AUTOMATIC GENERATION OF CLASS DIAGRAM FROM NATURAL LANGUAGE REQUIREMENTS USING NATURAL LANGUAGE PROCESSING

Author

Mishal Muneer

00000319899

MS-19 (CSE)

Supervisor

Dr. Usman Qamar

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

FEBRUARY, 2022

AUTOMATIC GENERATION OF CLASS DIAGRAM FROM NATURAL LANGUAGE REQUIREMENTS USING NATURAL LANGUAGE PROCESSING

Author

Mishal Muneer

00000319899

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

Thesis Supervisor:

Dr. Usman Qamar

Thesis Supervisor's Signature: _____

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

FEBRUARY, 2022

In the name of Allah most beneficent most merciful

Declaration

I certify that this research work, titled *"Automatic Generation of Class Diagram from Natural Language Requirements using Natural Language Processing"* is my own work. This work is not presented elsewhere for assessment. The material used from other sources are properly acknowledged/ referred.

Signature of Student

Mishal Muneer

2022-NUST-MS-Soft-19

Language Correctness Certificate

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

Signature of Student

Mishal Muneer

Registration Number

00000319899

Signature of Supervisor

Dr. Usman Qamar

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 only in 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 EME, 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 thankful to Allah Almighty for giving me countless blessings and guidance in this research work. I want to thank my supervisor Dr. Usman Qamar for encouraging and supporting me throughout my research. He is a source of great knowledge and his expertise helped a lot in achieving my goals. I also want to thank Dr. Summair Raza and my GEC members Dr. Wasi Haider Butt and Mr. Jhan Zeb for their unlimited guidance and support. I want to thanks my parents, siblings, and friends for encouraging me throughout my degree program and motivating me to overcome obstacles and difficulties. I would like to express my gratitude to the Department of Computer and Software Engineering and the management of College of Electrical and Mechanical Engineering for assisting me throughout my journey.

Dedicated
to my exceptional parents,

Who always picked me up on time and
encouraged me to go on every
adventure, especially
this one.

ABSTRACT

Software Development Life Cycle (SDLC) is a systematic approach that consists of software requirement engineering, software design, development, implementation, and deployment. Software design is an important phase that helps realize the requirements into working code. Recently, Unified Modelling Language (UML) has become an important tool to develop software design. It provides various modeling structures to depict both static and dynamic behaviors of system. For static structure, class diagram is an important model that shows different classes and their relationship. It is significant to develop tools to automatically generate class diagram from natural language requirements.

Various techniques have been proposed in literature to automatically generate class diagram from natural language requirements, but these techniques fail to deal with redundant information present in the form of synonyms. Requirements written in compound and complex sentences are also problem for these techniques. Furthermore, the generated class diagram may not be optimized in terms of coupling relationship between classes. These factors make automation of class diagram generation from natural language requirements a highly challenging task.

Natural Language Processing (NLP) is a well-known approach of computational linguistics used to extract structured information by processing unstructured text automatically. The technique has been applied to a number of fields in this regard like sentiment analysis, newspapers analysis, and bio medical and so on. With advancement in computing, improvement in software development methodology has also gained vital importance from researchers in order to speedup software development to fulfill market needs.

In this research study, we comprehensively investigate the application of NLP for the generation of class diagram. In this research a Systematic Literature Review (SLR) is carried out to select 29 articles published during 2014-2021. After quality Evaluation, only 17 articles consider that fully fulfills the objective of our research. Subsequently, 14 combinations of main NLP activities (i.e., Tokenization, POS tagging, Chunking, and Parsing) and 12 NLP algorithms are identified. Furthermore, 23 existing tools are identified that are further divided into two categories tools utilized by the researchers are 11 and purposed by researchers are 12. Finally, a comprehensive analysis is performed to investigate the automation level of NLP applications for the generation of the class diagrams and test cases from early plain text requirements.

Moreover, this research proposes a model to automatically generate a more accurate and optimized class diagram from natural language requirements using natural language processing. A tool named SD-LINGO is developed in this research. The effectiveness of the proposed model will be analyzed by comparing the generated design with other state of the art approaches. The validation is performed through six benchmark case studies and six different set of requirements. The experimental results proved that the proposed NLP approach is fully automated and considerably improved as compared to the other state-of-the-art app.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Without software, we would be unable to function in today's world. Writing efficient codes for the development of high quality and successful programs is one of the procedures covered by software engineering [1].

There are still numerous reports of unsuccessful software projects and software failures. Software engineering is regarded as insufficient for the production of modern software. However, most of these so-called software failures, in my opinion, are the result of two causes:

- ***Increasing demands***: The demands vary as latest software engineering approaches enable us to design larger and more sophisticated systems. Systems must be designed and delivered very speedily; bigger, even more, complicated systems are needed; and systems must possess previously unimaginable capabilities. Current software engineering approaches are unable to fulfill these new expectations so new software engineering approaches must be invented [1].
- ***Low expectations***: Without using software engineering principles and techniques, writing computer software is very simple. Many organizations have veered into software development as their products and services have grown. In their daily activities, they do not apply software engineering techniques. As a result, their software is usually costly and less trustworthy than it should be. To resolve this concern, we require enhanced software engineering training and education [1].

Object Oriented Technology (OOT) is a very famous approach to building software systems. A little while back OOT is extended from the implementation stage of the SDLC to cover the SDLC initial stage like analysis and design. A lot of object-oriented methods have been suggested for the initial stages of SDLC e.g., OMT, Booch, and OOSE. In the development process, definition of the concept vary from one OO method to another. While CASE tools provide help in generating the software design i.e., use case diagram generation and class diagram generation in many cases. As programming became more complicated, additional framework was required for the development effort to serve as a foundation for project management, as well as to assist planning and communication because now teams were required to build software instead of individuals [2]. This eventually evolved to the SDLCM described below. The Software Development Life Cycle (SDLC) model is used to create high-quality software.

1.1 Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is an important aspect of software development. It explains the steps associated in transforming codes into final products, which are commonly

referred to as software. It includes a number of stages, starting with requirements analysis and ending with maintenance [3].

SDLCMs were created with the primary goal of providing a structure for software development, as well as a framework for software development methods and tasks [4]. SDLCM facilitates in the splitting of this highly complicated task into smaller subtasks, which helps in the planning and monitoring of work, as well as the support of cooperation and communication among the various individuals and groups involved, and the quality of the end product [2].

Today, two of the most essential responsibilities for software development companies are software development and delivery. The SDLC, sometimes known as the application life cycle, describes how the development process is organized within a project or a company. The SDLC refers to the process of planning, developing, testing, controlling, and distributing software. When SDLC is implemented, it is trivial to identify which stage the team is on in any software project, which assets are required, and what stage will be the next [3].

1.1.1 Phases of SDLC Model

Every software development life cycle consists of several phases. The major six phases involved in software development life cycle model (See Figure 1).

1.1.1.1 Requirement Analysis

This is the first step of the SDLC, during which all essential information is gathered and reviewed [3]. Requirement management is both the most essential and the most ignored aspect of Software Engineering and Project Management. Poor requirement collection and tracking follow-ups cause 80% of projects to fail [5]. This phase determines the development team's performance, customer satisfaction, project success, and several other factors. The fundamental goal of this stage is to identify the requirement and construct an SRS (software requirement specification). We specify user and system requirements throughout this step. We describe an issue and create a document in this step, which is called the problem domain phase [4]. The more time and effort you put into obtaining requirements, analyzing them, and documenting them properly, the smoother and more result-oriented the subsequent phases will be.

1.1.1.2 Design

This is the second step of the SDLC, and it focuses on the overall structure of the upcoming project [3]. It is also the very crucial and essential part of the SDLC. The process of converting requirements into a thorough design representation of a software system is known as software design. It is believed that the key to reliable and intelligible software is good software design. [6]. We produce a Software Design Document (SDD) in this stage, and we gather requirements in the requirement analysis stage. As a result, we design an SDD, this step takes an input from the previous step and outputs to the next step [4].

1.1.1.3 Implementation

The implementation phase of the SDLC, also known as deployment, is the third step of the SDLC. It is the coding phase of the SDLC. All aspects are incorporated into the developed software, which generates source code [3]. In this step, we implement a design phase and create a system that produces output, but we don't know whether the result is correct or incorrect [4].

1.1.1.4 Testing

The testing process is highly crucial since it allows us to determine whether our system is working properly or not. Whether our system satisfies the customer's requirements or not. Unit testing and system testing are the two main types of testing. We test smallest element of the system in unit testing, but we test the entire system in system testing, which does not require the underlying design logic [4]. The verification and rectification of any code bugs are part of the testing process. Everything is rigorously tested and retested as needed until all issues are rectified [3].

1.1.1.5 Deployment

After analyzing the application and completing all required iterations, the code is ready for implementation. After that, end users will be able to access the project [3].

1.1.1.6 Maintenance

The software life cycle is followed by maintenance [3]. Error detection, rectification, and enhancement of product features are all part of software maintenance. Every piece of software must be maintainable since customers will want to update and add new features over time [4]. If there are issues with software, they may be rectified or corrected in the following version, depending on how serious the issues are [3].

Figure 1 Software Development Life Cycle

1.1.2 Motivation for Selecting Software Design in SDLC

Software Design is very essential and crucial part of the SDLC. It plays the central role in the software development [7]. Software design helps realize the requirements into working code. The requirements are used as input during the software design phase. However, when designing, it is possible to find ways to offer greater functionality without incurring a significant additional expense, or perhaps without incurring any additional economic burden. We may need to change some of the requirements to accommodate new technology. We must keep in mind that end users are not exposed to all of the computer's features or capabilities. They wouldn't be able to specify things in such a way that the computers and software development platforms' capabilities are fully utilized. As a result, we should search for ways to improve functionality that will benefit end users or management during the design phase. Modifications requests from consumers or any other stakeholder are another part of requirement management that we must handle throughout the design process. We need to accommodate all of the modification requests received so far into the design phase of the software so that the developers team is not delayed by design changes throughout development [8].

1.2 Level of Research Carried Out on This Problem

1.2.1 Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a modeling and design language for software systems. Its origins can be traced back to the object-oriented approach, but it is now employed in a wide range of software development projects. The OMG (Object Management Group) is the organization that invented and maintains UML. One thing to keep in mind is that UML is a modeling tool, not a software development approach. It's more of a lingo for describing the system.

When you utilize the UML to create a design, you'll usually create two types of models:

1. *Structural models*, which use object classes and relationships to explain the system's static structure. Composition, uses/used-by and generalization (inheritance) relationships are all significant relationships to document at this level. Structural models can be static (showing the system's design structure) or dynamic (showing the system's organization while it's running) [1].
2. *Dynamic models*, which reflect the relationships between system components and describe the system's dynamic structure. The sequence of service requests performed by objects, as well as the state changes generated by these object interactions, are examples of interactions that could be documented.

As demonstrated in Figure 2, structural diagrams include object diagrams, packages diagrams, deployment diagrams, class diagrams and composite structure diagrams, all of which depict the system's static structure in terms of the components that make it up and their interactions. Our main focus in this research to automate class diagram because for static structure class diagram is

an important model that shows different classes and their relationships. It is significant to develop tools to automatically generate class diagram from natural language requirements.

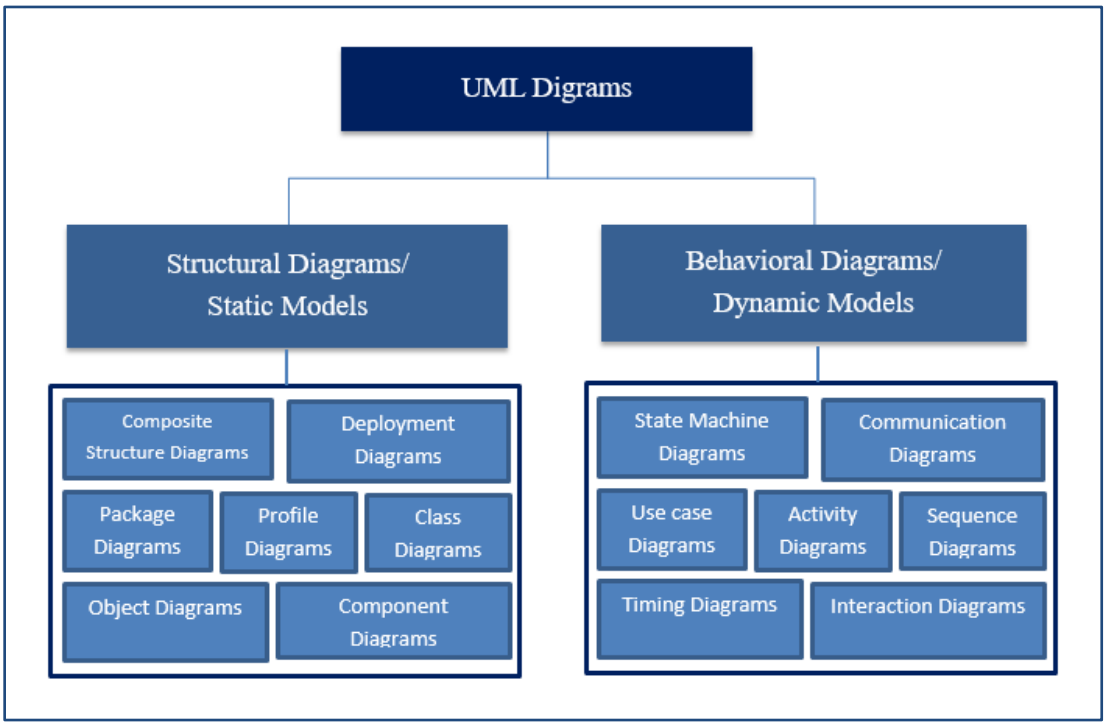

Figure 2 UML Diagrams

1.2.1.1 Component Diagram

The structural relation between components in a software system is depicted using a component diagram. These are typically employed while working with large, complex systems. Interfaces allow components to communicate with one another. Connectors are used to connect the interfaces [9].

1.2.1.2 Deployment Diagram

A deployment diagram depicts your system's hardware as well as the software that runs on it. When your software solution is deployed over numerous machines, each with its own configuration, deployment diagrams come in handy [10].

1.2.1.3 Object Diagram

Object diagrams, also known as Instance diagrams, are similar to class diagrams in appearance. They explain the link between objects in the same way as class diagrams do, but they do so use real-world examples. They depict how a system will appear at a certain point in the future. The objects are utilized to demonstrate complex relationships between objects since they contain data [11].

1.2.1.4 Package Diagram

A package diagram, as the name implies, depicts the interdependencies between distinct packages in a system [12].

1.2.1.5 Profile Diagram

In UML 2, a new diagram type called a profile diagram was added. This is a diagram that is only used in very few specifications [13].

1.2.1.6 Composite Structure Diagram

The internal structure of a class is depicted using composite structure diagrams [14].

1.2.1.7 Use Case Diagram

Use case diagrams are the most well-known type of the behavioral UML types, providing a graphic representation of the actors involved in a system, the various functions required by those actors, and how these functions interact. It's a wonderful place to start when talking about a project since you can quickly identify the important processes and actors of the system [15].

1.2.1.8 Activity Diagram

Activity diagrams are graphical representations of workflows. They can be utilized to explain the business and operational workflows of any system component. Activity diagrams are sometimes used instead of state machine diagrams [16].

1.2.1.9 State Machine Diagram

State machine diagrams are comparable to activity diagrams, although the notations and usage differ slightly. They're sometimes referred to as state diagrams or state chart diagrams. These are highly handy for describing the behavior of objects that behave differently depending on their current state [17].

1.2.1.10 Sequence Diagram

In UML, sequence diagrams depict how objects interact with one another and in what order they interact. It's important to keep in mind that they only reveal interactions for a specific scenario. Interactions are illustrated as arrows and processes are depicted vertically [18].

1.2.1.11 Communication Diagram

They were known as collaboration diagrams in UML 1. Communication diagrams are identical to sequence diagrams, but they focus on the messages that are sent between objects. A sequence diagram and other objects can be used to express the same information [19].

1.2.1.12 Interaction Overview Diagram

Activity diagrams and interaction overview diagrams are extremely similar. Interaction overview diagrams display a sequence of interaction diagrams, whereas activity diagrams depict a

sequence of processes. They're a set of interaction diagrams in the order in which they occur. As previously stated, there are seven different types of interaction diagrams, each of which can be used as a node in an interaction overview diagram [20].

1.2.1.13 Timing Diagram

Sequence diagrams and timing diagrams are extremely similar. They depict the activity of objects over a period of time. If there is only one object, the diagram is simple. However, if several objects are involved, a timing diagram is utilized to demonstrate how the objects interact over that time period.

1.2.1.14 Class Diagram

The important element of every object-oriented solution is the class diagram. The classes in a model are represented using class diagrams. To model high-level design, class diagrams are utilized (roughly equivalent to SRS). Customer requirements are examined, classes are abstracted, and class diagrams are used to model them. A class in most modeling software is made up of three components. The first component has the class name, the middle has the class attributes, and the bottom has the class operations or methods. Classes are put together to build class diagrams in a complex system with numerous related classes. Various kinds of arrows represent various types of relation between classes [8].

1.2.1.14.1 Cohesion and Coupling

Stevens et al. was the first to use the term "coupling" in software engineering [1], during the times when structured programming was the standard. It was defined as a metric for determining the strength of an association established by a link between two modules. Coupling is defined in the context of object-oriented design as how one class is linked to another. The term coupling refers to a class's dependency on another class [21]. Excessive dependence may reduce the class's reusability and enhance maintenance work. The presence of an increasing number of dependencies indicates that changes to other classes are more likely to break the functionality of the class. It's possible that modifications in one class will have an impact on others. Furthermore, strong coupling will necessitate additional testing effort and time [22]. Testing efforts, comprehension activities, maintenance tasks and reuse are made easier with minimum coupling and strong cohesion.

1.2.2 Natural Language Processing (NLP)

Data scientists are primarily concerned with three things: gathering data, analyzing data and inferring information from the data. All of these tasks need specialized personnel, requires time and money. The next and most difficult step is to convert data into products. As a result, several academic and industrial research groups are interested in this topic. Data-driven techniques have grown in popularity in recent decades, owing to the fact that they require far less

human labor. Surprisingly, Natural Language Processing (NLP) is one of the fields that is affected by data.

Natural language processing (NLP) is a technique for processing natural language material and providing the researchers with the appropriate results. Natural language processing, in simple terms, is the act of converting natural language text into desired outcomes such as requirements templates, design creation or test cases. As said initially that desired useful information collected from the user requirements is such a difficult and tedious task to be done. Here we can use the natural language processing. There are many approaches to handle this specific issue, but we adopt the most famous approach that is object-oriented approach. Natural language processing is the branch of artificial intelligence which is basically focuses on automatic analysis of plain natural language user requirements. There are several techniques in the natural language processing e.g., tokenization, POS tagging, chunking, and parsing. These techniques are used by different researchers in different scenarios with different combinations but here we used POS tagging as NLP approach.

1.2.2.1 Parts-of-Speech/ (POS) Tagging

POS tagging is a helpful technique for tagging words of a sentence. The POS method gives a lot of information related to the words. Semantics, translation and syntax are only a few of the NLP tasks that are taken into account when processing a language. Corpus linguistics and POS tagging research are intrinsically tied. There are two types of tagging techniques: supervised and unsupervised. In the supervised technique, a previously trained corpus is used, and the output is generated based on it. It's simple to tag a sentence in the supervised technique. The first step in the unsupervised technique is training of the whole corpus, after which the input sentence is tagged. In comparison to the supervised tagging technique, this technique is more complicated. Verbs, nouns, adjectives, adverbs, determiners and other grammatical groups are included in the POS [23]. Based on its perspective and definition, POS tagging is as well-known as grammatical text tagging. Figure 3 depicts the POS tagging process. The input sentence is first read, and then it is tokenized into words in the following phase. To assign tags to these tokens, the POS tagging technique is employed. After that, the tagged output is generated.

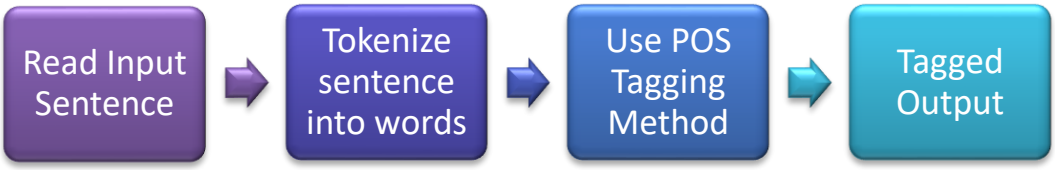

Figure 3 POS Tagging Process

1.3 Thesis Objectives

Natural Language Processing (NLP) is the study of the automatic processing of written natural languages like English. Natural languages can be processed at a number of different levels. From

word analysis to sentence processing to whole discourse analysis, these levels are available. Recent advancements in this field point to interesting ways that could aid software developers in the early stages of software development analysis. The main goal is to device a methodology that uses natural language processing to analyze and process software user requirements written in plain natural language, as well as to investigate how natural language processing can be used to generate a class diagram, which is an important component of software design.

As we have done state of the art systematic literature review, we found the field of NLP so crisp and saw some recent advances in this field. So, this research thesis is concerned with investigating of how natural language processing techniques can be utilized to enable object-oriented user requirements analysis, which is basically a plain text.

To address all these tasks, objectives are shown as follow:

- Investigate previous AI-based work that has been used to assist with the SDLC's early stages.
- Analyzing previous object-oriented Analysis methodologies and identifying common concepts among them.
- Analyzing and identifying NLP tools and techniques.
- Define a practical approach that supports the object-oriented analysis process by utilizing natural language processing tools.
- Execution of the above-mentioned technique.
- Implementation of proposed methodology on various case studies.
- Evaluation of the proposed approach against the human performance and a fair comparison with other past work in the research area.

1.4 Thesis Contribution

This research study will contribute a lot in the Natural Language Processing field. Our contributions are marked as follow:

From state-of-the-art systematic literature review by us, we identified various NLP techniques combinations Table 6 relevant to the respected research studies discussed in the Chapter 2. We also identified different NLP tools utilized by researchers to generate the respected UML diagrams as seen in Table 7. We also identified the NLP algorithms and the purposed tools by researchers as seen in Table 8. We also check the automation level of each tool that is identify through the literature.

