Newman, C.D., et al. *Automatically Generating Natural Language Documentation for Methods*. in *2018 IEEE Third International Workshop on Dynamic Software Documentation (DySDoc3)*. 2018.
- [45]. Moreno, L., et al. *JSummarizer: An automatic generator of natural language summaries for Java classes*. in *2013 21st International Conference on Program Comprehension (ICPC)*. 2013.
- [46]. McMillan, C., et al. *Portfolio: finding relevant functions and their usage*. in *2011 33rd International Conference on Software Engineering (ICSE)*. 2011.
- [47]. Li, W., et al. *Toward Summary Extraction Method for Functional Topic*. in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2017.
- [48]. LeClair, A., S. Jiang, and C. McMillan. *A Neural Model for Generating Natural Language Summaries of Program Subroutines*. in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 2019.
- [49]. Haiduc, S., et al. *On the Use of Automated Text Summarization Techniques for Summarizing Source Code*. in *2010 17th Working Conference on Reverse Engineering*. 2010.
- [50]. Fowkes, J., et al. *TASSAL: Autofolding for Source Code Summarization*. in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 2016.
- [51]. Demartini, G. and S. Mizzaro. *A Classification of IR Effectiveness Metrics*. 2006. Berlin, Heidelberg: Springer Berlin Heidelberg.

- [52]. Fernández-Sáez, A.M., et al. *On the use of UML documentation in software maintenance: Results from a survey in industry*. in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2015.
- [53]. Granić, A. *Technology in use: The importance of good interface design*. in *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*. 2017.
- [54]. Ghazi, A.N., et al., *Survey Research in Software Engineering: Problems and Mitigation Strategies*. IEEE Access, 2019. **7**: p. 24703-24718.
- [55]. <https://endnote.com/>
- [56]. <https://visualstudio.microsoft.com/>
- [57]. <https://www.google.com/forms/about/>
- [58]. <https://docs.google.com/drawings/>
- [59]. <https://archive.codeplex.com/?p=sharpnlp>