AI Based Software Development Approach Over Traditional Software Development Approach in the Design Phase of Software Development Life Cycle

A MAJOR PROJECT REPORT

Submitted to

ASSAM DON BOSCO UNIVERSITY

By

JEDIAEL MESHUA SUMER

Roll No. DC2016BTE0139

MARLOM BEY

Roll No. DC2016BTE0195

MRIGANKA SHEKHAR SARMAH

Roll No. DC2016BTE0202

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SCHOOL OF TECHNOLOGY, ASSAM DON BOSCO UNIVERSITY

AZARA, GUWAHATI 781017, ASSAM, INDIA.

BATCH (2016- 2020)

CERTIFICATE

This is to certify that the project report entitled **"AI BASED SOFTWARE DEVELOPMENT APPROACH OVER TRADITIONAL SOFTWARE DEVELOPMENT APPROACH IN THE DESIGN PHASE OF SOFTWARE DEVELOPMENT LIFE CYCLE"** submitted by Jediael Meshua Sumer (DC2016BTE0139), Marlom Bey (DC2016BTE0195) and Mriganka Shekhar Sarmah (DC2016BTE0202) to the Assam Don Bosco University, Guwahati, Assam, in partial fulfilment of the requirement for the award of Degree of Bachelor of Technology in Computer Science and Engineering is a bonafide record of the project work carried out by them under my supervision during the year 2019-2020.

Supervisor:
Dr. Bobby Sharma,
Head of Department,
Department of Computer Science & Engineering,
School of Technology,
Assam Don Bosco University.

CERTIFICATE

This is to certify that the Project Report entitled **“AI BASED SOFTWARE DEVELOPMENT APPROACH OVER TRADITIONAL SOFTWARE DEVELOPMENT APPROACH IN THE DESIGN PHASE OF SOFTWARE DEVELOPMENT LIFE CYCLE”** submitted by Jediael Meshua Sumer (DC2016BTE0139), Marlom Bey (DC2016BTE0195) and Mriganka Shekhar Sarmah (DC2016BTE0202) respectively to the Assam Don Bosco University, Guwahati, Assam, in partial fulfilment of the requirement for the award of Degree of Bachelor of Technology in Computer Science and Engineering is a bonafide record of the project work carried out by them during the year 2019-2020.

Dr. Bobby Sharma	Prof. Manoranjan Kalita
Head of the Department	Director, School of Technology
Dept. Of CSE, School of Technology,	Assam Don Bosco University
Assam Don Bosco University	
Date:	Date: ....

EXAMINATION CERTIFICATE

This is to certify that **Jediael Meshua Sumer**, **Marlom Bey** and **Mriganka Shekhar Sarmah** bearing Roll Number **DC2016BTE0139**, **DC2016BTE0195** and **DC2016BTE0202** respectively of the Department of Computer Science & Engineering has carried out the project work in a manner satisfactory to warrant its acceptance and also defended it successfully.

We wish them all the success in their future endeavours.

Examiners:

01. External Examiner:

02. Internal Examiner:

03. Internal Examiner:

DECLARATION

We hereby declare that the dissertation work entitled **“AI BASED SOFTWARE DEVELOPMENT APPROACH OVER TRADITIONAL SOFTWARE DEVELOPMENT APPROACH IN THE DESIGN PHASE OF SOFTWARE DEVELOPMENT LIFE CYCLE”** submitted to the Assam Don Bosco University, Guwahati, Assam, in partial fulfilment of the requirement for the award of Degree of Bachelor of Technology in Computer Science and Engineering is an original work done by us under the guidance of Dr. Bobby Sharma (Head of Department, Department of Computer Science & Engineering, School of Technology, Assam Don Bosco University) and has not been submitted for the award of any degree.

Jediael Meshua Sumer
DC2016BTE0139
Department of Computer Science & Engineering
School of Technology, Assam Don Bosco University.

Marlom Bey
DC2016BTE0195
Department of Computer Science & Engineering
School of Technology, Assam Don Bosco University.

Mriganka Shekhar Sarmah
DC2016BTE0202
Department of Computer Science & Engineering
School of Technology, Assam Don Bosco University.

ACKNOWLEDGEMENT

We would like to take this opportunity to extend our heartiest gratitude to our guide and supervisor, Dr. Bobby Sharma, Associate Professor, Department of Computer Science & Engineering, School of Technology, Assam Don Bosco University, whose constant guidance and encouragement made the progress of our B.Tech Major Project possible. Also we would like to thank her for giving us this golden opportunity.

We would also like to express our gratitude to our Project Coordinators, Mr. Alok Choudhury, Assistant Professor, Department of Computer Science & Engineering, School of Technology, Assam Don Bosco University, and Mr. Syed Sazzad Ahmed, Assistant Professor, Department of Computer Science & Engineering, School of Technology, Assam Don Bosco University, for their constant support and guidance.

We are obliged to all professors of the Department of Computer Science & Engineering, School of Technology, Assam Don Bosco University, for instilling in us the basic knowledge about the field that greatly benefitted us while carrying out the project and achieving the goal.

Lastly, we are grateful to our friends, for their relentless support in augmenting the value of work; our families, for being considerate and appreciative throughout; and the Almighty for everything.

ABSTRACT

Artificial intelligence has had a widespread impact in almost every aspect of human life, however, the field of software engineering and the design phase of software engineering in particular has experienced some negligence towards incorporating artificial intelligence which, in theory, should make the development of a software easier, faster and more accurate since human error is eliminated. This research aims to introduce artificial intelligence to the design phase of software engineering and draw a comparison between it and the traditional method of software engineering.

KEYWORDS: *Artificial Intelligence, Software Engineering, Class Diagram Generator, Design Phase in Software Engineering.*

List of Tables

List of Figures

Table of Contents

CHAPTER 1
INTRODUCTION

1.1 PROJECT TITLE:

AI based software development approach over traditional software development approach in the Design Phase of Software Development Life Cycle.

1.2 OBJECTIVE:

The primary objective of this research is to draw a comparison between the use of artificial intelligence in the design phase of software development against the traditional approach which leans towards being manual in nature.

To perform the above research, we aim to develop a class diagram generator which utilises natural language processing (NLP) to transform user requirements in the form of simple sentences into class diagrams, and thus draw a comparative study between the whole process against the traditional "pen and paper" method.

1.3 PROBLEM STATEMENT:

With advancements made in the field of artificial intelligence and their application in every aspect of our lives, it comes to no surprise that software development has become a lot easier with numerous tools which implement artificial intelligence that help a developer in developing a software. The aim of this project is to measure the exact quantity of the effect of using artificial intelligence in software engineering. The effects of artificial intelligence can be categorised accordingly:

1. Optimal level of automation[2]
2. Cost of project development using AI
3. Time spent on project development using AI
4. Accuracy of the generated diagrams

We aim to achieve the task above by developing our own class diagram generator[3] which greatly utilise artificial intelligence to develop a software and thus use the developed tool to draw a comparative study on the effects that artificial intelligence has on software development, as compared to the traditional way of software development.