From all these identified things, our research study will beneficial for the future researchers and practitioners of this domain to overview the identified material and go through this study to investigate the complexity nature of tools, techniques and algorithms.

On the other hand, a methodology is proposed to generate an automated software design i.e., class diagram. For validation purpose of our proposed methodology, we developed a CASE tool. We evaluated our proposed methodology over several case studies.

1.5 Relevance to National Needs

Software industry has grown rapidly in Pakistan in past decade. Manual process for creating a class diagram is very time consuming and a lot of effort is required which may increase the cost and time of developing the software. This tool overcome these types of problems and provide ease to software industry.

1.6 Advantages of this Research

- Speed up the time of analyst for creating a class diagram.
- Class names, operations and relationships, such as associations and more advanced relationship kinds such as dependency, aggregation and generalization are all derived from UML class elements.
- Generated classes are highly cohesive and loose coupling.
- Our proposed solution will be an automated system that has ability to directly interact with the users as user is the main stakeholder. This is the main advantage of our research which will single out every other research on this field to the best of my knowledge.

CHAPTER 2
LITERATURE REVIEW

Natural Language Processing (NLP) is a well-known artificial intelligence and computational linguistics approach that is commonly used to automatically extract components of concerns from preliminary information provided in simple natural language by humans. This leads to attaining certain business aims like high productivity and low cost/time to-market barriers. Therefore, the applications of NLP are very common in the area of text mining to classify relevant facts from the bulky raw plain text information. The NLP approach comprises different activities [24, 25] (i.e., tokenization, Parts of Speech tagging, chunking and parsing e.tc) as shown in Figure 4.

Generally, the preliminary information of concern is available in raw plain text. It takes a long time and resources for extracting relevant information from the given text document to meet business objectives. To overcome this problem, major NLP steps are performed to automate the extraction of relevant information. Tokenization is the first step to split given plain text file into tokens such as separating words and punctuation and so on.

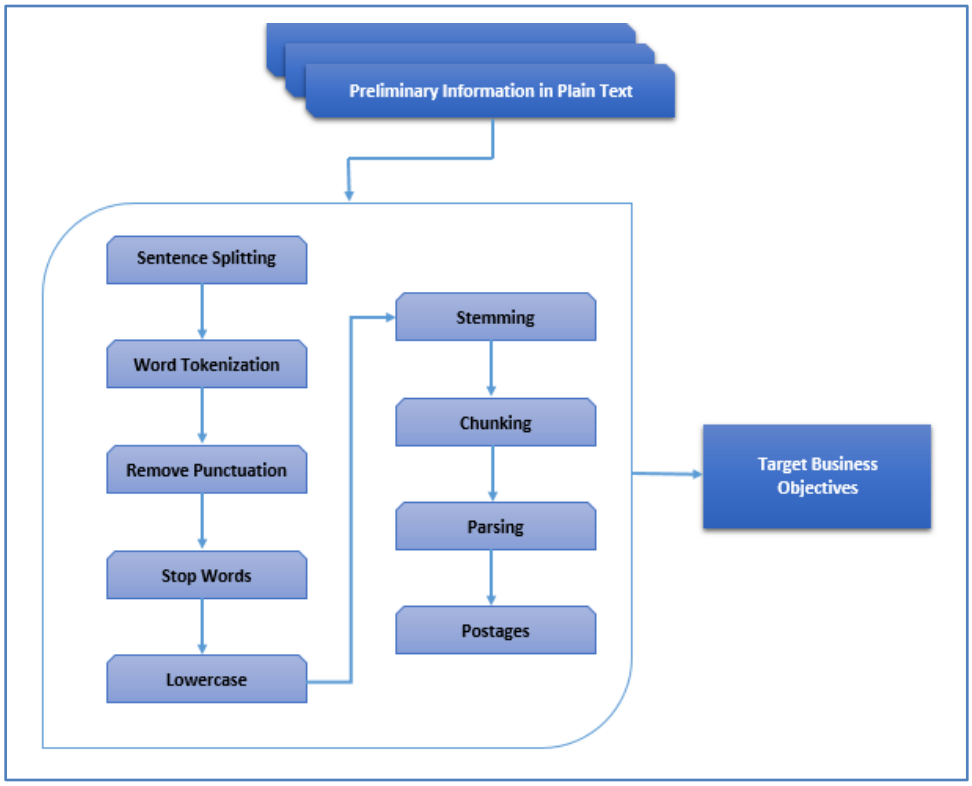

Figure 4 Overview of NLP Techniques

Subsequently, POS tagging is used to assign a part of speech to each word such as a noun or a verb. After that, chunking is performed to detect the boundaries between the phrases. This leads to automatically extracting the relevant features from the given plain text as shown in Figure 4**.**

Software Development Life Cycle (SDLC), usually involves six phases[26] i.e., Requirement, design, implementation, testing, deployment, and maintenance. The requirement and design are two primary phases that provide the input to the other phases. Furthermore, testing is another important phase to perform verification/validation of the developed software applications. There are certain labor-intensive/time-consuming operations involved in this two software development phases (i.e., Requirement analysis, and design) that severely effect productivity, cost, and time-to market goals. Such manual and time-wasting operations of software development phases can be automated by exploiting the features of NLP. However, to the best of our knowledge, no significant research is performed yet to explore, highlight and summarize the utilization of NLP techniques/tools regarding software development phases within a single research work. As a result, in this article, we look into using natural language processing (NLP) to automate the requirement and design phases of software development. The following research questions are the ones we're looking for answers to:

2.1 Research Questions

RQ1: What are the major NLP activities, tools, techniques, and algorithms for the generation of class diagram from natural language requirements?

RQ2: What level of automation is achieved through the NLP for the generation of class diagrams?

A review protocol is developed for this research study on the basis of systematic literature review (SLR) standards [27] . In addition to this, our review protocol defines the relevant research questions, quality assessment, inclusion/exclusion criteria, search strategy and data extraction and synthesis of explored research data. The other details of review, the protocol is as follow:

2.2 Search Strategy

2.2.1 Data Sources

This study was conducted using five famous e-repositories:

1. IEEE Xplore (https://ieeexplore.ieee.org/)
2. Elsevier (https://www.sciencedirect.com/)
3. ACM Digital Library (https://dl.acm.org/)
4. Springer (https://link.springer.com/)
5. Microsoft Academia (https://academic.microsoft.com)

2.2.2 Search Query

The search query used was constructed keeping the following points in mind:

- The main keyword used in the query was ("*Natural Language Processing*" AND "Class Diagram") because the study was focusing, specifically, on the all-natural language processing tool, techniques, and algorithms that are used for extracting the class diagram.
- The study was focused on the automation level of class diagrams through natural language processing so the keyword("automa*") was added to the search.

Hence, the complete query used was:

(("NLP" OR "Natural Language Processing") AND" Class Diagram") AND Automat*

2.3 Selection Strategy

In SLR, criteria are defined to filter down the results acquired against the search query. The study only considered the articles matching the inclusion criteria mentioned below and the remaining articles (or those falling in the exclusion criteria) were dropped from the analysis.

2.3.1 Inclusion Criteria

- IC1: Articles having the main keywords 'Natural Language Processing or NLP and Class Diagram' in their title.
- IC2: Articles published between January 2014 and December 2021.
- IC3: Articles published only in journals, conferences, and conference proceedings.
- IC4: Articles having English as the primary language.
- IC5: Articles with an abstract matching the research objective of the study.

2.3.2 Exclusion Criteria

- EC1: Articles not having the main keyword in their title.
- EC2: Articles published before January 2014 or after December 2021.
- EC3: Articles published elsewhere; besides journals, conferences, and conference proceedings.
- EC4: Articles not having English as a primary language.
- EC5: Articles with an abstract not relevant to the research objective of the study.
- EC6: Exclude all research that has a vague validation method. Which is not research-oriented
- EC7: Reject the researchers which have exactly similar research contents and one of them is already selected.

2.4 Quality Evaluation

The selected articles were evaluated against the following aspects, each on a pre-defined scale of 0 to 1. This was done to check an article's importance in providing answer(s) to any or all of the RQs identified for the study. Table 1 shows the Quality Evaluation Criteria for each article. Any

article having a total score of 1.5 or below was not considered valuable enough and was excluded from the analysis.

- QE1: There are two types of tools. Tools that are used by researchers and the tools that are purposed by the researchers. If article deals with both tools- Yes (+1), partially (+0.5), No (0).
- QE2: Articles deals with NLP techniques-Yes (+1), NO (0).
- QE3: Articles deals with NLP Algorithms-Yes (+1), NO (0).
- QE4: There are different elements of Class diagram. Articles deal with classes- Yes (+1), NO (0); Attributes- Yes (+1), NO (0); Operations/Functions- Yes (+1), NO (0).
- QE5: There are different types of relationship so each relationship type evaluated separately Articles deals with relationships-Association-Yes (+1), NO (0); direct Association-Yes (+1), NO (0); Generalization/Inheritance-Yes (+1), NO (0); Realization-Yes (+1), NO (0); Multiplicity- Yes (+1), NO (0); composition- Yes (+1), NO (0); aggregation- Yes (+1), NO (0).
- QE6: Articles remove the redundancy such as synonyms- Yes (+1), NO (0).

Table 1 Quality Evaluation Criteria

Quality Evaluation Criteria	
NLP Tools	
Used by Researchers	0.5
Purposed By Researchers	0.5
NLP Techniques	1
NLP Algorithm	1
Class Diagram	
Classes	1
Attributes	1
Operations/Functions	1
Relationships	
Association	1
Direct Association	1
Inheritance/Generalization	1
Realization	1
Multiplicity	1
Composition	1
Aggregation	1
Redundancy	1
Total	**14**

2.5 Search Process

Data sources presented in (Section 2.2.1), elaborated those five databases (i.e., ACM, Springer, IEEE, Elsevier, and Microsoft academic) have been selected to conduct this SLR. Selected databases include conference proceedings and high-level impact journals. To achieve search process and get relevant results, we have used different search terms according to our search query and defined criteria e.g., NLP, Natural Language Processing and class diagram etc. Results of our search query in different databases are summarized in Figure 5. Different filters were used against our search terms, according to our inclusion and exclusion criteria. Because without filters selected databases gives a lot of irrelevant results that's why to limit the results different filters were used.

The summary of steps, used during our search process (Figure 5), is given below

- Search query was used in our selected database and get 504 search results as per inclusion and exclusion criteria.
- Eliminate 442 research works on the basis of reading search query main keyword in Abstract as per our inclusion and exclusion criteria.
- Eliminate 28 research works on the basis of time frame that is mentioned in inclusion and exclusion criteria.
- Consider only those articles that is in English according to our inclusion and exclusion criteria.
- After duplication removal we get 29 articles that fit the scope of the study.
- Finally, we include 69 research works which are fully fulfilling our inclusion and exclusion Criteria.
- After quality Evaluation only 17 articles consider that is fully fulfill the objective of our research.

2.6 Distribution of Articles

After quality evaluation distribution of articles

- The selection of five renowned data bases (i.e., ACM, IEEE, Springer and Elsevier, Microsoft academia) is another quality assessment attribute as these databases only published high impact research works. The summary of selected studies, in the context of scientific databases, is given in the Table 2**.**
- We try to include latest studies as much as possible (Table 3) to support the outcomes of SLR through modern developments in the target area.
- We also try to include journal publications as much as possible because these provide detailed study of subject under consideration. However, we found only 2 journals publications, fully agreed with inclusion and exclusion criteria, as given in (Table 4**)**.

Table 2 Distribution of selected researches with respect to scientific databases

Sr. No.	Database	References	Total
1	ACM	[28]	1
2	Elsevier	[29],[30],	2
3	IEEE	[31],[32],[33],[34],[35],[36],[37],[38],[39],[40],[41]	11
4	Springer	[42],[43]	2
5	Microsoft Academia	[44]	1

Table 3 Year wise distribution of selected researches

Sr. No.	Year	References	Total
1	2014	[30],[36]	2
2	2015	[34]	1
3	2016	[28],[37],[42],[40]	4
4	2017	[33],[43]	2
5	2018	[32]	1
6	2019	[44]	1
7	2020	[29],[35],[39],[31]	4
8	2021	[38],[41]	2

Table 4 Conference / Journal wise distribution of our selected researches

Sr. No.	Publication	References	Total
1.	Conferences	[28],[29],[31],[32],[33],[35],[34],[36],[37],[38],[39],[40, 42],[43],[41]	15
2.	Journals	[44],[30]	2

2.7 Data Extraction and Data Synthesis

A total of 505 articles were found in the five repositories against the search query. After applying all five inclusion criteria and removing duplicate items, 29 articles were left that fit the scope and objective of the study. Figure 5 shows the complete selection process of the articles.

For RQ1 and RQ2, the selected articles were to be synthesized based on their coverage of one or more point of NLP i.e., NLP techniques, NLP tools purposed or utilized by researchers, NLP Algorithm to generate class diagram and base on the coverage of identifying different class diagram elements. Table 6, Table 7, Table 8, and Table 9 shows the extraction and synthesis of selected researches to get the answers of the research questions.

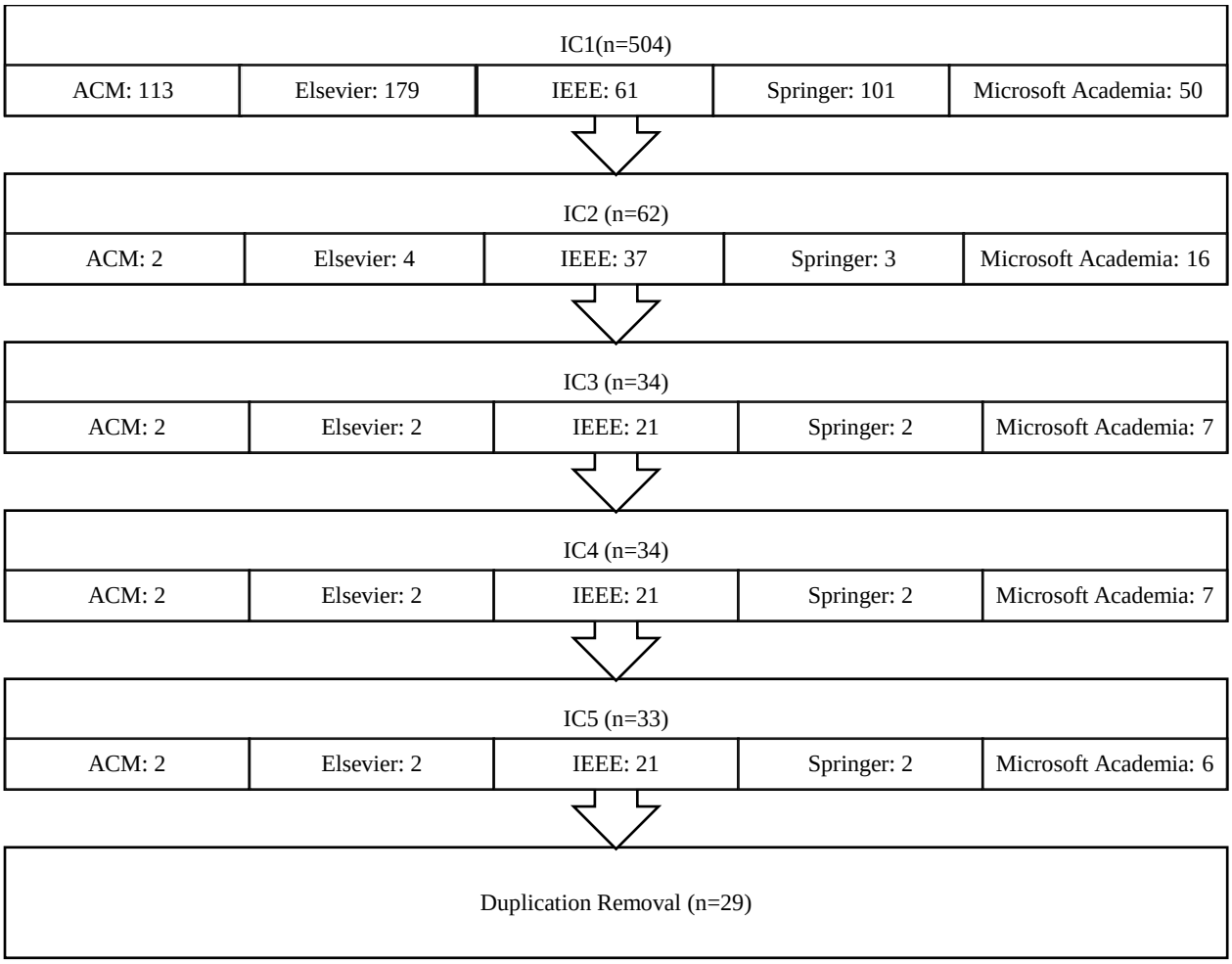

Figure 5 Primary Article Selection Process

2.8 Findings

The 29 articles considered for the study were carefully chosen based on the criteria defined in selection strategy. Each article was then studied in light of the research questions identified for the study. This section outlines the main findings of each article against every RQ. Table 5 shows the total quality score of the 29 selected articles against the quality evaluation (QE) criteria defined earlier.

Table 5 Quality Evaluation of Selected Articles

Sr. No.	Rf.	QE1	QE2	QE3	QE4	QE5	QE6	Total
1	[28]	1	1	1	2	1	0	6
2	[29]	1	1	1	3	3	1	10
3	[30]	1	1	1	3	3	0	9
4	[31]	0.5	1	0	0	0	0	1.5
5	[32]	1	1	0	1	1	0	4
6	[33]	0.5	1	1	1	0	0	3.5
7	[34]	1	1	1	2	2	0	7
8	[35]	1	1	1	3	4	0	10
9	[36]	1	1	1	1	1	0	5
10	[37]	1	1	1	3	0	0	8
11	[38]	1	1	1	3	1	0	7
12	[39]	0.5	1	0	1	0	0	2.5
13	[40]	1	1	1	3	1	0	7
14	[42]	1	1	1	0	0	0	3
15	[43]	1	1	1	1	1	1	6
16	[44]	1	1	1	3	1	0	7
17	[41]	1	1	1	0	0	0	6
18	[45]	0	1	0	0	0	0	1
19	[46]	0.5	0	0	0	0	0	0.5
20	[47]	0.5	0	0	0	0	0	0.5
21	[48]	0	0	0	0	0	0	0
22	[49]	0	1	0	0	0	0	1
23	[50]	0.5	0	0	0	0	0	0.5
24	[51]	0.5	0	0	0	0	0	0.5
25	[52]	0	0	0	0	0	0	0
26	[53]	0.5	0	0	0	0	0	0.5
27	[54]	0	0	0	1	0	0	1
28	[55]	0	0	0	0	0	0	0
29	[56]	0	0	0	0	0	0	0

2.9 Overview of Research Articles

Saimia Nasiri et al [29] purposed an approach that is based on the transition of CIM to PIM. The transition is accomplished by the creation of a platform that creates a class diagram in an XMI file from specifications provided in user stories expressed in Natural Language (English). The object-oriented design elements were extracted using a natural language processing (NLP) tool called "Stanford CoreNLP" by the author. Author uses a purposed algorithm called design element extraction to extract the components of the diagram like (classes, attributes, relations, and operations of classes) from numerous user stories. To process user stories, tokenization, and

Part of Speech, and co reference resolution were utilized. After that, the word dependencies are applied to each sentence in a user story. To prevent extracting classes in a plural and the same singular, the author employed a stemming technique, which is the process of reducing a word to its root. Redundancy in classes was eliminated with the help of WordNet. Finally, using the PyEcore API, the author created an XMI file, that is an Ecore file. In addition, the class diagram is visualized in a PNG image using the PlantUML API. Python was used to implement the entire system.

Vidhu Bhala R. Vidya Sagar et al [30] propose a technique for automatically generating a conceptual model from functional specifications expressed in natural language. The first four functional modules, text pre-processing, syntactic feature extraction, design element extraction, and relation type classification, produce the fundamental conceptual model. NLP tool "Standford core NLP" utilized by author to create a conceptual model diagram. Different NLP techniques was used by author such as tokenization, stemming, POS tagging and parsing for processing the text that is in the form of natural language processing. Author made different rules for creating the class diagram and extract different elements of class diagram such as classes, attributes, operations, association, inheritance, aggregation and composition. Author purposed tool didn't focus on the redundancy and the complex problem statement also fail to identify the cardinality between the classes. The result of the author purposed model are fairly good.

A unique automated technique for recovering scenarios from the source code of web applications is presented by Joanna C. S. Santos et al [31]. These scenarios are depicted with use case diagrams, which are accompanied by sequence diagrams that explain how each use case is implemented within the system. The author applied natural language processing (NLP) approaches like dependency parsing and POS tagging to pull use cases out of these recognized endpoints, and then used the computed program slices to create sequence diagrams for every use case. The author then used Sagan, an open-source Web application, to perform an initial evaluation of the purposed approach by identifying endpoints. Then, using sequence diagrams, they show how the use cases were produced and how they were implemented.

Shweta et al [32] propose a modification on current rules that takes into account the keywords inside the text. The author's main focus was on developing rules for extracting class diagram elements using NLP tools and methodologies. The newly developed rules are based on terms like main scenario, use case name, actor and so on. The identification of class diagram elements was broken down into four stages by the author. The first step is to extract classes, then attributes, methods, and finally relations. The author created the new rules in addition to the previously created rules that are already in use. For the implementation of the defined rules, Stanford Parser v3.5.2 was utilized. NLP techniques such as stop words removal, lemmatization, parsing is utilized by author for the processing of natural language. Purposed tool only identifies the classes, attributes and operations. Author do not focus the relationship. Accuracy of the purposed tool is fairly good.

Mathawan Jaiwai [33] purposed a tool to analyze requirements written in Thai language and extract class diagrams by identifying classes, attributes, relationships, and operations. Author employ NLP tools to automatically process requirement texts and manually apply the rules to extract classes and attributes. And in next steps they apply rules to extract relationships and operations as well as making the process to be as automatic as possible. Heuristic rules may be applied to help increase accuracy as well. Purposed tool can benefit Thai software developers by reducing time in generating class diagrams from requirements written in Thai and also help novice developers learn the relationships between requirements and class diagrams. Firstly, tool preprocessed and reconstructed the requirement that is written in Thai in a plain text format. Secondly, NLP techniques that is tokenization apply on that requirement that tokenize the sentences into words. Third, NLP technique POS tagging is performed on each token. Lastly, class diagrams are extracted using heuristic rules. Also, they used WordNet or a synonym resource to compare nouns with similar meanings.

Richa Sharma [34] presented a method for automatically generating class diagrams from natural language requirements specifications using syntactic dependency analysis and Grammatical Knowledge Patterns (GKPs). The author-purposed technique does not need rewriting the requirements specifications, nor does it impose additional input format constraints. The author presented a system in which the textual representation of requirements is stored in an intermediate form that can accept updates (optional) from the user. As shown by the case studies, the author overcame the limitations of existing approaches. Yet, the correctness of the parser's results severely limits the accuracy of their technique. Despite this, the Stanford parser produces excellent results. They did exploratory research to get human analysts' impression of class diagrams for case-studies addressed in the provided work because there is no gold standard for the actual class-diagram for such scenarios. Author established the benchmark approach to utilize as a baseline for which they produced evaluation metrics for our solution method, based on the received responses.

Esra A. Abdelnabi [41] reviews the literature on converting textual requirements into UML class models and identifies their benefits and weaknesses. The study provides a thorough overview and assessment of current methodologies and tools. The degree of automation, completeness, and efficiency, as well as the approaches used, are all investigated and examined. The research proved the importance of automating the process, as well as merging artificial intelligence with engineering requirements and extracting class diagrams from natural language requirements utilizing NLP tools. The author concludes that it appears that no significant effort has been taken to generate UML class diagrams from NL specifications. All of the techniques are either extremely complicated or have numerous drawbacks. A few of these systems could detect classes and build object models; however, the resulting diagrams frequently include unnecessary classes while omitting the necessary ones. Many current systems do not support some essential and enriched kinds of relationships such as generalization, association, aggregation, dependency and composition. There is no mechanism for producing entire class diagrams or other UML diagrams

automatically from free-text requirements. The majority of previous tools did not allow to see UML diagrams, and some of the current technologies required human intervention in order to build UML diagrams with related attributes and methods automatically. Just few techniques are completely automated. Moreover, many current solutions simply accept a limited number of requirements and needs developers' assistance in refining process and in identifying requirements inconsistencies. Rather than NL-free texts, the current systems almost demand that requirements to be written in a restrictive language or in a specified format.

Esra A. Abdelnabi et al [35] suggested a technique for evaluating natural language requirements and retrieving relevant software information and concepts to aid in the creation of a UML class diagram from unlimited natural language requirements. Applying NLP approaches, the author-purposed technique minimizes the uncertainty and complexities of NL. Tokenization, stemming, POS tagging, parsing, are lemmatization some of the NLP techniques employed by the author. Open Information Extraction (OpenIE) is a technique used for extracting domain relation triples, which describe subjects, relations, and their objects. To execute the transformation process, the author presented a number of heuristics rules. The findings were encouraging, and they back up the use of heuristic principles and NLP techniques to merge the benefits of automation and human thinking. A test case was used to test and assess the approach, and the findings prove that it is both satisfactory and practical.

Abinash Tripathy et al [36] describe a design activity which is utilized to create a design document from an informal specifications found in the SRS document. The modular design document illustrates the modular structure of a formal specification that is the class module's external design. As a result, every class contains the information on the class module name and function name. There are various sub-activities in the design activity. By using informal specification, every one of these creates an intermediate product. As input, a text file containing an SRS document is used. It has details of the software you're looking for. The file is then tokenized, which converts it into words. The word is checked in many tables to represent the software design process using natural language and to acquire object-oriented components, such as the Class diagram for ATM system. The process tool comprises of "eclipse indigo" for Java programming and for determining the class name, attribute, and function within it. Eight different tables are developed throughout the purpose design activity. The tokenize process takes the SRS document as input. To determine which POS the result of the tokenizer belongs to, it is checked in various tables. Both noun and verb tagged words are saved with in NON array via various POS. The array is then examined to produce the final result.

Jitendra et al [37] developed a method that interprets sentences using a richer language model based on Hornby's verb patterns and recognizes domain elements by using semantic relationships between the words within sentences acquired via type dependencies (TDs). AnModeler, a GUI-based application, is used to support the author method. With the support of 40 subjects and 40 UCSs, the author performed a control experiment to validate the proposed method. The analysis class diagrams generated by the suggested approach are significantly more correct, more

complete, and far less redundant than those produced by current automated methods, according to the results of the experimental research performed for the assessment of the approach.

Bashir et al [38] present an approach called requirement engineering analysis & design (READ) that uses domain ontology and natural language processing (NLP) techniques to build a unified modeling language (UML) class diagram. The READ system was developed in Python, and it efficiently produces a UML class diagram of textual requirements provided in English, including class name, attributes, functions, and relationships. The READ software accepts requirements expressed in English as text files (.txt). The very first step is to break down the requirements into words and sentences to make them easier to understand. Applying NLTK's tokenization approach, the author separates every sentence in the requirements and saves the results inside a list. There is only one sentence in every element of the list. After that, use NLTK's word tokenization approach to further break every sentence into words. Parts of speech tagging is the most significant phase in the purposed system. Author uses NLTK's POS-tagging mechanism to tag every word by its part of speech tag, which accepts the word like an argument and produces a Python tuple containing the word itself and part of speech tag. The precision with which NLTK's POS tagging mechanism tags the words determines the accuracy of the author purposed system. The result of POS tagging segmentation module is arranged like a python list that includes a list of sentences which are further POS tagged when it is completed. In simple words, the author claims to have a list of uniquely labeled sentences. The READ knowledge extraction module receives the segmentation module's output. Author developed certain heuristics principles to assist us in extracting class names, attributes, functions, and associations in this module. Author additionally sets up the thresholds list in this module. Such thresholds are being used to reduce the problems that earlier technologies had with over-generation. Even though some of the earlier studies presented the concept of thresholds, their concept is merely based on the entity's occurrence in requirements. They evaluate both the entity's occurrence in requirements and a list of concepts which are most frequently observed in informal requirements and, while nouns, can't be regarded a class in the author's methodology. In their suggested technique, the author presented Strong Threshold (STH) and Weak Threshold (WTH). Aside from the over-generation issue, the concept of thresholds aids us in mitigating the flaws in NLTK's POS tagging mechanism to certain level. The suggested system was assessed against publicly available standards to determine its efficiency, and the findings reveal that it surpasses existing approaches for object-oriented program design.

Rijul Saini et al [39] use natural language processing techniques with machine learning for extracting domain models from problem specifications provided in natural language. The author presented an automated and tool-supported methodology that produces more accurate extracted domain models than other methods. Furthermore, the method creates trace linkages for every domain model element. The trace links allow new modelers to run queries on the extracted domain models to learn more about the modelling decisions they made and improve their modelling abilities. Authors also suggest a new comparability metric and explain

their experimental methodology in order to assess their purposed technique. Finally, the authors propose a study roadmap that outlines research goals and challenges.

Narawita et al [40] conducted study on using Natural Language Processing to automate the creation of Unified Modeling Language (UML) diagrams from analyzed requirements. The method has been evaluated with over twenty (20) cases by the author, and it has a 70% accuracy level. This figure was derived based on all of the scenarios' passed test case results.

Conte et al [28] aims to use natural language processing to help in class diagram creation. For this, an application prototype is modeled and implemented in order to validate the proposal. The initial evaluation of the use of the tool was considered satisfactory.

Bousmaha et al [42] introduced a framework (AL2UML) for semi-formalizing specification requirements from NL to a UML class diagram. The author proposes AL2UML for the conceptualizing the information system specification texts, with a focus on the construction of UML class diagrams using hybrid methodologies that combine statistical and linguistic approaches. The reality that the specification texts are in Arabic is a unique addition of their effort. The Arabic language is particularly difficult due to its complicated linguistic features. They created the Alkhalil+ software to create the linguistic model as well as to proceed with the pretreatment of the requirements texts. It performs a wide range of tasks including tokenization, MSA, segmentation, lemmatization, morphological analysis, part-of-speech (POS) tagging, discretization and disambiguation. The experimentation yielded a disambiguation accuracy of more than 85%. An XML file having information required for conceptual model is produced as the result. The chunker is designed in the second step by Author. They created a list of chunk categories which were required for sentences classification and the extraction of meaning. The author then conducts a sematic assessment of their work. In order to represent the entirety of the retrieved data, a semantic network is constructed as an output. After that, they created the UML Class diagram.

To construct the class model using software specifications, Ahmed et al [43] suggest a unique and completely automatic NLP approach. The innovative rules of sentence splitting, tokenization, and POS tagging provided by the author are included in the suggested NLP technique. To produce a conceptual class model, the given rules were implemented to the initial simple text requirement specification. The author's suggestion focuses on using matched nouns for extracting Noun Plural (NNS), Proper Noun Singular (NNP), and Proper Noun Plural (NNPS). To eliminate repetition among classes, the author defines a vocabulary that contains all unnecessary glossary words such as user, software, number, and so on. The Automated Requirement 2 Design Transformation (AR2DT) software was created as part of the study to automatically produce class models with code using plain text specifications. AR2DT doesn’t currently support the creation of class attributes, functions, or relationships such as inheritance, composition or aggregation.

Utama et al [44] propose a method for extracting class diagrams from problem statements. The name of the class, its attributes, and its operations are all shown in a class diagram. Natural Language Processing can be used to generate a class diagram from a problem statement automatically. Tokenization, stemming, POS tagging, parsing, lemmatization, and chunking are all used by the author using the NLP software Spacy. The algorithm and preprocessing stage have a significant impact on the extracting outcomes. The algorithm was derived from a variety of sources, with extra rules added during the implementation stage.

2.10 Answers to Research Questions

RQ1: What are the major NLP techniques, tools, and algorithms for the generation of Class Diagram from Natural Language Requirements?

2.10.1 Natural Language Processing Techniques

The Tokenization, POS tagging, Chunking and Parsing are the basic NLP techniques to process the initial plain text requirements. The utilization of major natural language processing techniques, in the context of first two SDLC phases, has been summarized in Table 6**.** It can be analyzed from the Table 6 that these major NLP techniques are utilized by the researchers exclusively as well as jointly. There are researches that do not explicitly define the utilization of NLP techniques in their study but specific tool has been used to achieve the desired objectives.

Table 6 Natural Language Processing (NLP) techniques utilized in different articles

Sr. No.	NLP Techniques	Relevant Articles	Total
1	POS Tagging	[33],[42]	2
2	Tokenization+ Parsing	[36]	1
3	POS Tagging +Parsing +Type Dependency (TD)	[37]	1
4	Tokenization + Stemming+ POS Tagging + Stop Words	[38]	1
5	POS Tagging + Tokenization + Parsing	[39]	1
6	POS Tagging + Tokenization	[40],[28]	2
7	POS Tagging + Stop words Removal	[34]	1
8	POS tagging + Stemming + tokenization + parsing	[29],[30]	2
9	POS tagging + Stemming + tokenization + parsing + Lemmatization	[35]	1
10	POS tagging + dependency Parsing	[31]	1
11	Stop words Removal + Lemmatization + Parsing	[32]	1
12	Sentence splitting + tokenization + POS Tagging	[43]	1
13	POS tagging + Stemming+ tokenization + parsing + Lemmatization + Sentence Splitting	[41]	1
14	POS tagging + Stemming + tokenization + parsing + Lemmatization + Chunking	[44]	1
Total Articles			**17**

2.10.2 Natural Language Processing Tools

2.10.2.1 Tools Utilized by Researchers

There are few existing tools, utilized by the researchers, in order to achieve particular research objective. So, it is important to identify and highlight such NLP existing tools as shown in Table 7. NLP tools are used to perform certain relevant activities like parsing, text chunking etc. We identified 11 NLP tools that is used by researchers. Most of the researchers do not mention the specific NLP tool kit as we can see in Sr. No.1 of Table 7.

Table 7 NLP tools utilized by researchers

Sr. No.	Tools Utilized by Researchers	Relevant Article	Total
1	NLP	[28], [31], [36],[42],[41]	5
2	NLTK	[33]	1
3	Stanford core NLP	[30], [35]	2
4	Stanford Parser- v3.5.2	[32]	1
5	Stanford Parser	[34]	1
6	NLP+ used richer language model and AnModeler	[37]	1
7	NLTK + Domain ontology techniques	[38]	1
8	SpaCy	[39],[44]	2
9	SharpNLP	[40]	1
10	SharpNLP-1.0.2529 + Wordnet	[43]	1
11	Stanford core NLP + Wordnet	[29]	1
Total Articles			**17**

2.10.2.2 Tools and Algorithms Purposed by Researchers

We identify 12 algorithms, proposed/utilized by researchers to develop particular tools. Table show the different Algorithms against the tools that is identified from the different articles. Provide the foundation to develop particular NLP-based tool for the generation of required SDLC artifact. For example, [38] employ READ algorithm to develop READ tool.

Table 8 NLP algorithm used by researchers in purposed tool

Sr. No.	Tool Name	Algorithm	Relevant Article
1	UML class diagram generation	Design Element Extraction	[29]
2	Conceptual Model	Rule based approach	[30]
3	Class diagram	solution approach based on syntactic dependency analysis and GKPs	[34]
4	UML Class Diagram	NLP+ heuristic rule algorithm	[35]
5	Class name	Algorithm to Extract class name by analyzing test file	[36]
6	Identify domain elements	systematic NLP rule-based approach	[37]
7	READ	READ Algorithm for class generation	[38]
8	UML generator	NLP+XML rules	[40]
9	AR2DT	rule based algorithm to generate class diagram	[43]
10	AL2UML	Rule based Algorithm to process Arabic Language	[42]
11	Automated class diagram	extract classes and attribute algorithm	[44]
12	LIDA, CM Builder, RE-Builder UML, UMLG, RACE, RAPID, RAUE, DC-Builder, ABCD, SUCM	LIDA model Algorithm, Parsing and conference algorithm for CM Builder, RAUE filtering Algorithm for RAUE	[41]

RQ2: What level of automation is achieved through the NLP for the generation of class diagrams?

2.10.3 Automation Level of Design Phase

To this point, we already identified and present the significant NLP techniques, tools and algorithms for the generation of requirements, and design artifacts of SDLC through the Systematic Literature Review. In this section, we perform the detailed analysis of leading class generation approaches as summarized in Table 9. It has been analyzed that the design phase studies mostly deal with the generation of class diagram. Therefore, we deeply examine the significant class generation studies) with important evaluation parameters because in many articles researchers claim that they identify the complete elements of the class diagram and

remove the redundancy between the classes but they do not identify all the elements of the class .so we set some evaluation parameters to check the automation level of class diagram. The selected articles were evaluated against the following aspects, each on a pre-defined scale of 0 to 1. This was done to check automation level of class diagram. At the end, total Score of each article tell us the automation level of each.

So, following are the evaluation parameters of the class diagram to check the Automation Level

- CEP1: Articles deal with Classes- Yes (+1), NO (0).
- CEP2: Articles deal with Attributes- Yes (+1), NO (0).
- CEP3: Articles deal with Functions- Yes (+1), NO (0).
- CEP4: Articles deal with Association- Yes (+1), NO (0).
- CEP5: Articles deal with Direction Association- Yes (+1), NO (0).
- CEP6: Articles deal with Inheritance/Generalization- Yes (+1), NO (0).
- CEP7: Articles deal with Realization- Yes (+1), NO (0).
- CEP8: Articles deal with Multiplicity- Yes (+1), NO (0).
- CEP9: Articles deal with Aggregation- Yes (+1), NO (0).
- CEP10: Articles deal with Composition- Yes (+1), NO (0).
- CEP11: Articles deal with Redundancy- Yes (+1), NO (0).

Table 9 Level of Automation of Class Diagram using NLP

Sr. No.	Ref.	Class	Attributes	Functions	Association	Direct Association	Inheritance / Generalization	Realization	Multiplicity	Aggregation	Composition	Redundancy	Total
1	[28]	1	1	0	1	0	0	0	0	0	0	0	3
2	[29]	1	1	1	1	0	0	0	0	1	0	1	6
3	[30]	1	1	1	1	0	1	0	0	1	0	0	6
4	[32]	1	0	0	1	0	0	0	0	0	0	0	2
5	[33]	1	0	0	0	0	0	0	0	0	0	0	1
6	[34]	1	1	0	1	0	1	0	0	0	0	0	3
7	[35]	1	1	1	1	0	1	0	0	1	1	0	7
8	[36]	1	0	0	1	0	0	0	0	0	0	0	2
9	[37]	1	1	1	0	0	0	0	0	0	0	0	3
10	[38]	1	1	1	1	0	0	0	0	0	0	0	4
11	[39]	1	0	0	0	0	0	0	0	0	0	0	1
12	[40]	1	1	1	1	0	0	0	0	0	0	0	4
13	[43]	1	0	0	1	0	0	0	0	0	0	0	2
14	[44]	1	1	1	1	0	0	0	0	0	0	0	4

2.11 Limitations

The findings of this study are done against a defined methodology. There might be some relevant studies which fall beyond the boundaries of the methodology and hence, are missing from this study.

Cognitive Bias: the keywords selected to represent the search query were thought by the authors. There can be synonyms or other words, used in articles, to represent the same concepts.

Selection Bias: the articles selected for this study were taken from conferences or journal of five e-repositories, published between 2014 and 2021. Other repositories might have relevant literature. Also, there can be other articles in the same repositories that were published beyond the decided timeline and address the same concerns**.**

2.12 Conclusion

This paper investigates the Natural Language Processing (NLP) applications to automatically create the class diagram from early plain text requirements. A systematic literature review (SLR) was conducted to select 29 studies that were published between 2014 and 2021. After quality evaluation found 17 articles that helps to answer the research questions.

Consequently, 14 combinations of main NLP techniques (i.e., Tokenization, POS tagging, Chunking and Parsing e.tc) are identified. Moreover, NLP algorithms are identified that purposed by researchers to for their tools furthermore, overall, 12 tools, proposed by the researchers, are presented. In additional, 11 existing tools, utilized by the researchers in the given research context. It is concluded that NLP demonstrates promising outcomes for the automation of software development phases. For example, design artifacts like class diagrams are effectively generated from initial requirements by utilizing various NLP techniques. Furthermore, it is also concluded that some sort of manual processing is required on early requirements before applying NLP techniques for the generation of desired SDLC artifact i.e., requirement and design phase.

The findings of SLR like NLP techniques, tools and algorithms are highly beneficial for the students, developers and researchers of the domain. Furthermore, there is a fair possibility to extend this research in multiple directions. For example, one probable area would be the empirical investigation of identified tools through certain parameters like usability, operational complexity etc. Another interesting area would be the comparative analysis of the identified tool development algorithms.

CHAPTER 3
METHODOLOGY

3.1 Introduction

Getting adequate and desired information from preliminary data in the initial and the most the critical phase of analysis requires a lot of manual intervention which results in huge processing time. Additionally, such manual interventions in data processing cause a lot of errors if overall data is complex and huge. To overcome these issues, Natural Language Processing (NLP), a field of text mining shows some encouraging and propitious results especially in bio-medical domain [57]. Sentence splitting, tokenization, POS tagging, and other natural language processing (NLP) techniques provide complex and automated data processing features and now it is applied over different software development phases to automatically generate the requirement specifications [58], and design models which includes automated use case generation [59], class diagram generation [59], activity diagram generation [60] and so on from natural language user requirements.

As said earlier, the requirement identification and analysis phase are most critical stage in software development as analysis of software user requirements is extremely crucial task in the software development life cycle (SDLC), as input to this stage is mostly in natural language which is ambiguous. Communication between the project stakeholders i.e., users and developers are in natural language, as developers also need to analyze the requirements document was written in the plain natural language i.e., English. Advantage of using natural language for this purpose is the freedom of expression as each and everything can be described by using natural language. But on the other hand, still natural language creates many problems for the software designers because of its error-prone nature and ambiguities in requirements written in plain natural language can be interpreted in different ways and software designers must tackle a huge number of requirements which will create difficulties for them.

During the requirements analysis phase, analysts, and project stakeholders simultaneously gathers the requirements for the project. These preliminary requirements serve as an input to the next phases of software development for the planning of efforts and schedule. This needs to be completed as early as possible. Requirements change during any phase of the project will be more and more expensive than the requirement phase itself. The biggest problem with natural language requirements is that stakeholders assume that analysts know everything regarding requirements. Such problem in the initial and most critical phase will cause a disagreement between the project stakeholders i.e., users and analysts much later, for once or in later stages of the project that will consume a lot of time and effort to fix it.

To overcome the requirements change from the users, most projects need a sign off of the preliminary requirements by all the relevant stakeholders i.e., users, analysts, designers and testers. Requirement's analysis phase is subjective in nature and highly dependent on opinions of

personals. Some project stakeholders i.e., designers got trouble understanding the natural language user requirements without any analyst's assistance. It is significant for the stakeholders i.e., designers & developers to understand the user requirements presented by the analyst so that to understand the user requirements in an efficient manner.

Different tools exist to visualize and render requirement specification, but none of them help the analyst during the initial and critical phase. The conceptual model is a relevant and useful artefact for visualizing user requirements. In a problem domain, a conceptual/domain model depicts entities and relationships. Only domain entities & attributes are defined by Larman et al [61], which is a static model of the system. On the other hand, Booch et al [62] presents the OOAD model, which resembles a UML class diagram, which includes classes, attributes, operations and relationships and is a dynamic model of the system. A conceptual/ domain model is considered as an efficient model in terms of visual communication as of this type of model consumes less space than the natural language requirements and conveys the maximum information. In newer software development models, the use of the conceptual model is recommended, like in agile modeling [63] and adopted in OO development models by using UML [64].

LindLand et al [65] proposed a quality framework for the conceptual model validation for the first time in history which basically covers three types of quality models: pragmatic quality model, semantic quality model, and syntactic quality model. Pragmatic quality model addresses model efficiency and ensure that the audience understand the information given in the model or not. The semantic model addresses the validity and completeness of the model keeping in mind the target domain while the syntactic quality model addresses the correctness of the model by using formal syntax. Every quality model elaborates some goals and objectives. Using such a quality framework in a project's initial phase will make a suitable way for the systematic analysis of requirements in visual terms.