1.4 MOTIVATION AND SCOPE:

The motivation for taking this project comes from the growing rise in demand for artificial intelligence to make our lives easier. As with many other fields, artificial intelligence has seen a growing rise in popularity in the field of software development. With that in mind, our interest was to study by how much artificial intelligence impacts the work of a software developer.

The scope of this research is to introduce artificial intelligence in software development so that we can increase the efficiency of software development with reduced cost and effort.

CHAPTER 2
LITERATURE REVIEW

In [1], the various commonalities between the fields of AI and SE, current status and upcoming trends. Intersections between AI and SE are discussed, especially in the fields of Agent-Oriented Software Engineering, Knowledge Based Engineering Systems, Computational Intelligence and KDD and Ambient Intelligence. It also gives a short description about AI and software engineering, and how the possibilities of interaction with one another through various common points of contact.

In [2], the differences between the software engineering process and AI, and the hostilities that exists in software engineers towards using automation in developing a software are discussed. It further discusses the contribution of AI to the field of SE in fields like automatic programming and some projects like REFINE and Programmer's Apprentice (an MIT project) and the problem areas of SE where AI can be implemented like requirements analysis and definition, process modelling and process support and project planning. The author states one of the reasons for the existing hostility towards AI is that AI researchers are themselves software engineers, and therefore their method to integrating AI in software engineering follows a typical and traditional approach which precedes the advancements that AI and its endless possibilities can bring to the field of software engineering.

In [3], a survey on the application of artificial intelligence approaches to the software engineering processes is done. These approaches can have a major impact on reducing the time to market and improving the quality of software systems in general. Existing survey papers are driven by the AI techniques used, or are focused on specific software engineering processes. This paper relates AI techniques to software engineering processes specified by the IEEE 12207 standard of software engineering. Some of the tools discussed are:

1. SPECIFIER, a CASE based system that takes as input an informal specification of an operation
2. ATGen, a software test data generator
3. Class-Model Builder (CM Builder), a natural language based CASE tool that builds class diagrams specified in UML from natural language requirements documents

In [4], a taxonomy called AI in SE Application Levels or AI-SEAL that classifies 15 papers from previous editions of the RAISE workshop is discussed. It considers the context in which AI is being applied, I.e. "when" and "on what" AI is being applied. After classification the papers also assigns levels of automation to the papers.

In [5], a brief overview of SE and expert systems in artificial intelligence and how expert systems can be used to automate the programming process and code generation via the use of genetic programming is given. This paper also shows the absence of risk management in AI based systems due to the way they work.

As discussed in [6], translation of user requirements into design diagrams is a daunting task for a designer as that person has to translate textual requirements into a diagrammatic (UML) form. This paper gives a basic overview of existing systems regarding and how they convert the user requirements using NLP into UML diagrams. They developed their own tool DC Builder and presented in the paper a diagrammatic representation and a heuristic rule set in implementing NLP on user requirements. Finally they evaluated and compared the various tools mentioned and developed with one another.

In [7], the extraction of UML diagrams from textual requirements by requirements engineers is given as a daunting task and the time and effort spent on this justifies a tool to automate this process which brings the author to propose a tool called RAPID which uses NLP in an efficient manner to extract design diagrams from input textual user requirements. The methodology states various NLP technologies, algorithms and rules to extract class information from the textual requirements.

In [8], the ambiguity of textual requirements and the usage of NLP and domain ontology to generate UML diagrams from the said textual requirements is discussed. This paper also talks about existing systems and then approaches the problem using their own method called RAUE to extract UML diagrams from textual requirements.

In [9], existing systems which translate textual requirements into UML diagrams is discussed, after which the author proposes his own approach for RACE which is an improvement on the TCM system. The methodology of the RACE system is discussed with its algorithms and rules to identify classes, attributes and relationships. The author concludes the paper stating RACE being an advanced approach towards extraction of UML diagrams from textual requirements using a human-centred UI and what the system did not support.

In [10], an automatic test data generator called ATGen is introduced. ATGen is based on constraint logic programming and symbolic execution. Developing softwares include a number of testing methods. The main technique that is currently used in the industry is dynamic software testing where the software is executed using test data. Dynamic testing can be performed using automatic tools, which are - automation of administrative tasks, automation of mechanical tasks and automation of test generation tasks.

In [11], an approach towards extracting UML diagrams from natural language using NLP via a tool called CM Builder is given. CM Builder is a graphical CASE tool that does surface analysis of text to propose candidates for class, attributes and relationship and domain independent semantic analysis to automatically extract the candidates and finally represent them via UML diagrams. The tool also includes capacity to evaluate its candidate classes. The author also states benefits of object oriented analysis of the tool and the scope of improvement for it.

The session as discussed in [12] explored the reasons for the lack of impact in important areas in which AI has been expected to significantly affect real world Software Engineering. The session approached the failures of AI in software engineering, looking at the matter through a common cause- reliance on isolationist technology and approaches, rather than upon creating additive technology and approaches that can be integrated with other existing capabilities. The isolationism has been manifested in several areas, in essence, the market has rejected the isolationist and egocentric approach to implementing AI in software engineering, and that the whole system should be developed and executed in generalized workstations and PCs.

As seen in [13], recent state-of-the-art approaches automatically generate regular expressions from natural language specifications. Given that these approaches use only synthetic data in both training datasets and validation/test datasets, a natural question arises: are these approaches effective to address various real-world situations? To explore this question, in this paper, a characteristic study on comparing two synthetic datasets used by the recent research and a real-world dataset collected from the Internet, and an experimental study on applying a state-of-the-art approach on the real-world dataset is conducted. The study results suggest the existence of distinct characteristics between the synthetic datasets and the real-world dataset, and the state-of-the-art approach (based on a model trained from a synthetic dataset) achieves extremely low effectiveness when evaluated on real-world data, much lower than the effectiveness when evaluated on the synthetic dataset.

In [14], previous work done in the field of converting natural language requirements into design diagrams is looked upon, as well as the various issues we can encounter in the process.The author points out the issues in the context of software engineering, then in the context of natural language and finally proposes a methodology to solve these problems to get proper quality UML diagrams.

Paper [15] introduces artificial intelligence to software engineers and conversely, software engineering to artificial intelligence workers. It further highlights the contrast between the two fields and further accentuates their differences in the problems that they attempt to solve, their methodologies and the tools and techniques. The author strongly believes that the work of software engineers and artificial intelligence workers are similar, and the fusion of the two fields is essential for Computer Science as a field to move forward, however, the author also acknowledges the ridge that exists between the two communities. The paper does not however provide the '*how*' on which AI and software engineering can come together, other than details of the core components of both fields. The paper acknowledges that to move forward with 'user-friendliness', the system needs to allow natural language input and output, and natural language processing is one of the most researched areas in AI. In other words, a system needs AI to further improve its usability. Conversely, AI is in need of proper models to address its problems. These models are already evident in conventional systems, and it is clear that AI systems need some standardization in order to widen their applications. Thus, there exists a requirement for both fields to merge on some common ground.