In this paper, a methodology is proposed to focus on the automated extraction of classes and their attributes and relationships to create a conceptual model i.e., class diagram from natural language plain text in an effective way. The main goal is to create a class diagram that will help the designers in the later phases of software development, and it will help the requirements analyst too to help the other stakeholders of the project to understand the requirements in visual form (as told earlier the advantages of visual communication).

Another advantage of automation of this step is: the focus of the designer will be on model refinement rather than by creating a manual class diagram. This will help the designer to communicate with requirement analysts and other stakeholders and due to this, analysts can do multiple iterations of requirement analysis refinement which results in the requirements in which all the stakeholders feel comfortable.

Overall overview of this research is given in following Figure 6.

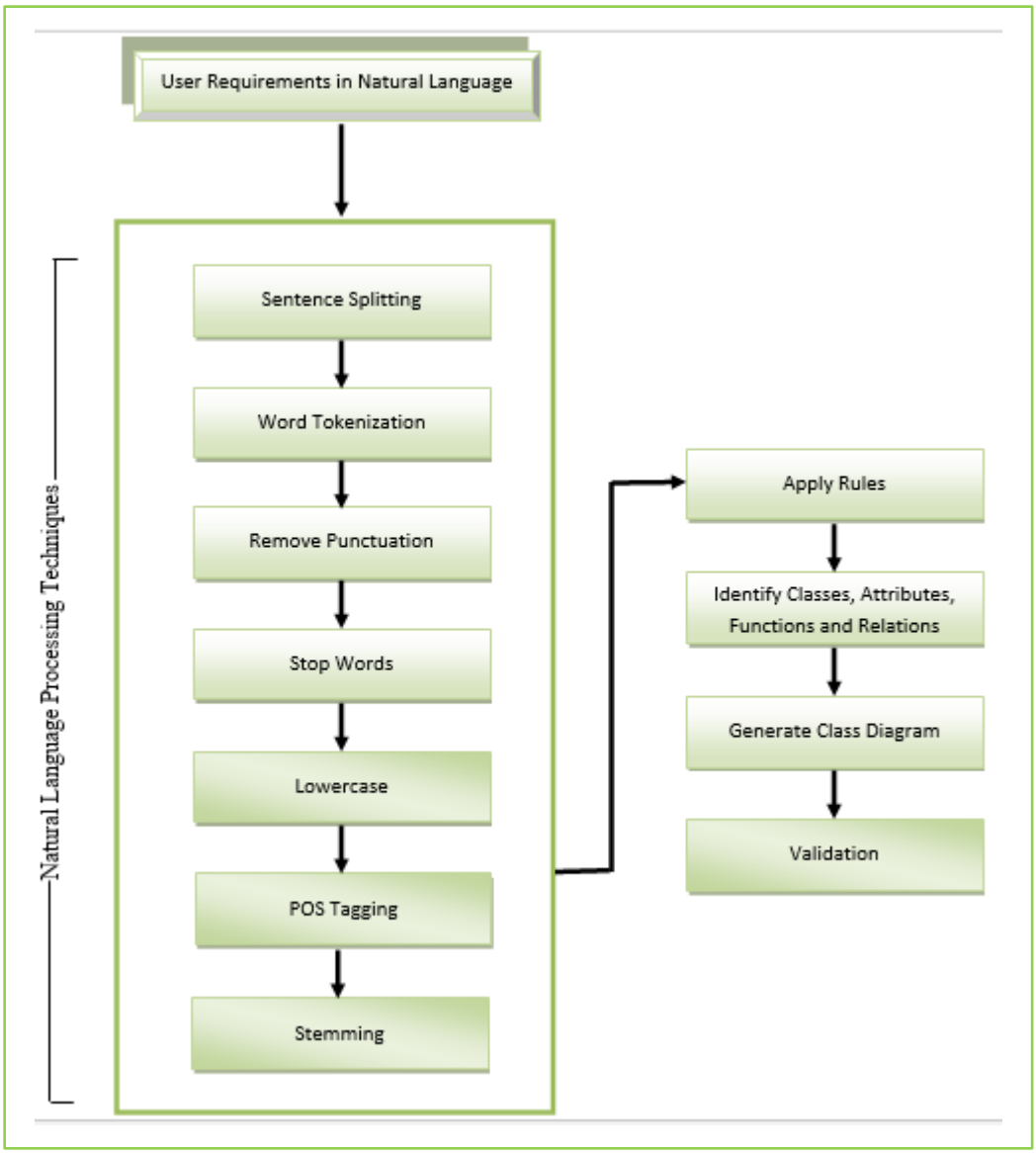

Figure 6 Overview of research study

In this thesis, a class diagram tool named SD-LINGO is developed for the automatic identification of classes, attributes, and relationships from raw requirements written in plain English language. SD-LINGO is validated over different case studies and results are presented in (Chapter 04). First existing literature regarding class diagram generation from user requirement using NLP is reviewed including NLP techniques that transform plain text to conceptual class model. Also reviewed and examined the rules to convert the natural language text to conceptual class model i.e., class diagram. SD-LINGO process the case study sentence by sentence. When one sentence in the case study is processed by the system then it moves to the next sentence. This research work starts with the identification of classes, attributes, and relationships from the plain text while using existing NLP tools named NLTK to identify noun phrases, adjectives, verb

phrases etc. This step is processed in our research based on the existing work. After POS tagging, the process of extraction of classes, attributes, and relationships is based on a set of rules.to remove the redundancy between the classes wordnet library is used that helps to find the synonyms of words. When all the processing is done then the system generates the class diagram.

3.2 Strategy and Design of the Proposed Tool

SD-LINGO is a natural language-based CASE tool that is used to perform object-oriented analysis. Natural language plain text is used as input to this tool, which is linguistically evaluated to extract classes, attributes, functions, and relationships. SD-LINGO is written in python with approximately twenty-four hundred lines of code. The tool used for front-end development is Spyder IDE 5.1.5. SD-LINGO produces all the elements of a class diagram such as classes, attributes, function, and relationships (Association, Direct Association, Inheritance/Generalization, Multiplicity, Realization). Python offers a number of alternatives for creating graphical user interfaces (GUIs). For designing the GUI of the tool, we used Tkinter. Tkinter is Python's standard GUI library. We chose Tkinter because it allows us to construct GUI apps quickly and easily using Python and Tkinter. Tkinter gives the Tk GUI toolkit a strong object-oriented interface.

The steps are summarized as follows:

- Take a set of natural language software requirements (plain English text).
- Used NLTK tool kit of NLP to identify noun phrases, adjectives & verb phrases from natural language text.
- Apply rules and identify all the elements of class.
- Generate a class diagram.
- All the processes of the presented tool are fully automated.

3.3 Proposed Methodology and Implementation

3.3.1 Natural Language Software Requirements

There is not any standardized format or template to writing natural language software requirements. As input to SD-LINGO is in natural language which is plain text, so obviously, it is inherently ambiguous and error-prone. The basic challenge to our tool is the plain text which is obviously our aim too. SD-LINGO will accept natural language software requirements as input.

3.3.2 Pre-Processing

Minor pre-processing steps are applied to the plain text for consistency purposes. Not so much altered the document so that to alive the subject of research of having natural language text as an input. Pre-processing steps are given as follow:

Rule no 1: Select abbreviations or full descriptions

In this rule, we have decided to select only one thing in description i.e., Abbreviation or a word whose description is given. For example, in the ATM case study word ATM is stated like this automatic teller machine (ATM) so we have to ignore automatic teller machine and just written ATM instead of automatic teller machine.

Rule no 2: Remove brackets (if any)

As described above we have selected (ATM) instead of an automatic teller machine. But in the passage, it will be with brackets around ATM which some time misses in the phase of POS tagging so we have eliminated such types of brackets to ease and accuracy of the POS tagging phase.

Rule no 3: Remove unnecessary full stop like if there is any e.g., remove it because it makes classes

We discovered 'e' and 'g' as nouns during the POS tagging step due to the purpose of writing 'e.g.,' that is being used for the purpose of short cut to “for example” so we recommend to remove such types of things from text to get correct entities only.

Rule no 4: If there are any examples to explain requirements remove it

Same in the case of things, which are not part of requirements, but are included in the text. To just explain the idea with different examples. We recommend removing such types of examples from the text. Like in the case study of the Bank Account management system they have exampled the joint account and given the husband-and-wife account for that purpose which is not part of the requirement. Husband and wife are nouns and if we process them as it is they will be our classes so it necessary is to remove such types of things.

Rule no 5: Unnecessary capital words into lower case

In some case studies or textual descriptions, we have found the unnecessary capital words which are not necessary like:

- The branches are considered subdivisions of the Consumer Division. The above the statement has C capital in Consumer and D capital in Division which is not necessary. We recommend converting such words in lower case and then put into the tool for further process. The processed sentence is “the branches are considered subdivisions of the consumer division”.

3.3.3 Comprehension of input using Natural Language Toolkit NLTK [66]

- NLTK is a popular Python environment for working with human language data. It includes a set of text processing libraries for tokenization, classification, tagging, stemming, semantic reasoning and parsing, as well as wrappers for industrial-strength

NLP libraries and an active discussion forum, as well as convenient interfaces to over 50 corpora and lexical resources like WordNet.

- Sentence Splitting: This tool's feature divides a big text file into a series of sentences, reducing the text file's complexity.
- Tokenization: This tool's functionality turns sentences into tokens, which includes punctuation, words, and numbers being separated.
- Stop words are terms that we want to omit, therefore the system removes them from the text as it is processed. Words like 'is,' 'in,' and 'an' are commonly utilized as stop words because they don't offer much significance to a text.
- Stemming is a text processing activity that involves reducing words to their root that is the most fundamental portion of the word. The root "help" is shared by words like "helping" and "helper." Stemming helps to focus on a word's essential meaning instead of the specifics of how this is being utilized. Although NLTK has several stemmers, we'll use the Porter stemmer.
- POS-tagging: Part of speech is a grammatical term that refers to the roles that words play in sentences when they are used together. The process of classifying the words within your text according to their part of speech, is known as POS tagging.
- Chunking: This capability of tool provides the detection of phrases boundaries. E.g., verb phrases chunk in a sentence.
- Parsing: This capability of tool provides formal description of a given sentence in a language called a grammar in an allowed structure.

Example of POS tagged extracting nouns, verbs, adverbs and adjectives from natural language requirements are given in Figure 7. Table 10 contains a list of POS tagged words and their abbreviations.

```
[('A', 'DT'), ('local', 'JJ'), ('hospital', 'NN'), ('consists', 'NNS'), ('of', 'IN'), ('many', 'JJ'), ('wards',
'NNS'), ('each', 'DT'), ('of', 'IN'), ('which', 'WDT'), ('is', 'VBZ'), ('assigned', 'VBN'), ('many', 'JJ'),
('patients', 'NNS')]
[('Each', 'DT'), ('doctors', 'NNS'), ('are', 'VBP'), ('assigned', 'VBN'), ('1', 'CD'), ('patients', 'NNS'), ('who',
'WP'), ('has', 'VBZ'), ('overall', 'JJ'), ('responsibility', 'NN'), ('for', 'IN'), ('the', 'DT'), ('patients',
'NNS'), ('in', 'IN'), ('his', 'PRP$'), ('or', 'CC'), ('her', 'PRP$'), ('care', 'NN')]
patient
doctor
[('Other', 'JJ'), ('doctors', 'NNS'), ('are', 'VBP'), ('assigned', 'VBN'), ('on', 'IN'), ('an', 'DT'), ('advisory',
'JJ'), ('basis', 'NN')]
[('Each', 'DT'), ('patient', 'NN'), ('is', 'VBZ'), ('prescribed', 'VBN'), ('drugs', 'NNS'), ('by', 'IN'), ('the',
'DT'), ('doctor', 'NN'), ('responsible', 'JJ'), ('for', 'IN'), ('that', 'DT'), ('patient', 'NN')]
[('Each', 'DT'), ('nurse', 'NN'), ('is', 'VBZ'), ('assigned', 'VBN'), ('to', 'TO'), ('a', 'DT'), ('ward', 'NN'),
('and', 'CC'), ('nurses', 'NNS'), ('all', 'DT'), ('patients', 'NNS'), ('on', 'IN'), ('the', 'DT'), ('ward', 'NN'),
('though', 'NN'), ('is', 'VBZ'), ('given', 'VBN'), ('special', 'JJ'), ('responsibility', 'NN'), ('for', 'IN'),
('some', 'DT'), ('patients', 'NNS')]
[('Each', 'DT'), ('patient', 'NN'), ('is', 'VBZ'), ('assigned', 'VBN'), ('one', 'CD'), ('nurse', 'NN'), ('in',
'IN'), ('this', 'DT'), ('position', 'NN'), ('of', 'IN'), ('responsibility', 'NN')]
[('one', 'CD'), ('of', 'IN'), ('the', 'DT'), ('doctors', 'NNS'), ('is', 'VBZ'), ('attached', 'VBN'), ('to', 'TO'),
('each', 'DT'), ('1', 'CD'), ('ward', 'NN'), ('as', 'IN'), ('an', 'DT'), ('overall', 'JJ'), ('medical', 'JJ'),
('advisor', 'NN')]
ward
doctor
```

Figure 7 Example of POS tagged sentence

3.3.4 Rules Used by SD-LINGO Tool

This section includes the set of rules that provides some correspondence among the natural language sentences and OO modeling. This set of rules is applied over the natural language requirements for the extraction of knowledge using natural language processing. The specific set of rules is given below for determining the entities (classes), attributes, and relationships.

Purposed Tool has a list for common attributes, non-attributes, common functions, non-function, and common classes and non-classes. When the system identifies any tag such as noun, verb, and adjective, etc. then firstly it matches that identified to the list that is mentioned above. These lists are helping the system to get valuable information from requirements and increase the accuracy of our system.

These lists (common attributes, non-attributes, common functions, non-function, common classes, and non-classes) are updated over time. When the lists become larger and become more versatile than it helps the system to enhance the accuracy with the best and accurate results.

3.3.4.1 Rules for the Identification of Classes, Attributes, and Functions

Rule No. 1:

If the system finds a token having a tag of NNS, then it checks the token in the non-classes list if that token does not exist in the list of non-classes, then the system uses that token as a class and add that token in classes-list and set that class as an active class for the attributes and functions.

Rule No. 2:

If the system finds a token having a tag of VB, then it checks the token in the list of the common attributes if that token exists in the list of common attributes, then the system adds that token in the list of attributes or functions according to the system other rules that are used for the identification of function and attributes.

Rule No. 2 also activates Rule No.4 and Rule No.15 for further processing.

Rule No. 3:

If the system finds a token having a tag of NN, and the system doesn't have any active class before then it checks the token in the list of non-classes if the identified token is not in the list of non-classes, then the system adds this token in classes-list and set that class as an active class if the system does not find any active class before.

If the system finds any active class, then it checks that token in the classes-list if that token exists in the class-list then the system set that token as an active class otherwise system used that token for the attribute or function according to other rules that are used for the identification of attributes and function.

Rule No. 3 is also responsible for activating Rule No.8, Rule No. 9, and Rule No. 13and also set the first combination for Rule No. 8 and Rule No. 11.

Rule No. 4:

If the system finds a token having a tag of DT, then the system activates the Rule No.4 Combination flag, and deactivates Rule No.4.

- **Rule No. 4 Combination Flag:** If the system gets a token having a tag of JJ, then it activates the Rule No. 4 combination-2 flag and deactivates the Rule No. 4 combination flag and sets the token as the first combination for Rule No. 4.
  If the system does not get the JJ tag, then it deactivates the Rule No. 4 combination and sets the first combination to unknown.
- **Rule 4 Combination-2 Flag:** If the system finds a token having the tag of NN then it makes sure that this token is not in the list of non-attributes if the token does not exist in the non-attributes list, then it combines that token with the first combination and insert in attributes of the active class. If the system finds that token in the functions list, then we combine it with the first combination and insert it in the active class functions list. If the system gets NN then firstly system make sure that this token is in the list of the common attributes if exist in this list, then it combines with the first combination and insert in active class attributes.

Rule No. 5:

If Rule No.4 flag is deactivated and the system has an active class and the system found the tag of DT, then it activates the Rule No. 5 combination flag. The system also activates the DT hunt flag.

- **DT Flag:** If the System identifies the token with NN then the system sets it as the first combination for Rule No. 5.
- **Rule 5 Combination Flag:** If Rule No.5 combination flag is true and the system gets the token with NN and active class name, not in the non-classes list then the system combines the first combination and adds in attributes or functions according to the rules for functions and attributes.

Rule No. 6:

If the system identifies the token having the tag of JJ and if it is at the last index of the sentence and the active class is not in the list of non-classes, then the system inserts the current token in the attributes or functions list according to the rules. It also activates the Rule No.6 combination flag if the system is not at the last index of the sentence.

- **Rule 6 Combination Flag:** When Rule No. 6 combination flag is activated, then the system checks if the current token is not in the non-attributes list and not in the active

class attributes list then the system inserts the current token in the current class attributes list that is active on that time.

Rule No. 7:

If the system gets the token having the tag of DT, then it activates the Rule No.7 and Rule No. 10 flag.

- **Rule 7 Flag:** If Rule No. 7 flag is activated and the system gets the token having a tag of NN then it activates the Rule No. 7 combination flag and sets the current token as a Rule No. 7 first combination.
- **Rule 7 Combination Flag:** If Rule No. 7 combination flag is activated and the system gets the token having a tag of NN then the system combines the Rule No. 7 first combination and the current token and inserts it in the active class as attributes or functions according to the rules.

Rule No. 8:

If Rule No. 8 is activated and the system gets the token having a tag IN, then the system activates the Rule No.11 and Rule No. 8 combination flag.

- **Rule No. 8 Combination Flag:** If the Rule No. 8 combination flag is on and the active class is not in the non-classes list, then the system then the system again checks the next token that is followed by IN and insert that token with the Rule No. 8 combination (that gets in Rule No. 3) in the active class attributes list.

Rule No. 9:

If the Rule No. 9 activation flag is true (that is activates in rule 3) and the system gets the token having the tag of CC, then the system activates the Rule No. 9 insertion flag.

- **Rule No. 9 Insertion Flag:** If the Rule No. 9 insertion flag is activated then the system inserts the current token that is find in the Rule no 9 insertion flag in the active class attributes list.

Rule No. 10:

If the Rule No. 10 flag is activated (activate in rule 7) and the active class flag is false, then the system activates Rule No. 10 Step 2.

- **Rule No. 10 Step 2:** When Rule No. 10 step 2 flag is activated, and the system gets the token having the tag of NN then the system activates the Rule No. 10 step 3 flag.
- **Rule No. 10 Step 3:** When Rule No. 10 step 3 is activated, and the system gets the token having the tag of DT then the system activates the rule 10 step 4 flag.

- **Rule No. 10 Step 4:** When Rule No. 10 step 4 is activated, and the system gets the token having the tag of NN then the system sets the current token as the first combination for Rule No. 10 and activates Rule No. 10 step 5.
- **Rule No. 10 Step 5:** When Rule No. 10 step 5 is activated, and the system gets the token having tag VBZ then the system set the first combination that identified in Rule no 10 step 4 that is NN as a class name and insert in the classes list and combines the first combination of Rule No.10 and the current token that is VBZ and insert the combination of NN +VBZ as a class attribute.

Get NN for Class name and then add NN+VBZ as a class attribute.

Rule No. 11:

If the Rule No. 11 flag is activated (that activates in Rule No.3) and the system gets the token having the tag of DT, then the system activates the Rule No. 11 step 2.

- **Rule No. 11 Step 2:** If Rule No. 11 Step 2 is activated and the system gets the token having the tag of NN or NNS or NNP then the system creates a new class of (NN, NNS, or NP) and insert the combination for Rule No. 11 (get in Rule No. 3) that is NN+DT in the attributes of this class. In other words, create the class NN that is before the DT and NN insert as an attribute that is after the DT.

Rule No. 12:

If the system gets the token having the tag of CD, then the system activates the Rule No. 12 Step 2 Flag.

- **Rule No. 12 Step 2 Flag:** If Rule No. 12 Step 2 flag is activated and the system gets the token having a tag of NN then it inserts the current token in the active class attributes.

Rule No. 13:

If Rule No.13 activation flag is true (activates in rule 3) and the system gets the token having the tag of WP, then the system activates the Rule No. 13 Step 2.

- **Rule 13 Step 2:** If Rule No. 13 step 2 is activated and the system gets the token having tag NN then the system inserts the current token in the active class attributes list.

Rule No. 14:

If the system gets the token having the tag of TO and the system has an active class, then it inserts the current token in the functions of the active class if the current token does not exist in the non-function list.

3.3.4.2 Rules for the Identification of Relationships

Association

If the system found two or more classes in a single sentence, then it declares an association between these classes and if the system found any other relationships between these classes, then it drops this relationship according to the relationship priority.

Direct Association

If the system gets the token having the tag of MD system activate the direct association flag and if the system finds any function that is in the functions list of the current class and after that if the system found a new class, then it declares a direct association between these two classes.

Multiplicity

If the system found a token having a tag of CD system stores this tag as a multiplicity number and after that, if the system found a new class, then it declares a multiplicity relationship between these classes and shows the number toward the second class and many (*) towards the first class also if system found class and after that it found CD tag and then find new class in next token then it declares the Multiplicity between the classes.

Generalization/Inheritance

If the system found a class and in the next token system again find a new class, then it declares these classes as inherit classes and if found words like a child, is kind of, is type of, generalized, is categorized, and similar words and after that found new class then the system also declares inheritance or generalization.

Realization

If the system found a function and this function is the same as other classes that have any relationship with this class, then we declare realization between these classes and function show on the relationship line.

3.3.4.3 Rule for Handling the Redundancy

To overcome the redundancy such as synonyms in the natural language requirement our system uses the library of word net which provides the facility to the system to check the different synonyms of the words. When the system finds a token firstly it checks the synonyms of that token if synonym of that token already exists in the classes list, then it ignores the current token and insert the attributes and functions of that token to the existing class that is synonym of the token.

Table 10 NLTK POS tag list.

Sr. No.	Abbreviation	Meaning
1	CC	Coordinating Conjunction
2	CD	Cardinal Digit
3	DT	Determiner
4	EX	Existential there
5	FW	Foreign Word
6	IN	Preposition/Subordinating Conjunction
7	JJ	This NLTK POS Tag is an Adjective (large)
8	JJR	Adjective, Comparative (larger)
9	JJS	Adjective, Superlative (largest)
10	LS	List Market
11	MD	Modal (could, will)
12	NN	Noun, Singular (cat, tree)
13	NNS	Noun Plural (desks)
14	NNP	Proper Noun, Singular (Sarah)
15	NNPS	Proper Noun, Plural (Indians or American's)
16	PDT	Predeterminer (all, both, half)
17	POS	Possessive Ending (parent\ 's)
18	PRP	Personal Pronoun (hers, herself, him, himself)
19	PRP$	Possessive Pronoun (her, his, mine, my, our)
20	RB	Adverb (occasionally, swiftly)
21	RBR	Adverb, Comparative (greater)
22	RBS	Adverb, Superlative (biggest)
23	RP	Particle (about)
24	TO	Infinite marker (to)
25	UH	Interjection (goodbye)
26	VB	Verb (ask)
27	VBG	Verb Gerund (judging)
28	VBD	Verb Past Tense (pleaded)
29	VBN	Verb Past Participle (reunified)
30	VBP	Verb, Present Tense not 3rd person singular(wrap)
31	VBZ	Verb, present tense with 3rd person singular (bases)
32	WDT	Wh-determiner (that, what)
33	WP	Wh-determiner (that, what)
34	WRP	wh- adverb (how)

3.3.5 SD-LINGO Design Tool

For designing the tool, we used the Tkinter package .it is the standard Python interface for the GUI toolkit. When python combines with the Tkinter it provides a fast and very easy way to create GUI applications [67].

3.3.5.1 Class Diagram Drawing in SD-LINGO

Firstly, we find the center of the canvas then finds the class that has maximum relationships than the other classes. We draw that class that has a maximum relationship at the center of the canvas. After drawing the first class our system knows their current position on canvas because before printing first-class system does not know the position. The system finds the second class that has maximum relationship but less than the first class that is already drawn at the canvas. And then draw second class at the right side of first-class on the canvas. After that system finds the third class and draws on the left side of first-class on the canvas. We draw only three classes in one row. When the system prints three classes in the center of the canvas in one row then it draws the other three classes on the top of the first classes in one row and then the bottom of the first class by following the procedure of drawing that is already discussed.

For relationships we set the priority to each relationship such as association priority is 1, direct association priority is 2, multiplicity priority is 3, inheritance/generalization priority is 4, and realization priority is 5. Higher numbers have high priority and low numbers have the lowest priority. For showing classes relationship system firstly checks the total relationships and after sorting the value of the relationships according to priority picks the highest priority relationship and discards the other relationship. And store all high-priority relationships in their dictionary. After storing the relationship in the dictionary system find the starting class and target class of that relationship. The system automatically finds the path for relationships from starting class to the target class without crossing the other classes. If the target class is on the left side of the class, then it draws the line automatically from the left side of starting and follow the same procedure for other target class that is at the top, bottom, and right side of the starting class. And also, we give color to each relationship such as for association color is green, color for the direct association is Brown, color for generalization and inheritance is orange, color for multiplicity is navy blue, and color for realization is purple.

CHAPTER 4
EXPERIMENTAL VALIDATION

4.1 Local Hospital Problem Case Study

4.1.1 Problem Statement

A local hospital consists of many wards, each of which is assigned many patients. Each doctors are assigned 1 patients, who has overall responsibility for the patients in his or her care. Other doctors are assigned on an advisory basis. Each patient is prescribed drugs by the doctor responsible for that patient. Each nurse is assigned to a ward and nurses all patients on the ward, though is given special responsibility for some patients. Each patient is assigned one nurse in this position of responsibility. one of the doctors is attached to each 1 ward as an overall medical advisor.

Figure 8 Duffy et al [68] LHP Problem Statement

4.1.2 After Pre-Processing

A local hospital consists of many wards, each of which is assigned many patients. Each patient is assigned to one doctor, who has overall responsibility for the patients in his or her care. Other doctors are assigned on an advisory basis. Each patient is prescribed drugs by the doctor responsible for that patient. Each nurse is assigned to a ward and nurses all patients on the ward, though is given special responsibility for some patients. Each patient is assigned one nurse in this position of responsibility. One of the doctors is attached to each ward as an overall medical advisor.

4.1.3 Manual Object diagram of the LHP

Object model of Local Hospital Problem case study by Duffy et al [68] is given in Figure 9.

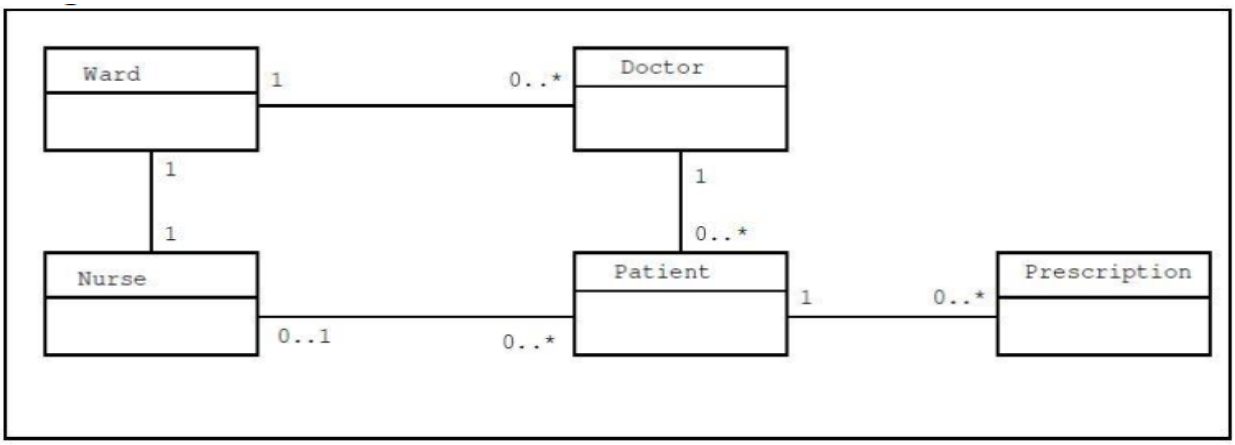

Figure 9 Object Model of Local Hospital Problem

4.1.4 SD-LINGO Screen Shot

Screenshot of AR2DT for LHP case study in context of class identification and initial attributes + relationships are given in Figure 10.

Figure 10 Screenshot of SD-LINGO for LHP Case Study

4.1.5 Class Diagram

Following class diagram is generated by SD-LINGO for the LHP case study automatically from initial plain text as shown Figure 11.

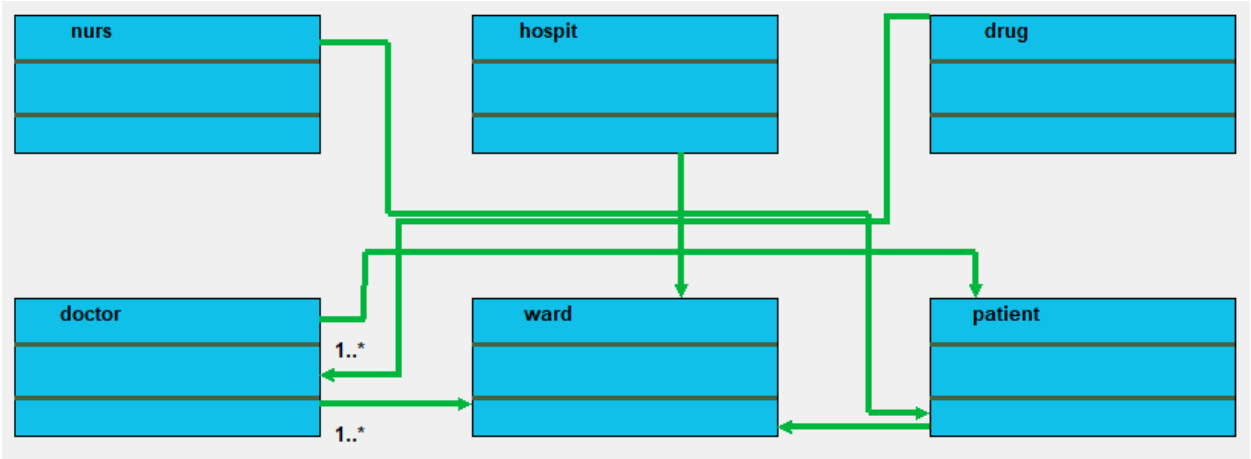

Figure 11 Class Diagram of LHP case study generated by SD-LINGO

4.1.6 Comparison of Generated Classes from Actual Model and SD-LINGO

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 11**.** SD-LINGO generated 4 correct classes, 1 incorrect class, 1 extra class and 1 missing class in comparison with the actual model where number of correct instances made by the system is represented by *'Ncorrect'*. Number of incorrect responses made by the system refers to the *'Nincorrect'*. Number of elements missed by the system but still extracted by the human experts are represented by *'Nmissing'*. Where number of extra elements retrieved by the system is said to be '*Nextra'.* Both number of *'Nincorrect' &* '*Nextra'* will be determined by the human experts. Precision, recall and over specification of the selected case study is given below Table 11**.**

Table 11 Comparison of GC/AM for LHP Case Study

Sr. No.	Actual Model Classes	Tool Generated Classes	Status Class
1	Patient	Patient	Correct
2	Ward	Ward	Correct
3	Doctor	Doctor	Correct
4	Nurse	Nurse	Correct
5	Prescription		Missing
6		Drug	Incorrect
7		Hospital	Extra

Correct=4, Incorrect=1, Extra=1, Missing=1

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision=4/ (4+1) =0.8=80%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing)}$

Recall= 4/ (4+1) =0.8=80%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing)}$

Overspecification = 1/ (4+1) = 20%

4.2 Bank Accounts Management System Case Study

4.2.1 Problem Statement

This system provides the basic services to manage bank accounts at a bank called OOBank. OOBank has many branches, each of which has an address and branch number. A client opens accounts at a branch. Each account is uniquely identified by an account number; it has a balance and a credit or overdraft limit. There are many types of accounts, including: a mortgage account (which has a property as collateral), a checking account, and a credit card account (which has an expiry date and can have secondary cards attached to it). It is possible to have a joint account (e.g. for a husband and wife). Each type of account has a particular interest rate, a monthly fee and a specific set of privileges (e.g. ability to write checks, insurance for purchases etc.). OOBank is divided into divisions and subdivisions (such as Planning, Investments and Consumer); the branches are considered subdivisions of the Consumer Division. Each division has a manager and a set of other employees. Each customer is assigned a particular employee as his or her 'personal banker.'

Figure 12 BAMS problem statement

4.2.2 After Pre-Processing

This system provides the basic services for manage OObank accounts. OOBank has many branches, each of which has branch address and branch number. clients open accounts at different branches. Each account is uniquely identified by an account number; it has a balance and a credit or overdraft limit. There are many account type , including: a mortgage account (which has a property as collateral), a checking account, and a credit card account (card has an expiry date and can have secondary card-attached). Account is possible to have a joint-account (e.g. for a husband and wife). Each type of account has a particular interest rate , a monthly fee and a specific set of account privilege (e.g. ability to write checks, insurance for purchases etc.). OOBank is divided into divisions and subdivisions (such division type as Planning, Investments and Consumer); the branches are considered subdivisions of the Consumer Division. Each divisions has a managers and a set of other divisions employees. clients are assigned a particular employee as his or her 'personal bankers?

4.2.3 Manual Object Diagram of BAMS

Object model of Bank Accounts Management System case human experts is given in Figure 13.

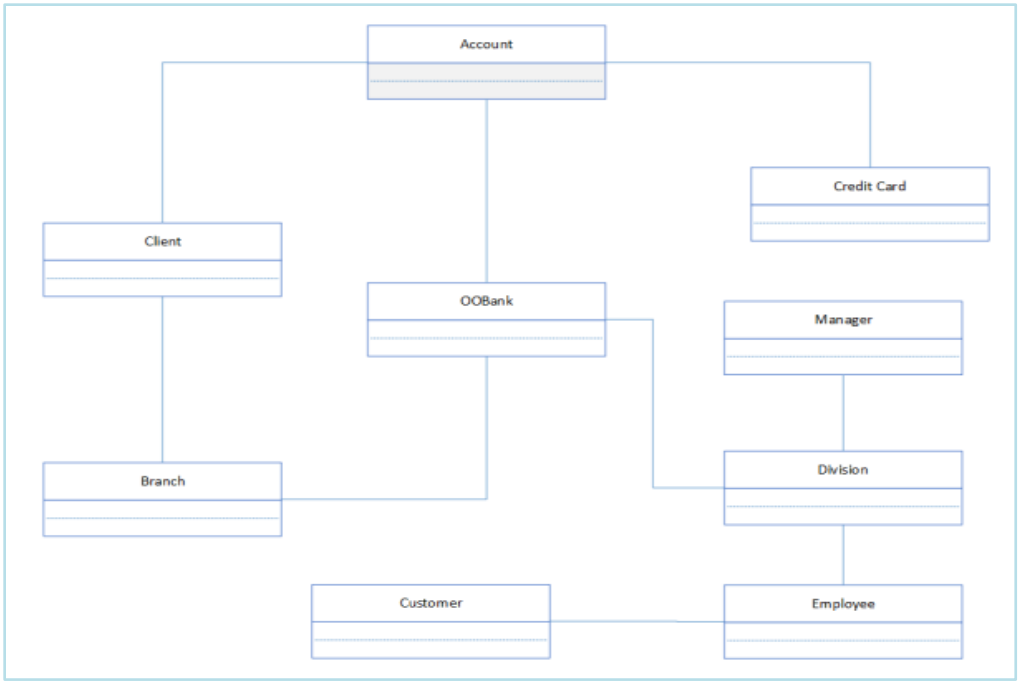

Figure 13 Object Model of Bank Accounts Management System

4.2.4 SD-LINGO Screen Shot

Screenshot of SD-LINGO for BAMS case study in the context of class identification and attributes + relationships are given Figure 14.

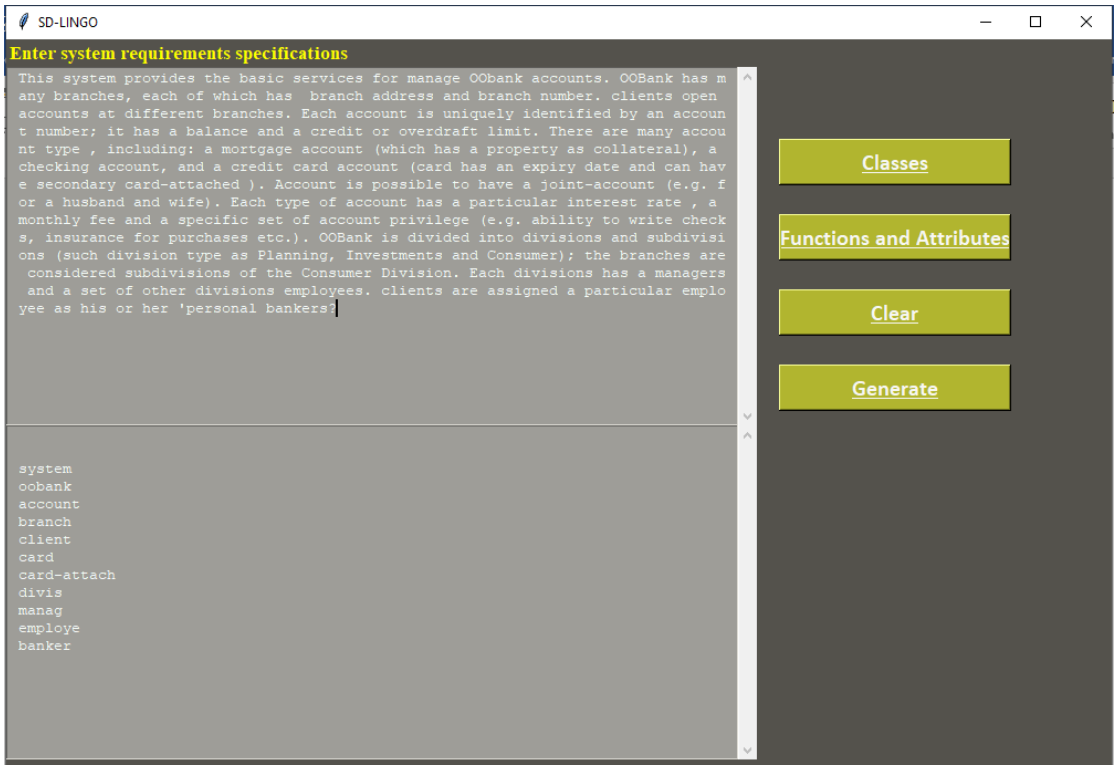

Figure 14 Screenshot of SD-LINGO for BAMS Case Study

4.2.5 Class Diagram

Following class diagram is generated by SD-LINGO for the BAMS case study automatically from initial plain text as shown in Figure 15.

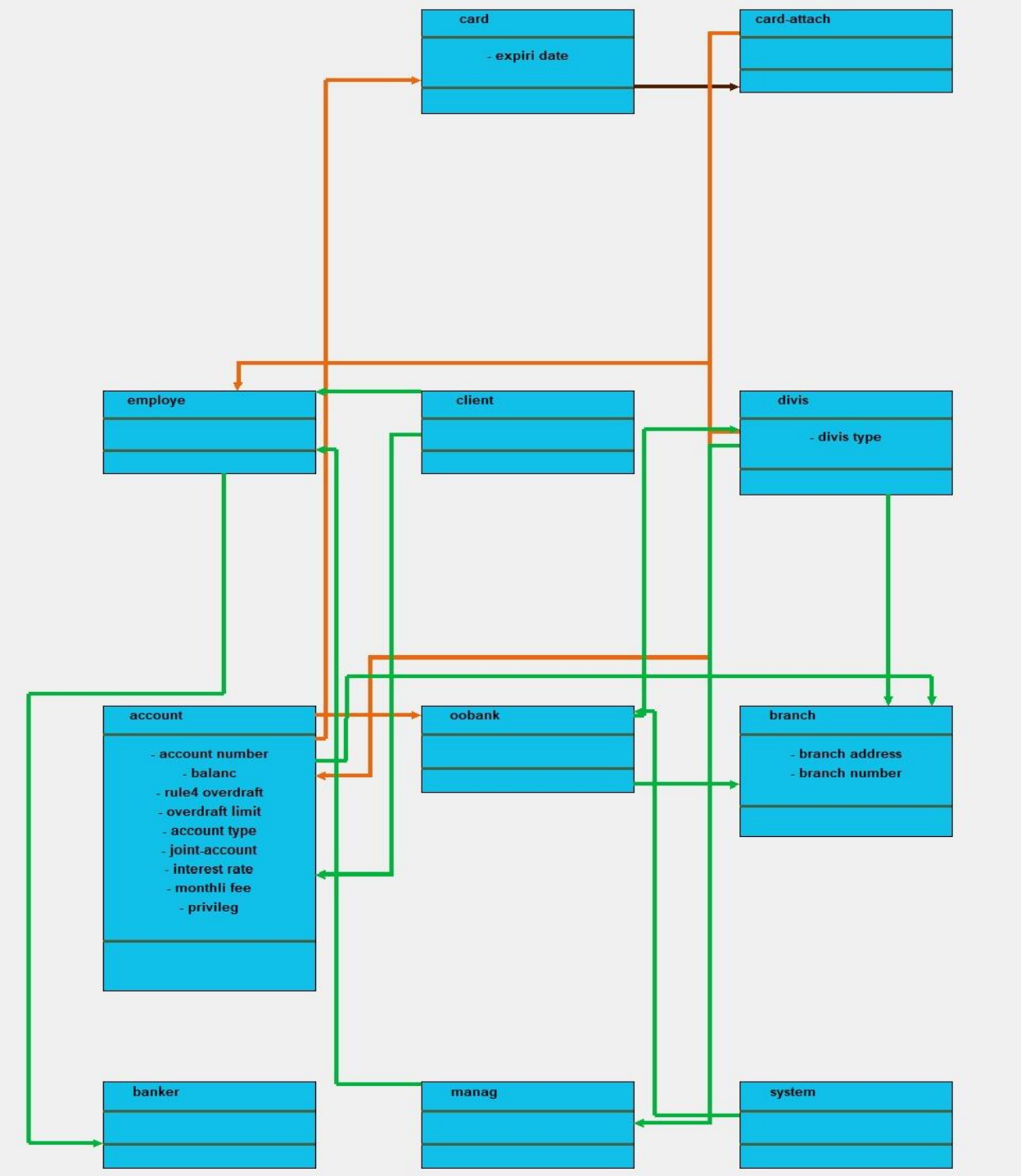

Figure 15 A Class Diagram of BAMS case study generated by SD-LINGO

4.2.6 Comparison of Generated Classes from Actual Model and SD-LINGO

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 12. SD-LINGO generated 8 correct classes, 1 incorrect class, 2 extra classes, and 1 missing class in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Both numbers of '$N_{incorrect}$' & 'N_{extra}' will be determined by the human experts. Precision, recall and over-specification of the selected case study is given below Table 12.

Table 12 Comparison of GC/AM for BAMS Case Study

Sr. No.	Actual Model Classes	Tool Generated Classes	Status
1	Account	Account	Correct
2	Branch	Branch	Correct
3	Credit card	Card	Correct
4	Customer		Missing
5	Manager	Manager	Correct
6	Division	Divis	Correct
7	OOBank	OOBank	Correct
8	Client	Client	Correct
9	Employee	Employee	Correct
10		Card Attach	Extra
11		Banker	Incorrect
12		System	Extra

Correct: 8, Incorrect: 1, Extra: 2, Missing: 1

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision=8/ (8+1) =0.888=88.8%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)

Recall= 8/ (8+1) =0.888=88.8%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)

Overspecification = 2/ (8+1) = 0.22=22.2%

4.3 Automatic Teller Machine (ATM) Case study

Rumbaugh et al [69] initially analyzed the ATM case study using their OMT methodology, has been presented here. We take the same problem statement in this section and present the analysis results. After which we present our system model and also present the comparison with Rumbaugh et al [69] model.