CHAPTER 3
FEASIBILITY STUDY AND REQUIREMENT ANALYSIS

3.1 SYSTEM REQUIREMENTS (HARDWARE AND SOFTWARE SPECIFICATIONS)

3.1.1 SOFTWARE REQUIREMENTS:

- Windows 7 or later or MacOS 10.10 or higher [17]
- Programming languages: Python, JavaScript
- Markup languages: HTML, CSS
- Anaconda Navigator 1.9.7
- Visual Studio 1.42.1

3.1.2 HARDWARE REQUIREMENTS:

- Processor: Intel Core i5 [17]
- RAM: 3 GB or more [17]
- Disk Space: 1.5 GB or more [17]

3.2 SCHEDULE FEASIBILITY:

3.2.1 WORK BREAKDOWN STRUCTURE:

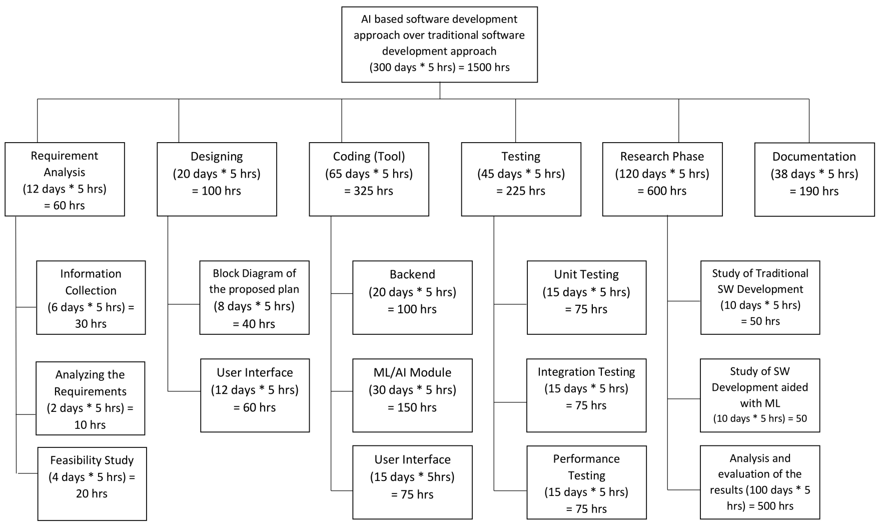
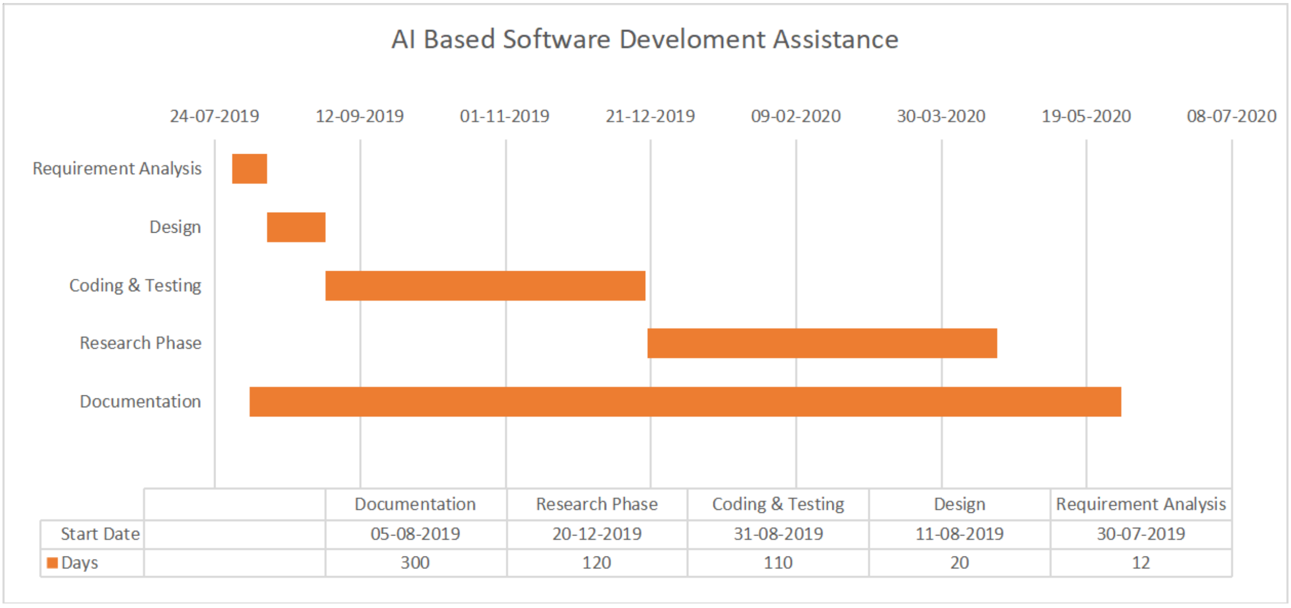

AI based software development approach over traditional software development approach (300 days * 5 hrs) = 1500 hrs
Requirement Analysis (12 days * 5 hrs) = 60 hrs
Designing (20 days * 5 hrs) = 100 hrs
Coding (Tool) (65 days * 5 hrs) = 325 hrs
Testing (45 days * 5 hrs) = 225 hrs
Research Phase (120 days * 5 hrs) = 600 hrs
Documentation (38 days * 5 hrs) = 190 hrs
Information Collection (6 days * 5 hrs) = 30 hrs
Analyzing the Requirements (2 days * 5 hrs) = 10 hrs
Feasibility Study (4 days * 5 hrs) = 20 hrs
Block Diagram of the proposed plan (8 days * 5 hrs) = 40 hrs
User Interface (12 days * 5 hrs) = 60 hrs
Backend (20 days * 5 hrs) = 100 hrs
ML/AI Module (30 days * 5 hrs) = 150 hrs
User Interface (15 days * 5hrs) = 75 hrs
Unit Testing (15 days * 5 hrs) = 75 hrs
Integration Testing (15 days * 5 hrs) = 75 hrs
Performance Testing (15 days * 5 hrs) = 75 hrs
Study of Traditional SW Development (10 days * 5 hrs) = 50 hrs
Study of SW Development aided with ML (10 days * 5 hrs) = 50
Analysis and evaluation of the results (100 days * 5 hrs) = 500 hrs

Fig 3.1: Work Breakdown Structure

3.2.2 GANTT CHART:



	Documentation	Research Phase	Coding & Testing	Design	Requirement Analysis
Start Date	05-08-2019	20-12-2019	31-08-2019	11-08-2019	30-07-2019
🟧 Days	300	120	110	20	12

Fig 3.2: Gantt Chart

3.3 ECONOMIC FEASIBILITY AND OPERATIONAL FEASIBILITY:

3.3.1 COCOMO MODEL:

The basic Constructive Cost Model (COCOMO) [16] equation will take the following form:

Effort Applied (E)= $a_b(KLoC)^{b_b}$ [person-months]

Development Time (D)= $c_b(\text{Effort Applied})^{d_b}$ [months]

People Required (P)=Effort Applied/Development time [count]

Where KLoC is the estimated number of delivered lines (expressed in thousands) of code

The coefficients a_b b_b c_b d_b are given by:

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 3.1: COCOMO model coefficient values

Taking the project type to be semi-detached, we get the following calculations:

Effort Applied, E= $a_b(\text{KLoC})^{b_b}$ [person-months]

E=$3.0*(4K)^{1.12}$ PM

E=14.17 PM

Development Time, D= $c_b(\text{Effort Applied})^{d_b}$ [months]

D=$2.5*(14.17)^{0.35}$ months

D=6.32 months

People Required, P=Effort Applied/Development time [count]

P=14.17/6.32 count

P=2.24 count

P≈3 count

CHAPTER 4
PROPOSED PLAN

4.1 METHODOLOGY:

The research can be carried out in the following steps:

1. Develop class diagram generator which utilises artificial intelligence in the form of natural language processing.
2. Define the traditional method of software development.
3. Define the sample space for the research.
4. Carry out the research and draw a comparison between the use of artificial intelligence, and the traditional method in software development.

4.2 DATASET DESCRIPTION:

Not applicable.

4.3 PERFORMANCE EVALUATION METRICS:

The evaluation of the effects of artificial intelligence in software engineering can be evaluated on the basis of the following fields:

1. **Level of automation:** It is defined as the degree to which a task is automated. There exists a conflict between software engineers and artificial intelligence[2] in the field of software engineering due to the level of freedom that automation deprives a software engineer, and due to the various risks involved in automating the software development process. The research will attempt to find an optimum level of automation in software engineering.
2. **Time:** The research will attempt to find the time saved by incorporating artificial intelligence in the software development process.
3. **Cost of development:** The research will attempt to find the cost which is saved by incorporating artificial intelligence in software engineering.
4. **Accuracy:** The research will attempt to calculate the accuracy of the generated diagrams, which have been generated using artificial intelligence.

4.4 BLOCK DIAGRAM:

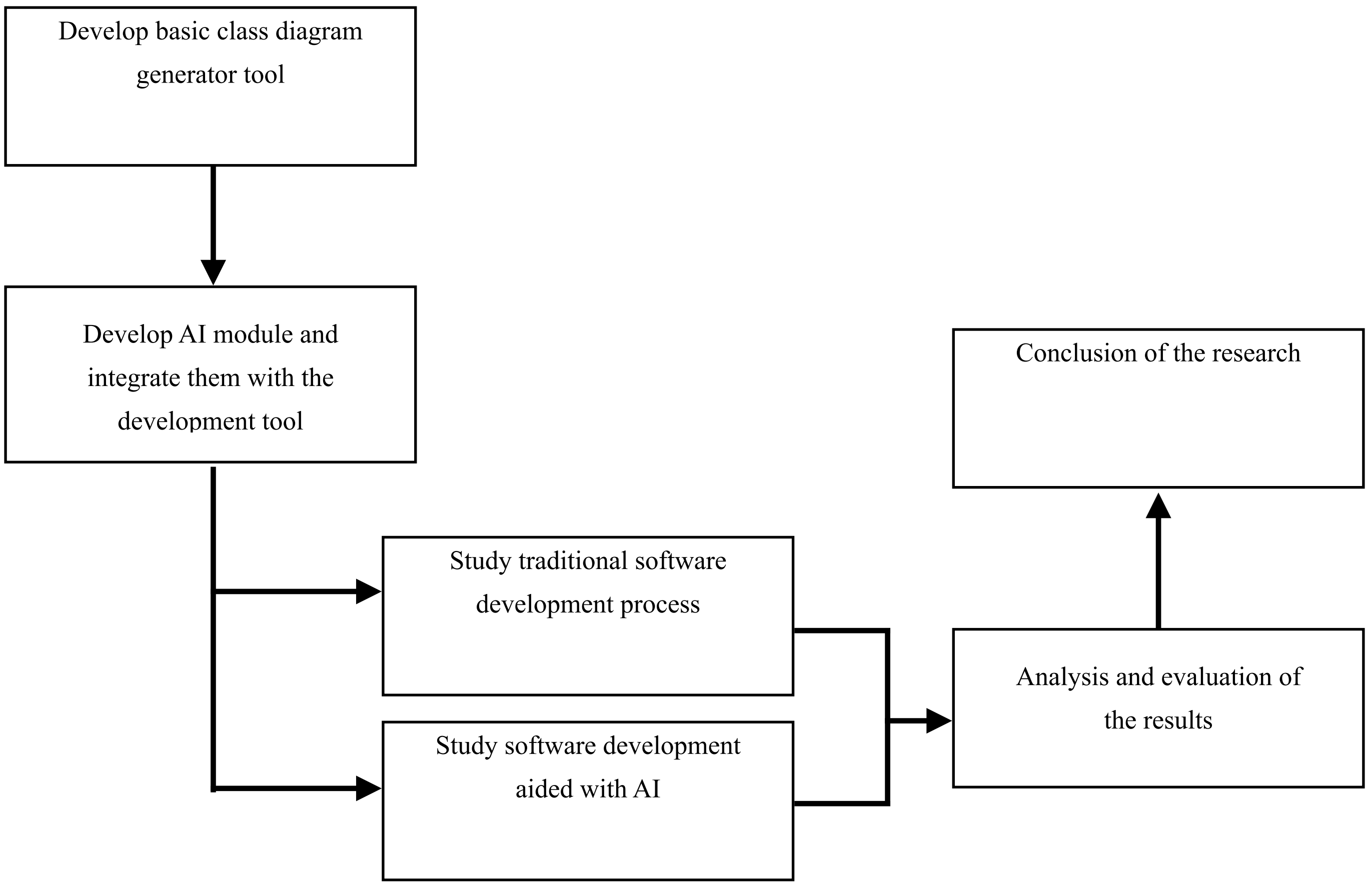

Develop basic class diagram generator tool
Develop AI module and integrate them with the development tool
Study traditional software development process
Study software development aided with AI
Analysis and evaluation of the results
Conclusion of the research