4.3.1 The Problem Statement

Automatic teller machine (ATM) problem statement as presented in the book [69] is shown in Figure 16 as follow:

> Banks provide computers to their cashiers. Each bank has bank-name bank-id and bank-city name. Users have accounts in banks, cashiers maintain bank accounts. The cashier should be ready to open accounts of users. user also has the user's name, id, account title, and bank name. Users should be ready to withdrawal money from accounts. Each account has an account title account name and account id. cashiers have cashier name, id, city, and cashier city to make sure record is entered by the correct person and there is no chance of misuse of the system. Each cashier has access to bank computers. Bank computers connect with main computers or supercomputers. Automatic Teller Machine (ATMs) connect with main-computers account withdrawal money from ATMs. Users' withdrawal money from ATMs.

Figure 16 ATM Problem Statement

4.3.2 After Pre-Processing

> Banks provide computers to their cashiers. Each bank has bank-name bank-id and bank-city name. Users have accounts in banks. cashiers maintain bank accounts. The cashier should be ready to open accounts of users. user also has the user's name, id, account title, and bank name. Users should be ready to withdrawal money from accounts. Each account has an account title account name and account id. cashiers have cashier name, id, city, and cashier city to make sure record is entered by the correct person and there is no chance of misuse of the system. Each cashier has access to bank computers. Bank computers connect with main computers or supercomputers. ATMs connect with main-computers account withdrawal money from ATMs. Users' withdrawal money from ATMs.

4.3.3 Object Model of Rumbaugh et al [69]

Object modelling techniques also known as OMT methodology are used by Rumbaugh et al [69] to build the object model of the automatic teller machine (ATM) case study. Author considered all the nouns and extracted list of candidate classes from the case study problem statement. The list of candidate classes is 7: Computer, Bank, Account, ATM, Cashier, Main computer, and customer. After identifying such list of classes, author [30] considered all the verb phrases and build an object model that basically presents the classes and their relationship as shown in figure 16.

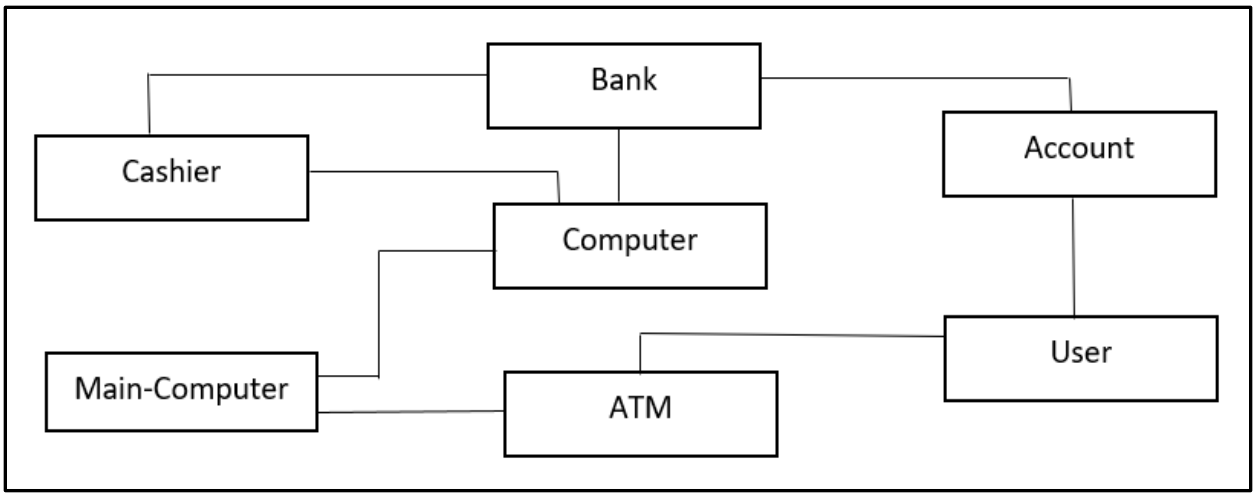

Figure 17 Object Model of ATM

4.3.4 SD-LINGO Screenshot

Screenshot of SD-LINGO for ATM case study in the context of class identification and initial attributes + relationships are given in Figure 18.

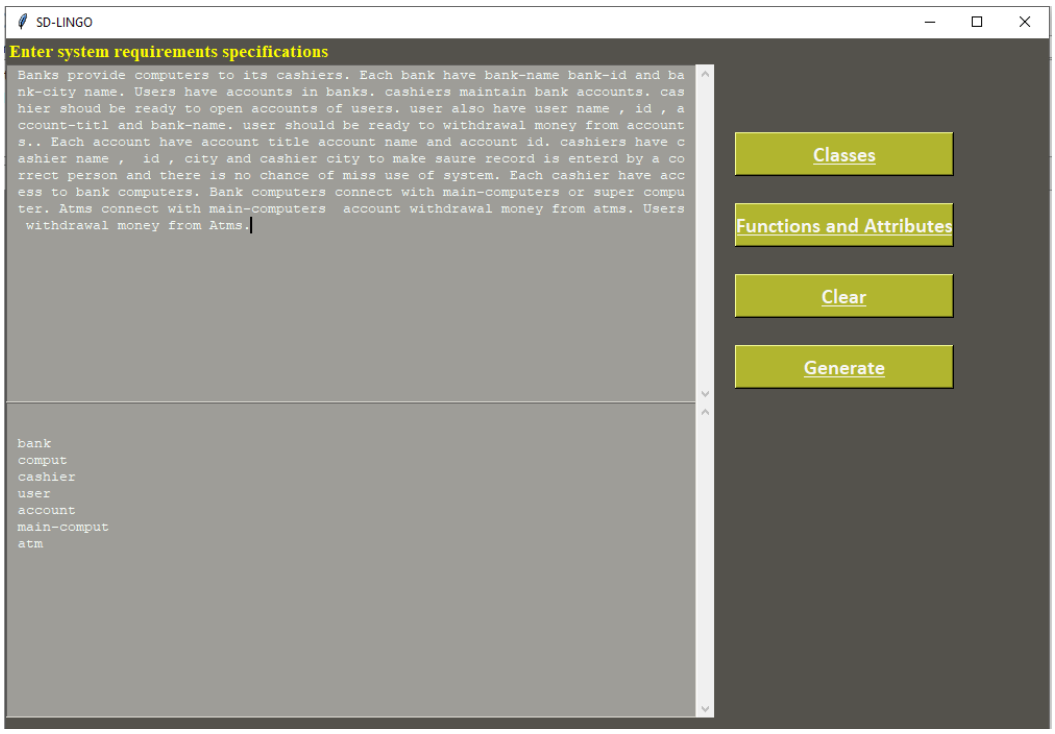

Figure 18 Screenshot of SD-LINGO for ATM Case Study

4.3.5 Analysis by SD-LINGO

From automatic teller machine problem statement, SD-LINGO analyzed the list of classes and relationships as shown below. Also, give a comparison with Rumbaugh et al [69] model in this section.

Class Diagram

From automatic teller machine problem statement, SD-LINGO analyzed a set of classes for ATM problem statement and produced 7 classes by considering all types of nouns i.e., NN, NNS, NNP and NNPS and by deleting the redundant classes and compound nouns. Finally, we got 7 classes attributes, and relationships as shown in Figure 19**.**

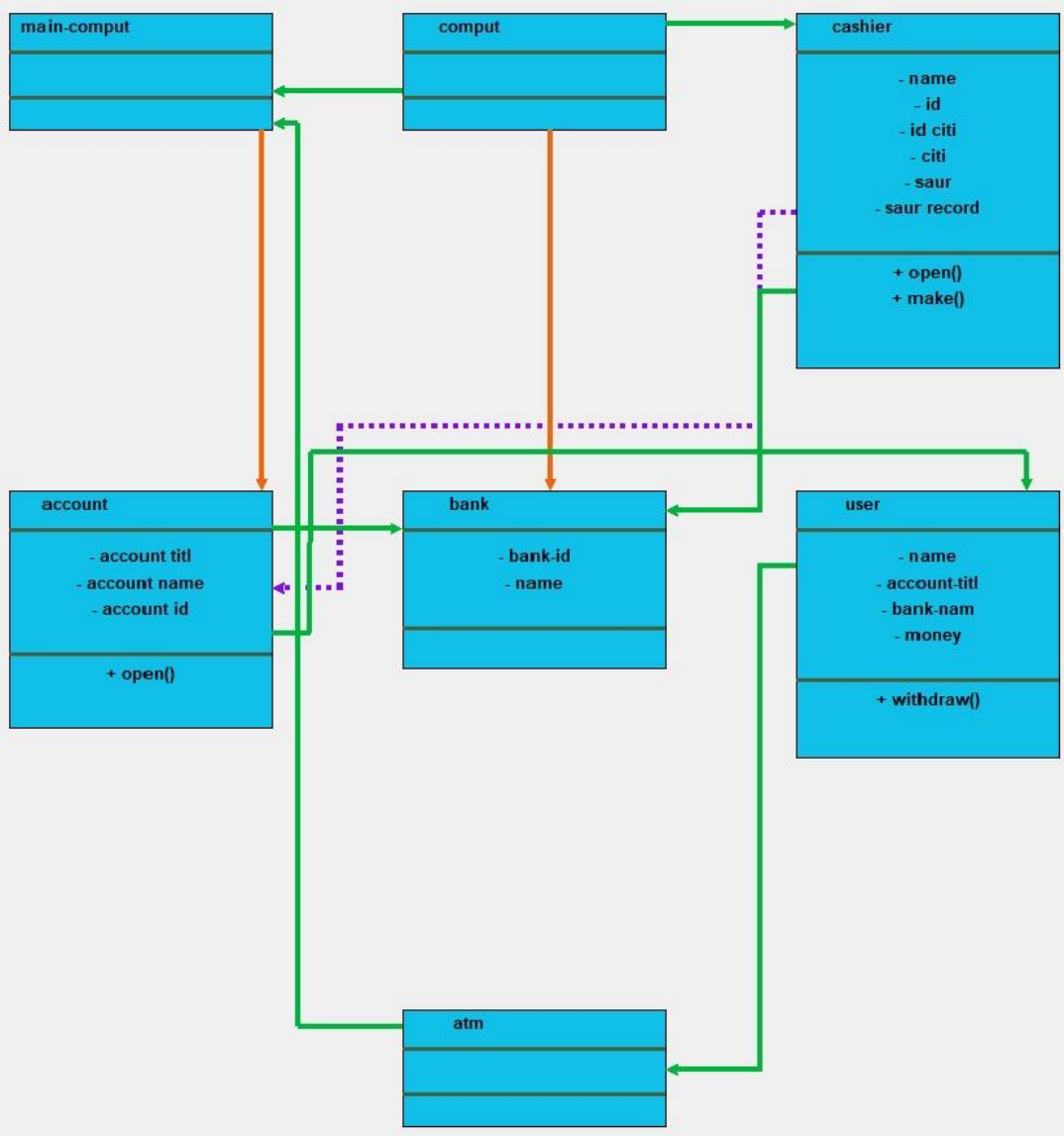

Figure 19 A Class Diagram of ATM case study generated by SD-LINGO

4.3.6 Comparison of Generated Classes by SD-LINGO and Actual Model

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table *13*. SD-LINGO generated 7 correct classes, 0 incorrect class, 0 extra classes, and 0 missing class in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the

'$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Both numbers of '$N_{incorrect}$' & 'N_{extra}' will be determined by the human experts. Precision, recall and over-specification of the selected case study is given below Table 13.

Table 13 Comparison of GC/AM for ATM Case Study

Sr. No.	Actual Model Classes	Tool Generated Classes	Status
1	Bank	Bank	Correct
2	Cashier	Cashier	Correct
3	Account	Account	Correct
4	User	User	Correct
5	Main-Computer	Main-computer	Correct
6	Computer	Computer	Correct
7	ATM	ATM	Correct

Correct:7, Incorrect:0, Missing:0, Extra:0

Precision = $N_{correct} / (N_{correct} + N_{incorrect})$

Precision=7/ (7+0) =1=100%

Recall = $N_{correct} / (N_{correct}+N_{missing})$

Recall= 7/ (7+0) =1=100%

Overspecification = $N_{extra}/ (N_{correct}+N_{missing})$

Overspecification = 0/ (7+0) = 0=0%

4.4 Library Management System

4.4.1 The Problem Statements

Library Management System problem statement as presented in the book [69] is shown in Figure 20 as follow.

Library members should be ready to search books by their title, author, subject category also by publication date. Each book will have a singular number and other details including a rack number which can help to physically locate the book. Each member will have a name, id, and fine. There might be quite one copy of a book, and library members should be ready to check out and reserve any copy. We'll call each copy of a book, a book item. member receives notifications from the system. The system should be ready to retrieve information like who took a specific book or what are the books checked out by a selected library member. There should be a maximum limit (5) on what percentage of books can members check out. There should be a maximum limit (10) on what percentage members can keep books maximum days. The system should be ready to collect fines for books returned after maturity. Members should be ready to reserve books that aren't currently available. The system should be ready to send notifications for books that are reserved and become available, also as when isn't returned within the maturity. Each book and member's card will have a singular barcode. The Card is a child of member class and has access data of member class. The system is going to be ready to read barcodes from books and members' library cards.

Figure 20 Library Management System Problem Statements

4.4.2 After Pre-Processing

Library members should be ready to search books by their title, author, subject category also by the publication date. Each book will have a singular number and other details including a rack number which can help to physically locate the book. Each member will have name, id and fine. There might be quite one copy of a book, and library members should be ready to check-out and reserve any copy. we'll call each copy of a book, a book item. member receive notifications from system. The system should be ready to retrieve information like who took a specific book or what are the books checked-out by a selected library member. There should be a maximum limit (5) on what percentage books can members check-out. There should be a maximum limit (10) on what percentage members can keep books maximum days. The system should be ready to collect fines for books returned after the maturity. Members should be ready to reserve books that aren't currently available. The system should be ready to send notifications for books that reserved and become available, also as when isn't returned within the maturity. Each book and members cards will have a singular barcode. The Card is a child of member class and have access data of member class. The system are going to be ready to read barcodes from books and members' library cards

4.4.3 Object Diagram of Library Management System

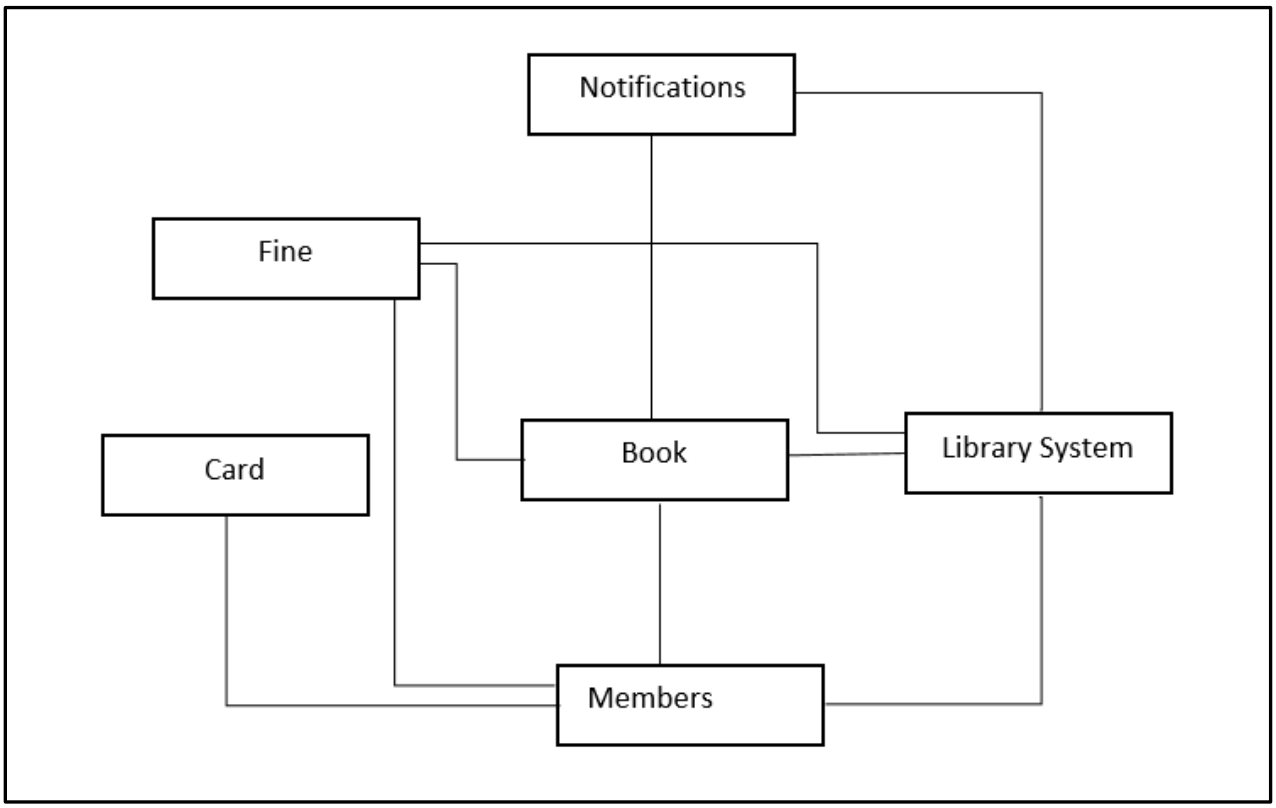

Figure 21 Object diagram of library management system

4.4.4 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Library Management case study in the context of class identification and attributes + relationships are given Figure 22.

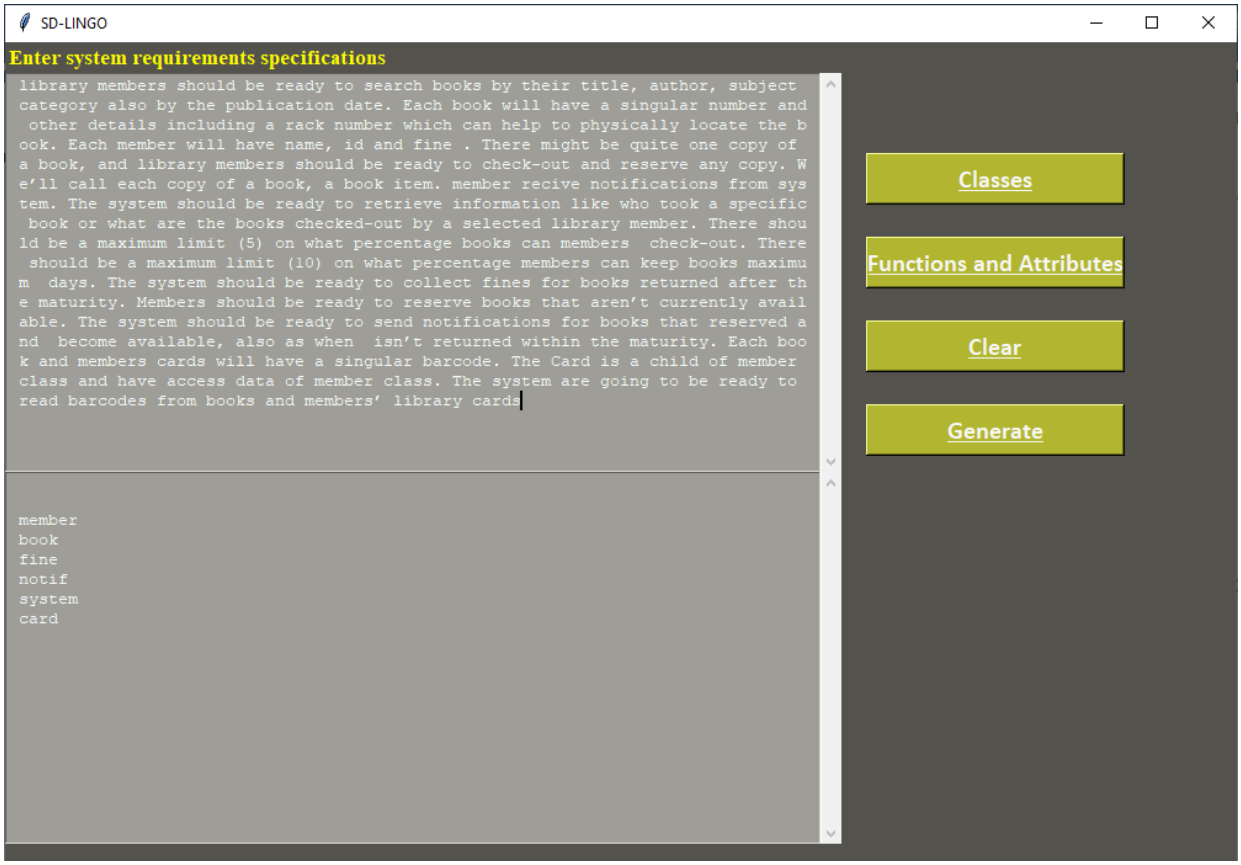

Figure 22 Screenshot of SD-LINGO for Library management system case study

4.4.5 Class Diagram

Following class diagram is generated by SD-LINGO for the Library Management System case study automatically from initial plain text as shown in Figure 23.

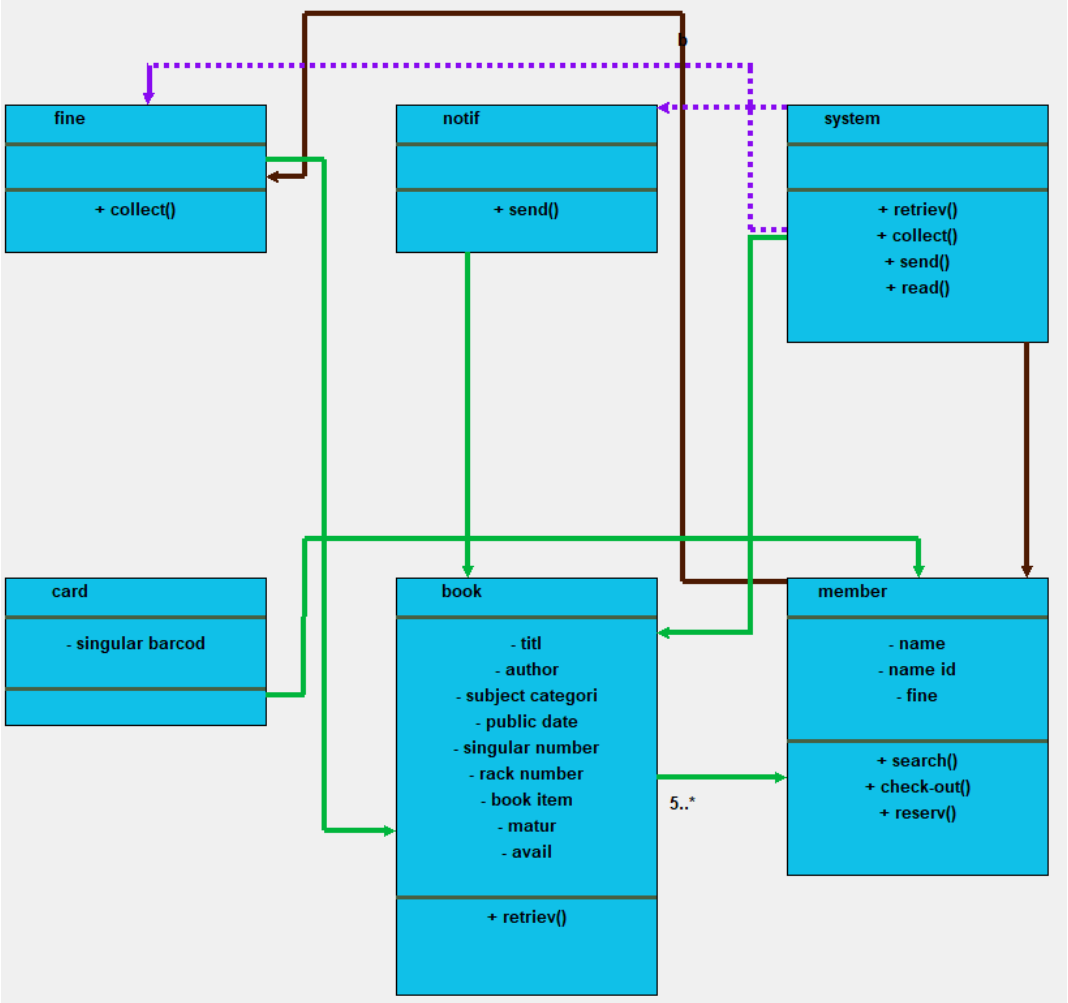

Figure 23 Class diagram of library management system

4.4.6 Comparison of Generated Classes from Actual Model and SD-LINGO

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 14. SD-LINGO generated 6 correct classes, 0 incorrect class, 0 extra classes, and 0 missing class in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Both numbers of '$N_{incorrect}$' & 'N_{extra}' will be determined by the human experts. Precision, recall and over-specification of the selected case study is given below Table 14.

Table 14 Comparison of GC/AM for Library Management system Case Study

Sr. No.	Actual Model Classes	Tool Generated	Status
1	Member	Member	Correct
2	Book	Book	Correct
3	Notification	Notif	Correct
4	Fine	Fine	Correct
5	System	System	Correct
6	Card	Card	Correct

Correct: 6, Incorrect: 0, Missing: 0, Extra: 0

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision=6/ (6+0) =1=100%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)

Recall= 6/ (6+0) =1=100%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)

Overspecification = 0/ (6+0) = 0=0%

4.5 Journal Registration Problem Case study

4.5.1 The Problem Statement

The personnel department of a large research institute is responsible for the purchase and dissemination of journals to readers in other departments in the organization. Readers may be interested in certain specific topics relating to their research interests, while it is also possible to be placed on a circulation list. Usually, readers get access to an issue of journals for a fixed journal period, typically two weeks. It is possible to have access to issues for a longer period, but permission must be granted from the personnel department for Journals. Journals appear on a regular basis and each journal contains information on the publisher, language, and frequency of publication. The system should keep readers informed of the topics that are of interest to them and which appear in the different journals. Furthermore, it should be possible for readers to find articles that deal with topics that they are interested in.

Figure 24 Journal Registration Problem (JRP) Case Study

4.5.2 After Pre-Processing

The personnel department of a large research institute is responsible for the purchase and dissemination of journals to readers in other departments in the organization. Readers may be interested in certain specific topics relating to their research interests, while it is also possible to be placed on a circulation list. Usually, readers get access to an issue of journals for a fixed journal-period, typically two weeks. It is possible to have access to issues for a longer period, but permission must be granted from the personnel department for Journals. Journals appear on a regular basis and each journal contains information on the publisher, language, and frequency of publication. The system should keep readers informed of the topics that are of interest to them and which appear in the different journals. Furthermore, it should be possible for readers to find articles which deal with topics that they are interested in.

4.5.3 Object Model of Journal Registration Problem

Object model of Journal Registration Problem(JRP) case study by Duffy et al [68] is given in Figure 25.

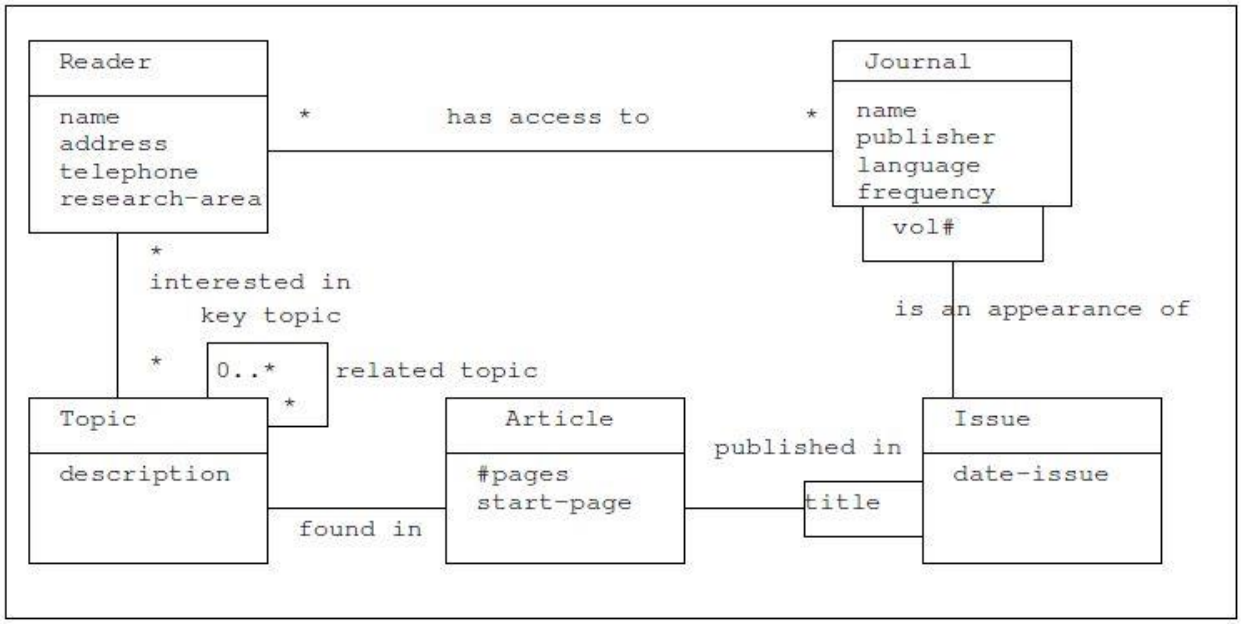

Figure 25 Object model of JRP case study

4.5.4 SD-LINGO Screen Shot

Screenshot of SD-LINGO for JRP case study in context of class identification and attributes + relationships are given in Figure 26.

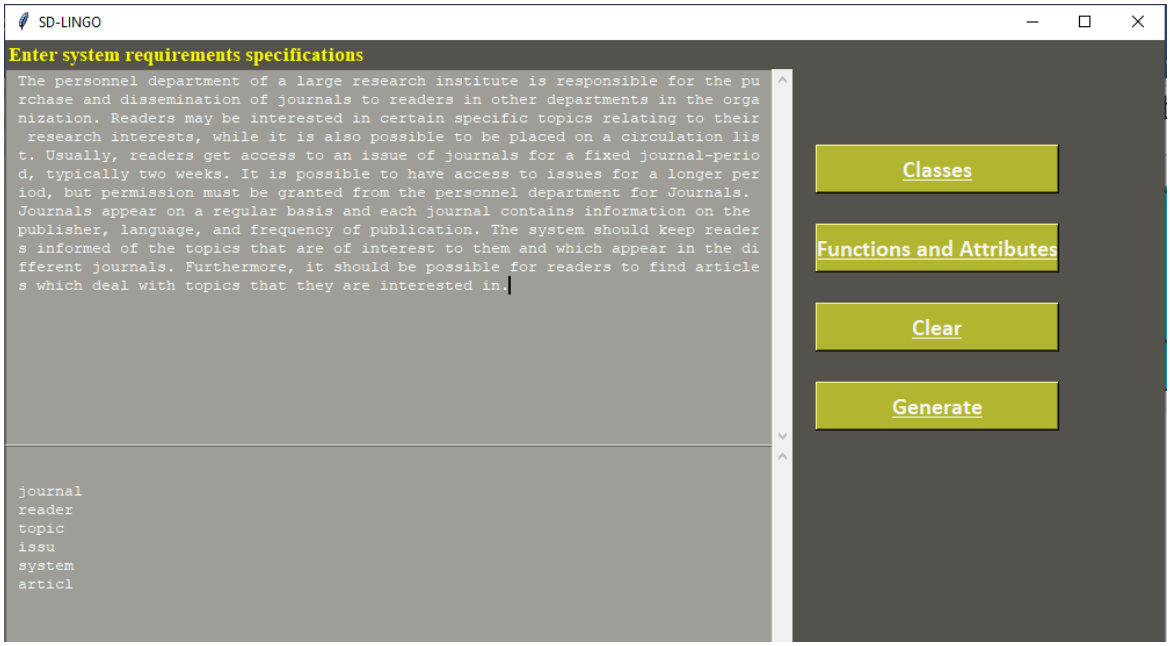

Figure 26 Screenshot of SD-LINGO for JRP

4.5.5 Class Diagram

Following class diagram is generated by SD-LINGO for the journal registration problem (JRP) case study automatically from initial plain text as shown in Figure 27.

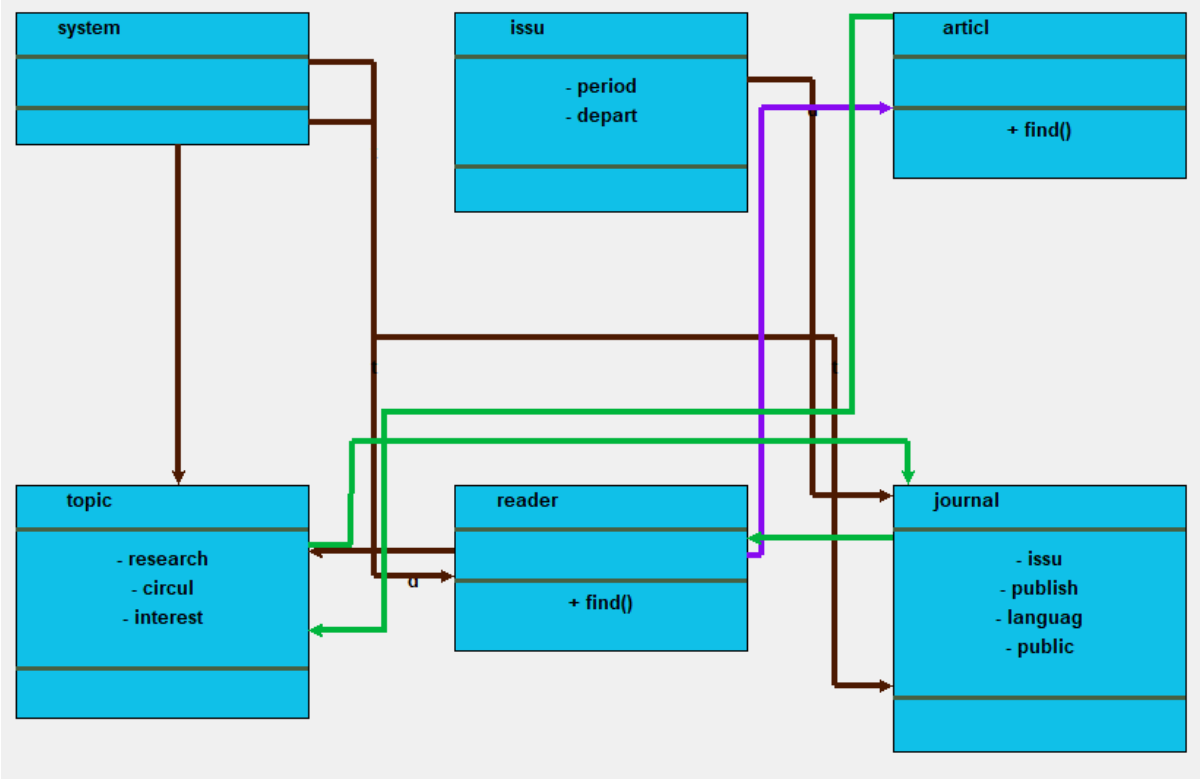

Figure 27 Class diagram of JRP case study

4.5.6 Comparison of Generated Classes from Actual Model and SD-LINGO

Comparison of classes generated by SD-LINGO with the actual manual model is given in *Table* 15. SD-LINGO generated 5 correct classes, 0 incorrect class, 1 extra classes, and 0 missing class in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Both numbers of '$N_{incorrect}$' & 'N_{extra}' will be determined by the human experts. Precision, recall and over-specification of the selected case study is given below Table 15.

Table 15 Comparison of GC/AM for JRP Case Study

Sr. No.	Actual Model Classes	Tool Generated Classes	Status
1	Journal	Journal	Correct
2	Reader	Reader	Correct
3	Topic	Topic	Correct
4	Issue	Issue	Correct
5	Article	Article	Correct
6		System	Extra

Correct: 5, Incorrect: 0, Missing: 0, Extra: 1

Precision = $N_{correct} / (N_{correct} + N_{incorrect})$

Precision=5/ (5+0) =1=100%

Recall = $N_{correct} / (N_{correct}+N_{missing})$

Recall= 5/ (5+0) =1=100%

Overspecification = $N_{extra}/ (N_{correct}+N_{missing})$

Overspecification = 1/ (6+0) = 0.16=17%

4.6 Course Registration Case Study

4.6.1 The problem Statement

The problem statement of course registration case study shown in the Figure 28 from IBM Corp [70].

> At the beginning of each semester, students may request a course catalogue containing a list of course offerings for the semester. Information about each course, such as professor, department, and prerequisites, will be included to help students make informed decisions.
>
> The new system will allow students to select four course offerings for the coming semester. In addition, each student will indicate two alternative choices in case the student cannot be assigned to a primary selection. Course offerings will have a maximum of ten students and a minimum of three students. A course offering with fewer than three students will be canceled. For each semester, there is a period of time that students can change their schedule. Students must be able to access the system during this time to add or drop courses. Once the registration process is completed for a student, the registration system sends information to the billing system so the student can be billed for the semester. If a course fills up during the actual registration process, the student must be notified of the change before submitting the schedule for processing.
>
> At the end of the semester, the student will be able to access the system to view an electronic report card. Since student grades are sensitive information, the system must employ extra security measures to prevent unauthorized access
>
> Professors must be able to access the on-line system to indicate which courses they will be teaching. They will also need to see which students signed up for their course offerings. In addition, the professors will be able to record the grades for the students in each class.

Figure 28 Course Registration Problem Statements

4.6.2 After Pre-Processing

At the beginnings of semesters, students may request a course catalogue containing a list of course offerings for the semester. Information about courses such as course professor, course department, and course prerequisite. will be included to help students make informed decisions. The new system will allow students to select 4 courses offering for the coming semester. In addition, each student will indicate two alternative choices in case the student cannot be assigned to a primary selection. Course offerings will have a maximum of ten students and a minimum of three students. A course offering with fewer than three students will be canceled. For each semester, there is a period of time that students can change their schedule. Students must be able to access the system during this time to add or drop courses. Once the registration process is completed for a student, the registration system sends information to the billing-systems so the student can be billed for the semester. If a course fills up during the actual registration process, the student must be notified of the change before submitting the schedule for processing. At the ends of the semesters, the student will be able to access the system to view an electronic report-cards . Since student-grades are sensitive information, the system must employ extra security measures to prevent unauthorized access. Professors must be able to access the on-line system to indicate which courses they will be teaching. They will also need to see which students signed up for their course offerings. In addition, the professors will be able to record the grades for the students in each class.

4.6.3 Manual Object Diagram of Course Registration Case study

Object model of Course Registration case study by IBM Corp [70] is given in Figure 29.

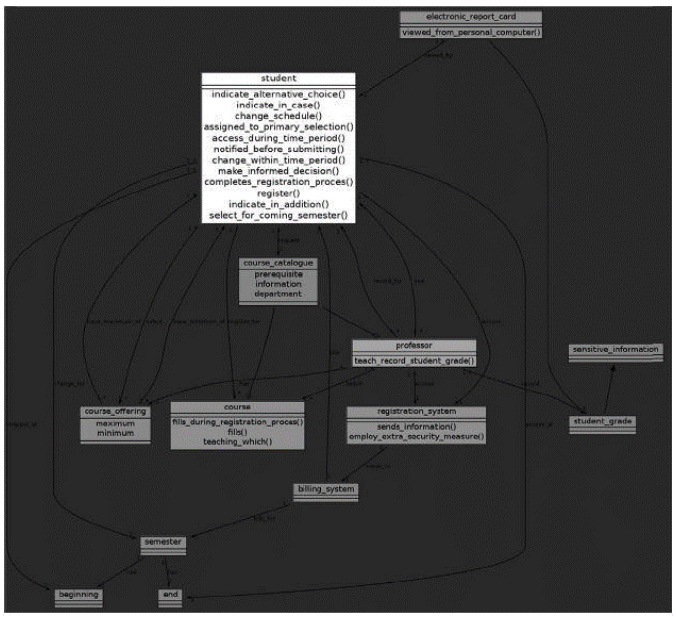

Figure 29 Object diagram of course registration case study

4.6.4 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Course Registration case study in context of class identification and initial attributes + relationships are given in Figure 30.

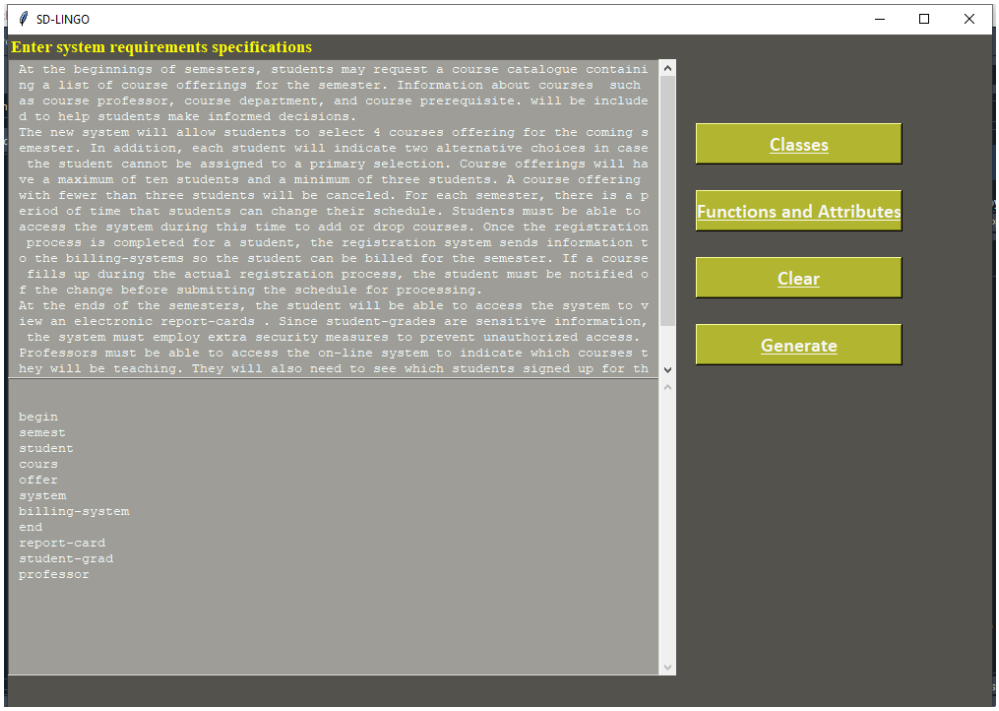

Figure 30 Screenshot of SD-LINGO for course registration

4.6.5 Class Diagram

Following class diagram is generated by SD-LINGO for the Course Registration case study automatically from initial plain text as shown in Figure 31**.**

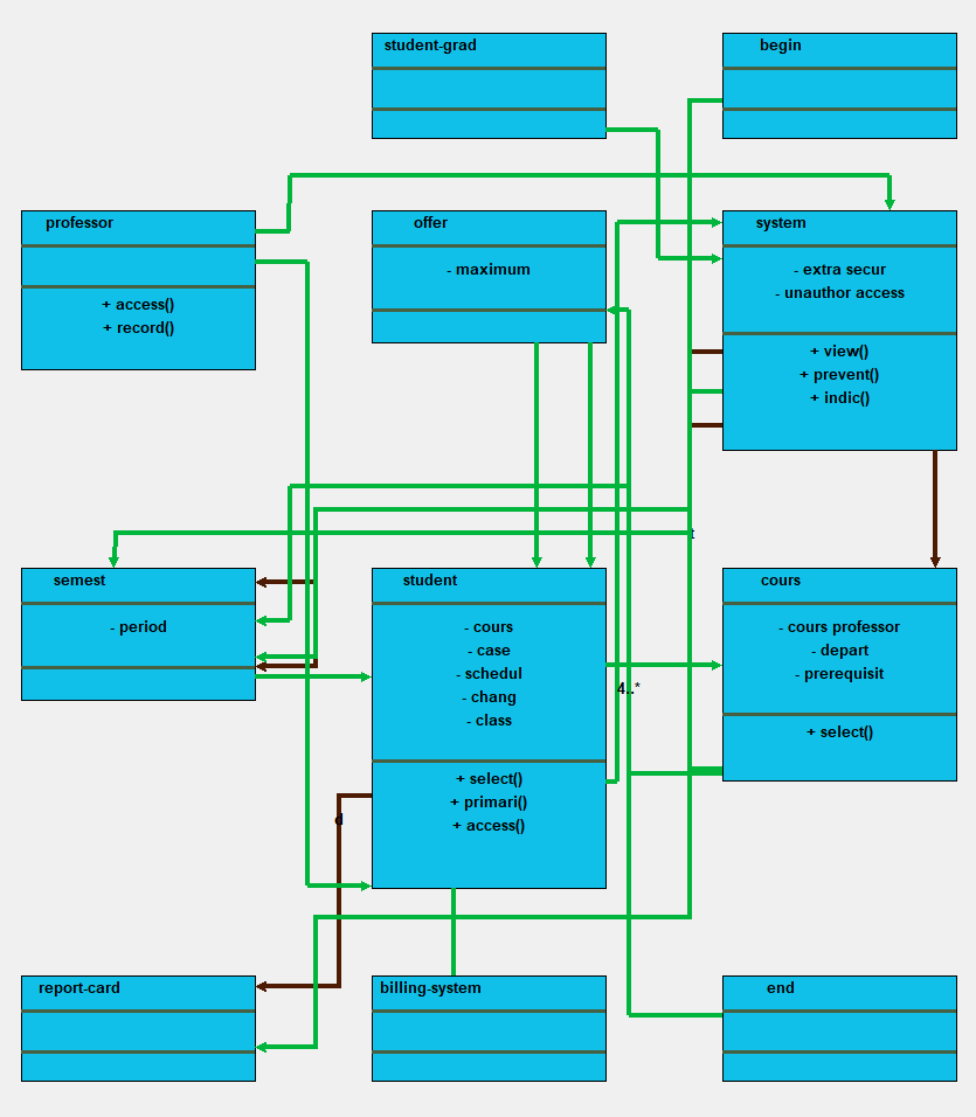

Figure 31 Class diagram of course registration case study

4.6.6 Comparison of Generated Classes from Actual Model and SD-LINGO

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table *16*. SD-LINGO generated 10 correct classes, 0 incorrect class, 1 extra classes, and 3 missing class in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Both numbers of '$N_{incorrect}$' & 'N_{extra}' will be determined by the human experts. Precision, recall and over-specification of the selected case study is given below Table 16.