Fig 4.1: Block diagram of the proposed plan

CHAPTER 5
IMPLEMENTATION

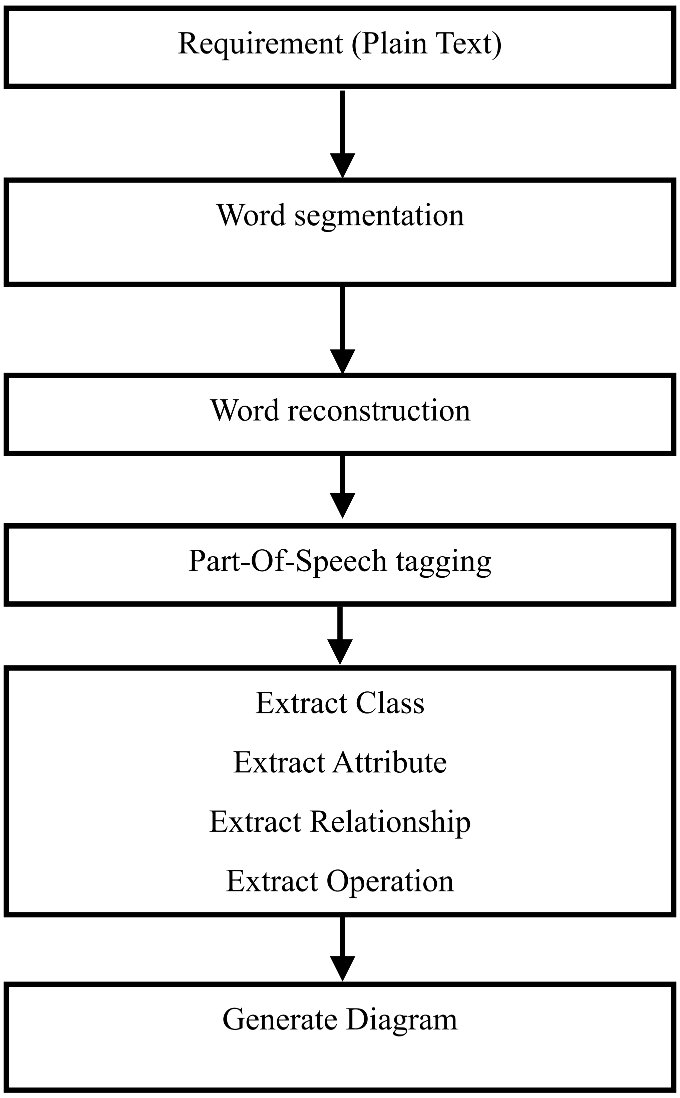


Fig 5.1: Implementation Procedure Block Diagram

1. **Requirement:** This is the plain text that is sent as input by the user to the program, from which we can generate the class diagram.
2. **Word reconstruction:** Requirements are preprocessed and re-constructed where texts are split into individual sentences.
3. **Word Segmentation:** After pre-processing the requirements, NLTK will attempt to segment the sentences into distinct keywords.
4. **Part-of-Speech tagging:** To process natural languages, part-of-speech tagging is considered a basic step. This step tags each word in a sentence with different categories. The categorisation

of parts of speech can vary depending on how precisely one wants to tag. In this paper, parts of speech are divided into seven categories: (1) nouns, (2) pronouns, (3) verbs, (4) adverbs, (5) prepositions, (6) conjunctions, (7) interjections.

5. **Extract class:** From the processed requirements, extract the class names that will form the basis of the class diagram.
6. **Extract Attribute:** From the processed requirements, extract the attributes associated with each extracted class.
7. **Extract Relationship:** Define the relationships between the different classes from the processed requirements.
8. **Extract Operation:** Define the various methods in the class from the processed requirements.
9. **Generating the diagram:** The class diagram can then be generated using Pygame from the processed requirements after the above steps have been completed.

5.1 SYSTEM ARCHITECTURE

The coding aspect of the project can be represented by the following diagram:

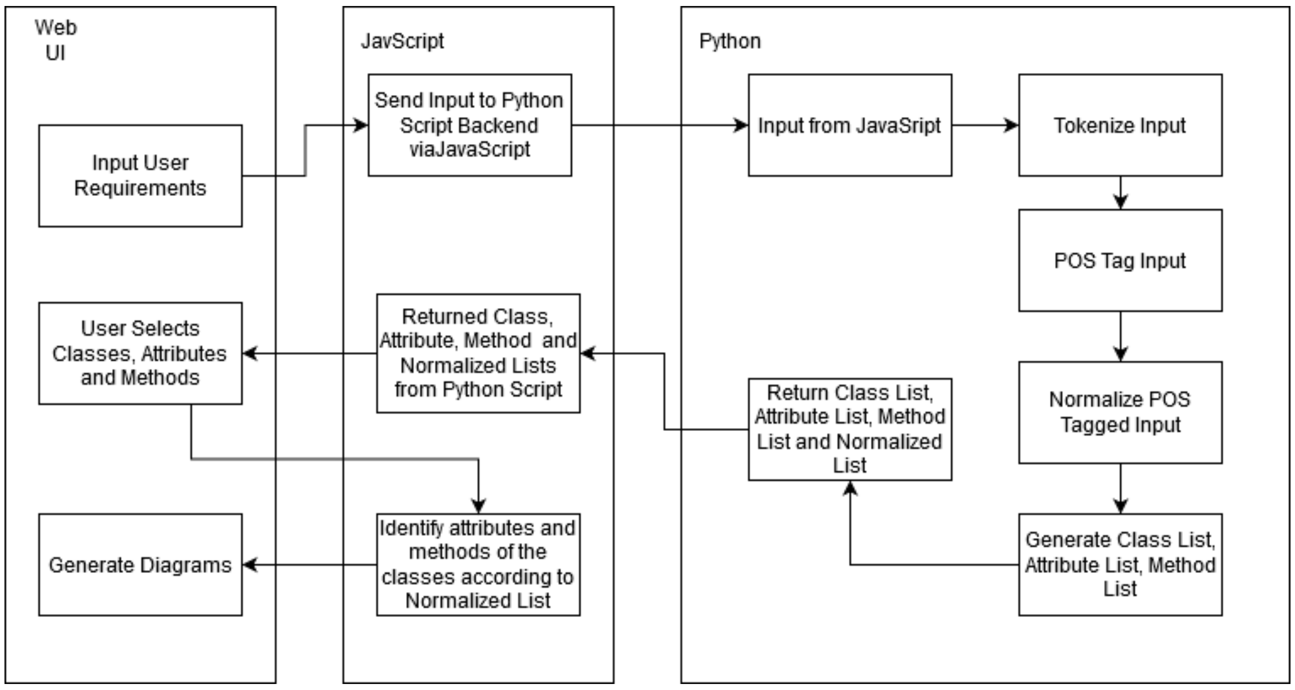

Web UI
Input User Requirements
User Selects Classes, Attributes and Methods
Generate Diagrams
JavScript
Send Input to Python Script Backend viaJavaScript
Returned Class, Attribute, Method and Normalized Lists from Python Script
Identify attributes and methods of the classes according to Normalized List
Python
Input from JavaSript
Tokenize Input
POS Tag Input
Normalize POS Tagged Input
Generate Class List, Attribute List, Method List
Return Class List, Attribute List, Method List and Normalized List

Fig 5.2: System architecture block diagram

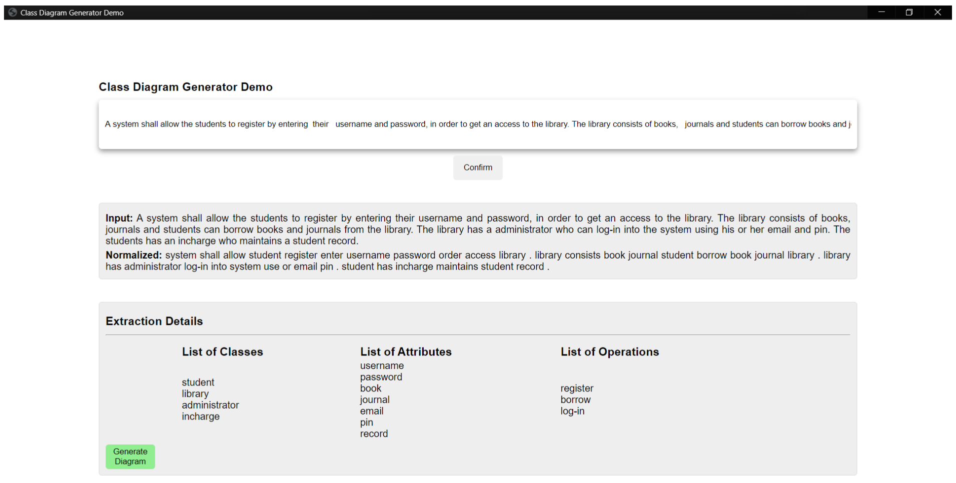
5.2 LIBRARIES USED

1. **Natural Language Toolkit (NLTK):** The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language

processing (NLP) for English written in the Python programming language. NLTK includes graphical demonstrations and sample data. NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning. NLTK supports classification, tokenisation, stemming, tagging, parsing, and semantic reasoning functionalities.

2. **Python Eel:** Eel is a little Python library for making simple Electron-like offline HTML/JS GUI apps, with full access to Python capabilities and libraries. Eel is designed to take the hassle out of writing short and simple GUI applications.

5.3 WORKING



Class Diagram Generator Demo

A system shall allow the students to register by entering their   username and password, in order to get an access to the library. The library consists of books,   journals and students can borrow books and j

Input: A system shall allow the students to register by entering their username and password, in order to get an access to the library. The library consists of books, journals and students can borrow books and journals from the library. The library has a administrator who can log-in into the system using his or her email and pin. The students has an incharge who maintains a student record.

Normalized: system shall allow student register enter username password order access library . library consists book journal student borrow book journal library . library has administrator log-in into system use or email pin . student has incharge maintains student record .

Extraction Details

List of Classes	List of Attributes	List of Operations
student	username	register
library	password	borrow
administrator	book	log-in
incharge	journal	
	email	
	pin	
	record	

Fig 5.3: Text input page

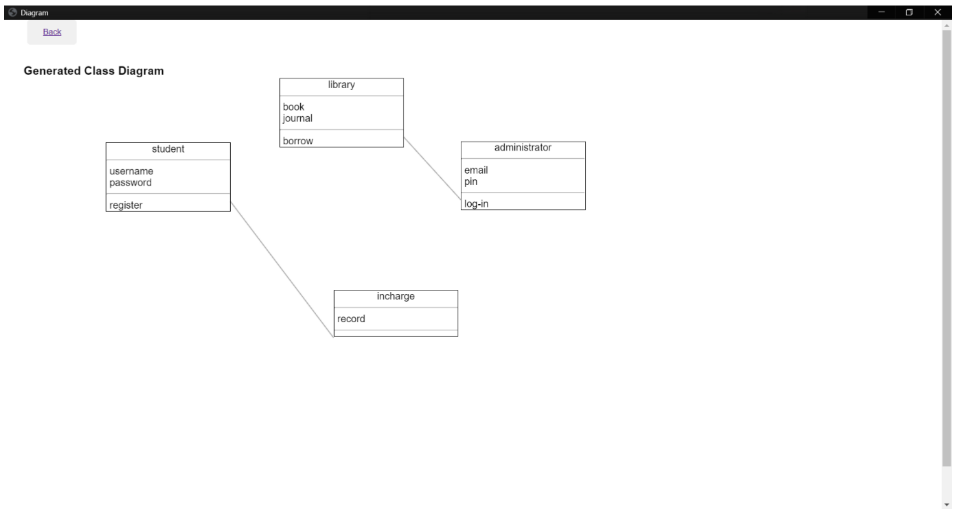


Fig 5.4: Generated diagram page

CHAPTER 6
RESEARCH

The aim of this research is to evaluate the effectiveness of artificial intelligence in the design phase of software development and thus draw a conclusive answer on whether artificial intelligence is preferred over the traditional method and to deduce a viable explanation over the findings of the research. The basis of comparison are on the following fields:

1. **Level of automation:** It is defined as the degree to which a task is automated. There exists a conflict between software engineers and artificial intelligence[2] in the field of software engineering due to the level of freedom that automation deprives a software engineer, and due to the various risks involved in automating the software development process. The research will attempt to find an optimum level of automation in software engineering.
2. **Time:** The research will attempt to find the time saved by incorporating artificial intelligence in the software development process.
3. **Cost of development:** The research will attempt to find the cost which is saved by incorporating artificial intelligence in software engineering.
4. **Accuracy:** The research will attempt to calculate the accuracy of the generated diagrams, which have been generated using artificial intelligence.

As has been stated, the aim of this research is to use class diagrams as an example of an essential design diagram involved in the design phase of software development, and in extension, a class diagram generator which uses natural language processing has been developed to compare against the traditional pen and paper method of making class diagrams.

6.1 RESEARCH METHODOLOGY

6.1.1 DETERMINING THE ACCURACY OF THE CLASS DIAGRAM GENERATOR

The accuracy of the class diagram generator is measured by subjecting it to a number of test problem statements and calculating the number of generated classes, methods and attributes. The metrics used for this purpose are given below: (Add reference to paper)

1. **True Positive (TP):** Number of classes, methods or attributes are correct and are generated correctly.
2. **False Positive (FP):** Extra classes, methods or attributes are generated and are incorrect.
3. **False Negative (FN):** No classes, methods or attributes generated.

From the accuracy metrics, two factors are then calculated:

1. **Precision:** It is a measure of how close the resultant diagram is to the correct diagram.
2. **Sensitivity:** It is a measure of how sensitive the tool is in detecting classes, attributes and methods.

The test problems used for testing the accuracy are as follows:

Statement Number	Problem Statement
S1	A system shall allow customers to create and login to their accounts using their username and password.
S2	A mall has a collection of perishables and non-perishables. Perishables have fruits, vegetables and meat and can expire. Non-perishables have plates, spoons and forks and can degrade.
S3	A library consists of books, novels, magazines, journals and has a librarian and an assistant. The librarian has a unique username and password to login to the system and maintain and add new records. The assistant also has a unique username and password to access the system and update the records. Members have a member-card to borrow and return the items. The member-card contains information like member-id, name, age, email and address.
S4	A system shall allow the customers to register and login by entering their username and password, in order to get access to the store and buy different products. The store consists of phones, laptops, tablets and desktops to sell and has an administrator. The administrator has a username and password to login and maintain the system. Payment methods include cash, card, cheque, UPI etc.
S5	An online grocery store should allow user to login to the system with their username and password to browse through a list of groceries and buy them. Payment can be done using card, net-banking, UPI.

Table 6.1: Table of problem statements

6.1.2 COMPARISON WITH TRADITIONAL METHOD

The research has been performed on a population of 10 final year students from the Department of Computer Science & Engineering, Assam Don Bosco University. The class diagrams and the problem statement were presented in English. Each subject performed the experiment in a controlled and supervised environment and each subject was instructed to not use any external aid to generate the class diagram from the given problem statement.

The subjects were given the following problem statement:

"A library consists of books, novels, magazines, journals and has a librarian and an assistant. The librarian has a unique username and password to login to the system and maintain and add new records. The assistant also has a unique username and password to access the system and update the records. Members have a member card to borrow and return the items. The member card contains information like member-id, name, age, email and address." (S3 from Table 6.1)

Once the problem statement has been received by the subjects, a timer has been set to calculate the following times:

1. **Analysis Time (AT):** This is the time taken by the subjects to read and understand the statement completely before proceeding to draw the diagram.
2. **Diagram Generation Time (DGT):** This is the time spent by the subjects to draw the diagram.
3. **Overall Time (OT):** This is the overall time spent by the subjects to fully complete making the class diagram.

The above statement was also used as an input statement to the class diagram generator, and the following times were recorded:

1. **Analysis Time (AT):** This is the time taken for the software to analyse the input statement and generate the candidate classes, methods and attributes. This also includes the time spent by the user to make any necessary modifications before the final diagram is generated.
2. **Diagram Generation Time (DGT):** This is the time taken by the class diagram generator to generate the diagram, after having generate the candidate classes, methods and attributes and after having the user make any necessary changes, i.e., the time taken to generate the diagram after analysis. This also includes the time spent by the user to rearrange the diagram in a suitable manner.

3. **Overall Time:** This is the total time taken by the tool to fully generate the class diagram.

In addition to the time taken, the accuracy of the generated diagrams is also calculated and compared accordingly using the metrics below:

1. **True Positive (TP):** Number of classes, methods or attributes are correct and are generated correctly.
2. **False Positive (FP):** Extra classes, methods or attributes are generated and are incorrect.
3. **False Negative (FN):** No classes, methods or attributes generated.

From the accuracy metrics, two factors are then calculated:

1. **Precision:** It is a measure of how close the resultant diagram is to the correct diagram.
2. **Sensitivity:** It is a measure of how sensitive the test subjects ir in detecting classes, attributes and methods.

6.2 DATA OBTAINED

For determining the accuracy of the class diagram generator using the methods stated above, the following data has been obtained:

Test Statement	Expected Output			Output Generated Correctly (True Positives, TP)			Output Generated Incorrectly (False Positives, FP)			Output not Generated (False Negative, FN)		
	No. of Classes	No. of Attributes	No. of Methods	No. of Classes	No. of Attributes	No. of Methods	No. of Classes	No. of Attributes	No. of Methods	No. of Classes	No. of Attributes	No. of Methods
1	1	2	2	1	2	2	0	0	0	0	0	0
2	3	8	2	3	8	2	0	0	0	0	0	0
3	5	15	5	5	15	5	0	3	0	0	0	0
4	4	12	6	4	11	6	0	1	0	0	1	0
5	2	5	3	1	4	3	1	2	0	1	1	0

Table 6.2: Accuracy values of class diagram generator

Where:

- **Expected Output:** Denotes the expected no. of classes, attributes, methods.
- **Output Generated Correctly (True Positives, TP):** Denotes the no. of classes, attributes and methods that match with the expected output.
- **Output Generated Incorrectly (False Positives, FP):** Denotes the expected no. of classes, attributes and methods that does match with the expected output or has not been generated.

For calculating the precision, the following formula is used:

$$\% \; of precision = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \; x \; 100$$

For calculating the average precision, the following formula is used:

$$average \; \% \; of precision = \frac{\sum \% \; of precision}{\text{no . of test statements}}$$

Test Statement	Precision (%)		
	Classes	Attributes	Methods
1	100	100	100
2	100	100	100
3	100	83.3	100
4	100	91.6	100
5	50	66.6	100
Average Precision (%)	90	88.3	100

Table 6.3: Precision percentage of class diagram generator

For calculating the sensitivity, the following formula is used:

$$\% \; of sensitivity = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \; x \; 100$$

For calculating the average sensitivity, the following formula is used:

$$average \; \% \; of sensitivity = \frac{\sum \% \; of sensitivity}{\text{no . of test statements}}$$

Test Statement	Sensitivity (%)		
	Classes	Attributes	Methods
1	100	100	100
2	100	100	100
3	100	100	100
4	100	91.6	100
5	50	80	100
Average Sensitivity (%)	90	94.32	100

Table 6.4: Sensitivity percentage of class diagram generator

For comparison with the traditional method of drawing class diagrams from a problem statement by using the method stated above, the following data was obtained:

Test Subject	Analysis Time (s)	Diagram Generation Time (s)	Overall Time (s)
T1	114	43	157
T2	246	603	849
T3	85	182	267
T4	76	68	144
T5	40	207	247
T6	110	576	686
T7	120	433	553
T8	118	432	550
T9	75	89	164
T10	140	152	292

Table 6.5: Time values for test subjects

From Table 6.5, the following calculations can be made:

Average Analysis Time: $AT_{avg} = \frac{\sum_{i=1}^{10} ATi}{10}$

$AT_{avg} = 112.4s$

Average Diagram Generation Time: $DGT_{avg} = \frac{\sum_{i=1}^{10} DGTi}{10}$

$DGT_{avg} = 278.5s$

Average Overall Time: $OT_{avg} = \frac{\sum_{i=1}^{10} OTi}{10}$

$OT_{avg} = 390.9s$

The problem statement was also passed as input to the class diagram generator and the following timings were recorded:

Analysis Time (AT): 12.58s

Diagram Generation Time (DGT): 10.50s

Overall time (OT): 23.08s

Consequently, the diagrams generated by the traditional method were analysed and compared with the following data:

- Expected No. of Classes: 5
- Expected No. of Attributes: 15
- Expected No. of Methods: 5

And thus the data given below was obtained:

Test Subject	Output Generated Correctly (True Positive, TP)			Output Generated Incorrectly (False Positive, FP)			No Output Generated (False Negative, FN)		
	No. of Classes	No. of Attributes	No. of Methods	No. of Classes	No. of Attributes	No. of Methods	No. of Classes	No. of Attributes	No. of Methods
T1	1	3	0	2	0	0	4	12	5
T2	4	9	2	4	10	12	1	6	3
T3	4	8	0	1	6	0	1	1	5
T4	3	2	0	1	4	0	2	13	5
T5	4	8	4	0	4	2	1	7	1
T6	5	11	4	1	4	2	0	4	1
T7	4	4	5	4	14	9	1	11	0
T8	3	5	4	5	7	3	2	10	1
T9	2	6	2	4	6	0	3	9	3
T10	4	10	5	2	6	3	1	5	0

Table 6.6: Accuracy values of test subjects

Therefore, with reference to Table 6.6, the precision and sensitivity of the test subjects could be calculated using the given formulas and were found to be as follows:

Test Subject	Precision (%)			Sensitivity (%)		
	Classes	Attributes	Methods	Classes	Attributes	Methods
T1	33.33	100	0	20	20	0
T2	50	47.36	14.28	80	60	40
T3	80	57.14	0	80	53.33	0
T4	75	33.33	0	60	13.33	0
T5	100	66.66	50	80	53.33	80
T6	83.33	73.33	66.66	100	73.33	80

Test Subject	Precision (%)			Sensitivity (%)		
	Classes	Attributes	Methods	Classes	Attributes	Methods
T7	50	22.22	35.71	80	26.66	100
T8	37.5	41.66	57.14	60	33.33	80
T9	33.33	50	100	40	40	40
T10	66.66	62.52	62.5	80	66.66	33.33

Table 6.7: Precision percentage and Sensitivity percentage of test subjects

With reference to Table 6.7, the average precision and sensitivity of the test subjects were found to be as:

	Classes	Attributes	Methods
Average Precision (%)	60.92	55.42	38.63
Average Sensitivity (%)	68	44	45.33

Table 6.8: Average Precision and Average Sensitivity of test subjects

6.3 INFERENCES

From the data collected, we can conclude the following points:

1. The class diagram generator showed the following average precision values:
   - For Classes: 90%
   - For Attributes: 88.3%
   - For Methods: 100%
2. The class diagram generator showed the following average sensitivity values:
   - For Classes: 90%
   - For Attributes: 94.32%
   - For Methods: 100%
3. For generating a class diagram from a problem statement (S3 from Table 6.1), the test subjects showed an average overall time of 390.9s while the class diagram generator took 23.08s. This amounts 367.82s of time saved which equates to the class diagram generator being 94.10% faster than the traditional method.
4. The test subjects the following average precision values:

- For Classes: 60.92%
- For Attributes: 55.42%
- For Methods: 38.63%

With reference to serial no. 3 from Table 6.3, for the same statement, the class diagram showed the following precision values:

- For Classes:100%
- For Attributes: 83.3%
- For Methods: 100%

5. The test subjects the following average sensitivity values:

- For Classes: 68%
- For Attributes: 55.42%
- For Methods: 45.33%

With reference to serial no. 3 from Table 6.4, for the same statement, the class diagram showed the following precision values:

- For Classes:100%
- For Attributes: 100%
- For Methods: 100%

CHAPTER 7
PUBLICATIONS

International Journal of Computer Sciences and Engineering (IJCSE): A survey paper conducted on the artificial intelligence in software engineering was made and submitted to the journal.

CHAPTER 8
CONCLUSION

Therefore, with the Class Diagram Generator, the class diagrams were generated from the input natural language text and have shown to be much faster and accurate than the traditional method of class diagram generation, which can extensively conclude that artificial intelligence is can make the design phase of software development a lot more efficient for software developers.

REFERENCES

Research Papers:

[1] Rech, J., & Althoff, K. D. (2004). Artificial intelligence and software engineering: Status and future trends. *KI*, *18*(3), 5-11.

[2] Sommerville, I. (1993). Artificial intelligence and systems engineering. Prospects for Artificial Intelligence: Proceedings of AISB'93, 29 March-2 April 1993, Birmingham, UK, 17, 48.*Computing Department, Lancaster University, LANCASTER LA1 4YR, UK.*

[3] Ammar, H. H., Abdelmoez, W., & Hamdi, M. S. (2012, February). Software engineering using artificial intelligence techniques: Current state and open problems. In *Proceedings of the First Taibah University International Conference on Computing and Information Technology (ICCIT 2012), Al-Madinah Al-Munawwarah, Saudi Arabia* (p. 52).

[4] Feldt, R., de Oliveira Neto, F. G., & Torkar, R. (2018, May). Ways of applying artificial intelligence in software engineering. In *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering* (pp. 35-41). ACM.

[5] Raza, F. N. (2009, March). Artificial intelligence techniques in software engineering (AITSE). In *International MultiConference of Engineers and Computer Scientists (IMECS 2009)* (Vol. 1).

[6] Herchi, H., & Abdessalem, W. B. (2012). From user requirements to UML class diagram. *arXiv preprint arXiv:1211.0713*.

[7] More, P., & Phalnikar, R. (2012). Generating UML diagrams from natural language specifications. *International Journal of Applied Information Systems, Foundation of Computer Science*, *1*(8), 19-23.

[8] Joshi, S. D., & Deshpande, D. (2012). Textual requirement analysis for UML diagram extraction by using NLP. *International journal of computer applications*, *50*(8), 42-46.

[9] Ibrahim, M., & Ahmad, R. (2010, May). Class diagram extraction from textual requirements using natural language processing (NLP) techniques. In *2010 Second International Conference on Computer Research and Development* (pp. 200-204). IEEE.

[10] Meudec, C. (2001). ATGen: automatic test data generation using constraint logic programming and symbolic execution. *Software Testing, Verification and Reliability*, *11*(2), 81-96.

[11] Harmain, H. M., & Gaizauskas, R. (2000, September). CM-Builder: an automated NL-based CASE tool. In *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering* (pp. 45-53). IEEE.

[12] Balzer, R., Fikes, R., Fox, M., McDermott, J., & Soloway, E. (1990, August). AI and software engineering: will the twain ever meet?. In *AAAI* (pp. 1123-1125).

[13] Zhong, Z., Guo, J., Yang, W., Xie, T., Lou, J. G., Liu, T., & Zhang, D. (2018, June). Generating Regular Expressions from Natural Language Specifications: Are We There Yet?. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

[14] Sharma, N. A. K. U. L., & Yalla, D. P. Issues in Developing UML Diagrams from Natural Language Text. In *Proc. WSEAS Conference, Recent Advances In Telecommunications, Informatics And Educational Technologies* (pp. 139-145).

[15] Ford, L. (1987). Artificial intelligence and software engineering: a tutorial introduction to their relationship. *Artificial intelligence review*, *1*(4), 255-273.

Web Links:

[16] COCOMO Model:

https://www.geeksforgeeks.org/software-engineering-cocomo-model (02/08/2019)

[17] Anaconda Navigator:

https://docs.anaconda.com/anaconda/navigator (18/10/2019)

[18] Project Idea:

https://www.smartdatacollective.com/traditional-vs-machine-learning-for-software-development-paradigms/ (25/07/2019)

https://www.analyticsindiamag.com/how-machine-learning-is-changing-the-software-development-paradigm/ (25/07/2019)