Table 16 Comparison of GC/AM for course registration case study

Sr. No.	Actual Model Classes	Tool Generated Classes	Status
1	Student	Student	Correct
2	Course catalog		Missing
3	Professor	Professor	Correct
4	Course Offering	Offer	Correct
5	Course	Course	Correct
6	Registration system		Missing
7	Student grade	Student-grade	Correct
8	Billing system	Billing-system	Correct
9	Semester	Semester	Correct
10	Electronic report card	Report card	Correct
11	Sensitive information		Missing
12	Begin	Begin	Correct
13	End	End	Correct
14		System	Extra

Correct: 11, Incorrect: 0, Missing: 3, Extra: 1
Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)
Precision=11/ (11+0) =1=100%
Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing)}$
Recall= 11/ (11+3) =0.785=79%
Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing)}$
Overspecification = 1/ (11+3) = 0.071=7%

4.7 Relationships of Single Line Requirements

We used single-line requirements from the software engineering books for the validation of relationships.

4.7.1 Inheritance/Generalization

4.7.1.1 Requirement 01

Dogs are kind of pets. Cats are kind of pets.

4.7.1.1.1 Class Model of Requirement 01

Class model of Requirement 01 by R Yilmaz et al [71] is given Figure *32*.

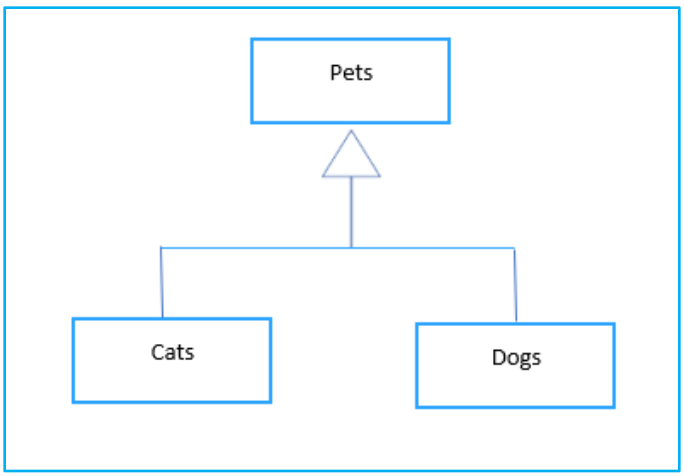

Figure 32 UML classes and inheritance

4.7.1.1.2 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Requirement#01 in context of class identification and relationships are given in Figure 33.

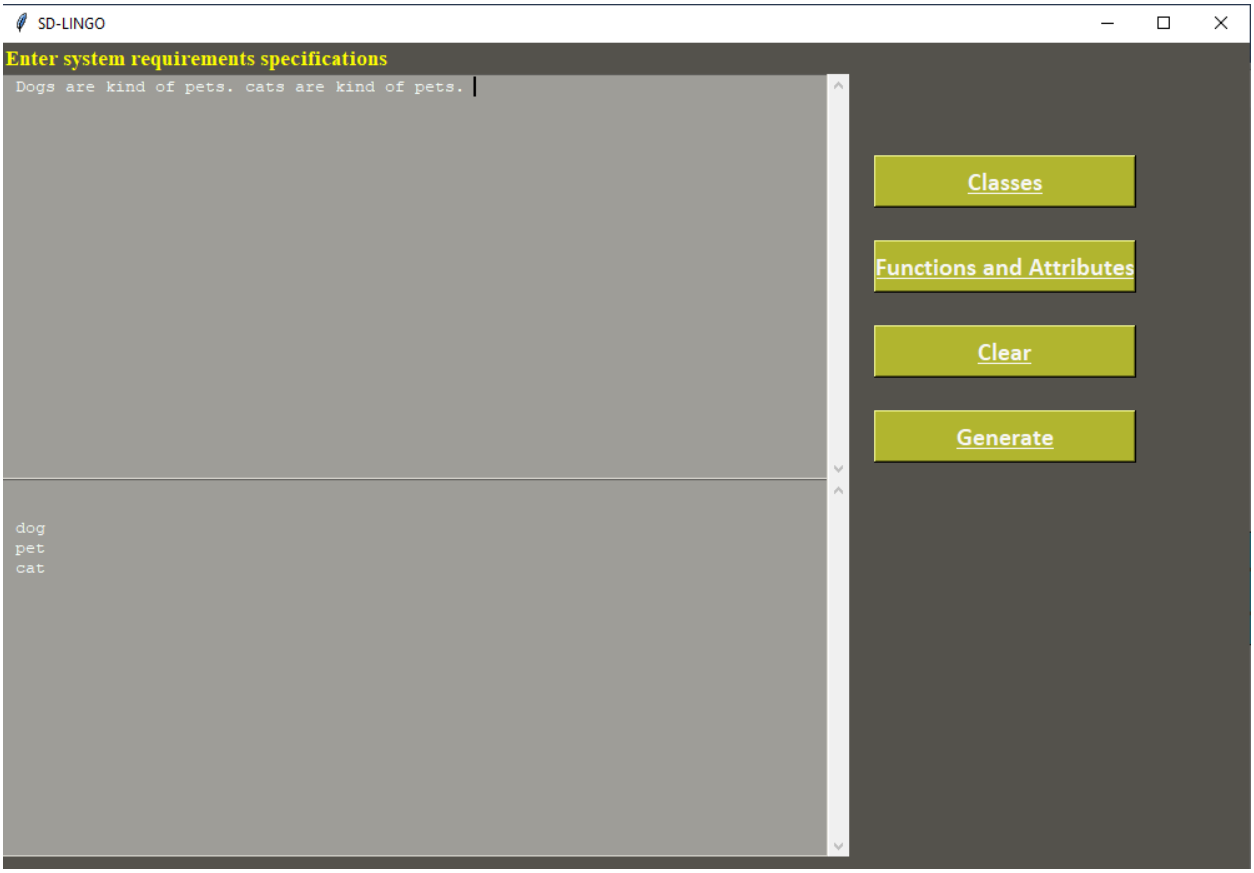

Figure 33 SD-LINGO tool screenshot req#01 for inheritance/Generalization

4.7.1.1.3 Class Diagram

Following class diagram is generated by SD-LINGO for Requirement#01 automatically from initial plain text as shown in Figure *34*.

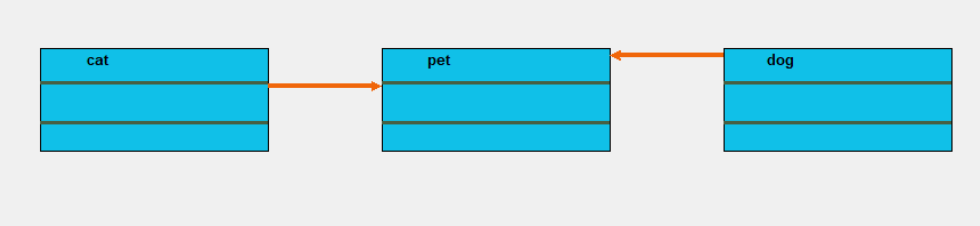

Figure 34 Class diagram of req#01 for inheritance/generalization

4.7.1.1.4 Comparison of Actual Model and SD-LINGO Generated Class

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 17. SD-LINGO generated 5 correct classes and relationships, 0 incorrect class, 0 extra classes, 0 missing class, in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Precision, recall and over-specification of the selected case study is given below Table 17.

Table 17 Comparison of GC/AM for Requirement#01

Sr. No.	Actual Model	Tool Generated	Status
1	Pet	Pet	Correct
2	Cat	Cat	Correct
3	Dog	Dog	Correct
4	Inherited relationship between cat and pet	Inherited relationship between cat and pet	Correct
5	Inherited relationship between dog and pet	Inherited relationship between dog and pet	Correct

Correct: 5, Incorrect: 0, Missing: 0, Extra: 0
Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)
Precision=5/ (5+0) =1=100%
Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)
Recall= 5/ (5+0) =1=100%
Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)
Overspecification = 0/ (5+0) = 0=0%

4.7.1.2 Requirement 02:

We generalized practioners as practioners doctors and nurses as a nurses doctors.

4.7.1.2.1 Class Model of Requirement 02

Class model of Requirement 02 by Sommerville [1] is given Figure *35*.

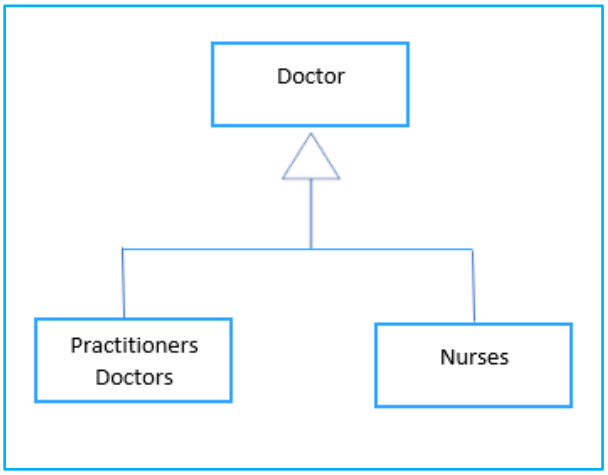

Figure 35 UML classes and generalization

4.7.1.2.2 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Requirement#02 in context of class identification and relationships are given in Figure 36.

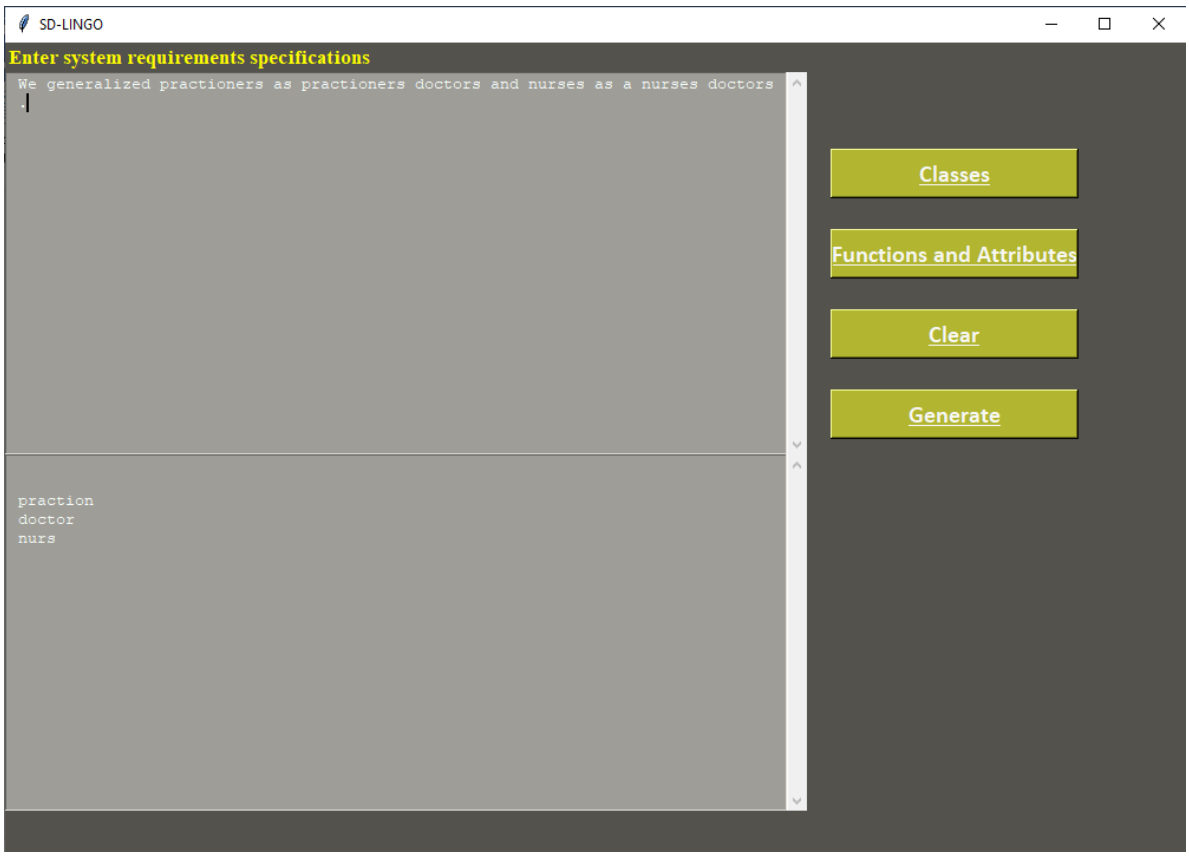

Figure 36 SD-Lingo tool screenshot of req#02 for generalization/inheritance

4.7.1.2.3 Class Diagram

Following class diagram is generated by SD-LINGO for Requirement 02 automatically from initial plain text as shown in Figure *37*.

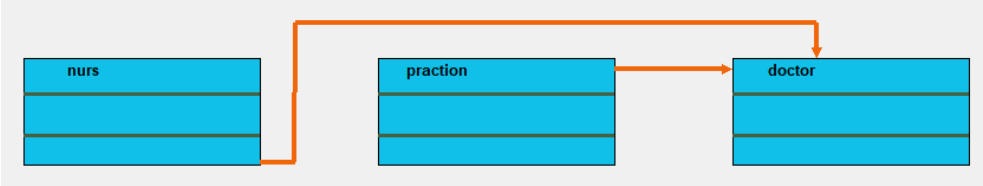

Figure 37 Class diagram of req#02 for generalization/inheritance

4.7.1.2.4 Comparison of Actual Model and SD-LINGO Generated Class

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 18. SD-LINGO generated 5 correct classes and relationships, 0 incorrect class, 0 extra classes, 0 missing class, in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Precision, recall and over-specification of the selected case study is given below Table 18.

Table 18 Comparison of GC/AM for Requirement#02

Sr. No.	Actual Model	Tool Generated	Status
1	Doctor	Doctor	Correct
2	Practitioners Doctors	Practitioners Doctors	Correct
3	Nurses	Nurses	Correct
4	Generalized relationship between practitioner doctors and doctors	Generalized relationship between practitioner doctors and doctors	Correct
5	Generalized relationship between nurses and doctors	Generalized relationship between nurses and doctors	Correct

Correct: 5, Incorrect: 0, Missing: 0, Extra: 0

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision=5/ (5+0) =1=100%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)

Recall= 5/ (5+0) =1=100%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)

Overspecification = 0/ (5+0) = 0=0%

4.7.2 Realization

4.7.2.1 Requirement 03:

Printers implement contract operations and interface printer-setups are used for contract operations by printers.

4.7.2.1.1 Class Model of Requirement 03

Class model of Requirement 03 by Sommerville [1] is given Figure *38*.

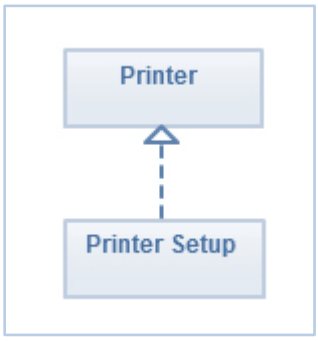

Figure 38 UML classes and realization

4.7.2.1.2 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Requirement 03 in context of class identification and relationships are given in Figure 39.

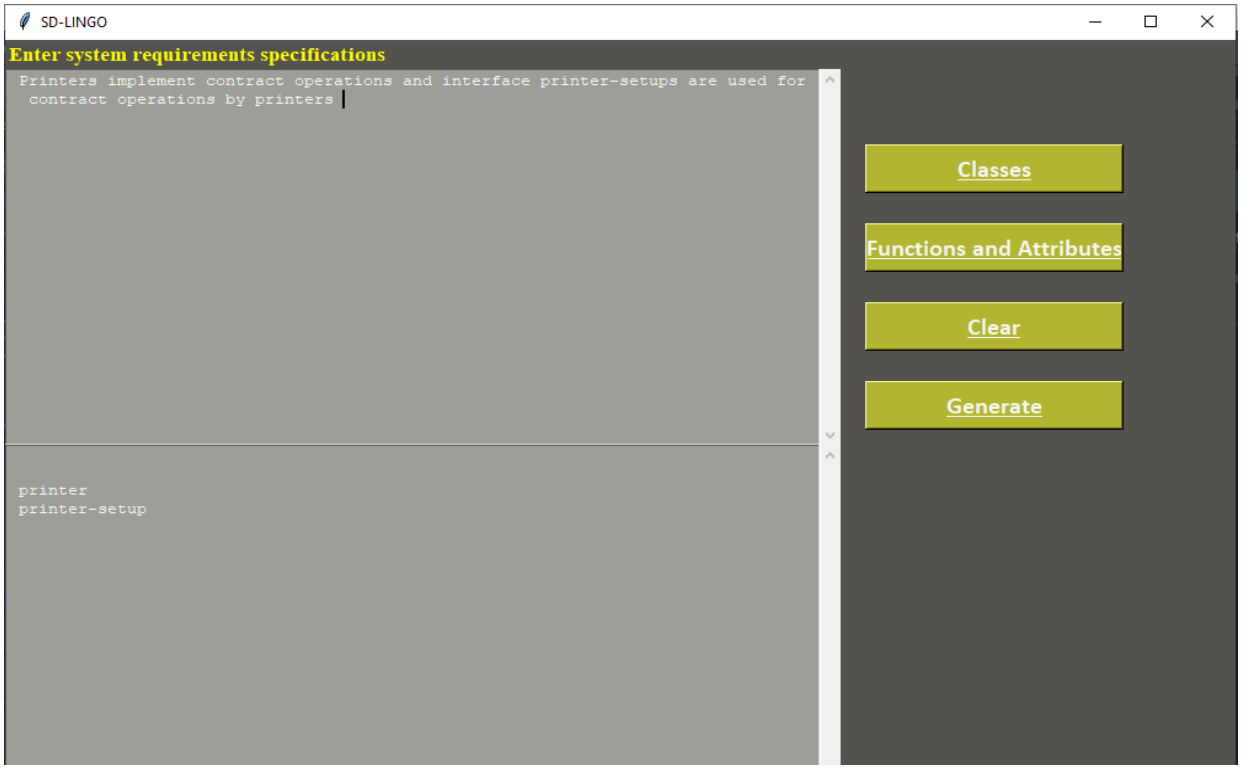

Figure 39 SD-Lingo tool screenshot for realization

4.7.2.1.3 Class Diagram

Following class diagram is generated by SD-LINGO for Requirement 03 automatically from initial plain text as shown in Figure *40*.

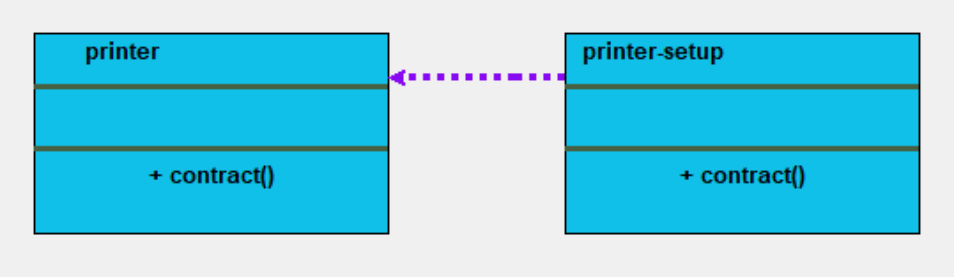

Figure 40 Classes show the realization

4.7.2.1.4 Comparison of Actual Model and SD-LINGO Generated Class

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 19. SD-LINGO generated 3 correct classes and relationships, 0 incorrect class, 0 extra classes, 0 missing class, in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Precision, recall and over-specification of the selected case study is given below Table 19.

Table 19 Comparison of GC/AM for Requirement#03

Sr. No.	Actual Model	Tool Generated	Status
1	Printer	Doctor	Correct
2	Printer-setup	Practitioners Doctors	Correct
3	Realization relationship between Printer and Printer-setup	Realization relationship between Printer and Printer-setup	Correct

Correct: 3, Incorrect: 0, Missing: 0, Extra: 0

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision = 3/ (3+0) =1=100%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)

Recall = 3/ (3+0) =1=100%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)

Overspecification = 0/ (3+0) = 0=0%

4.7.3 Direct Association

4.7.3.1 Requirement 04:

Members will be ready to search books.

4.7.3.1.1 Class Model of Requirement 04

Class model of Requirement 04 by R Yilmaz et al [71] is given Figure *41*.

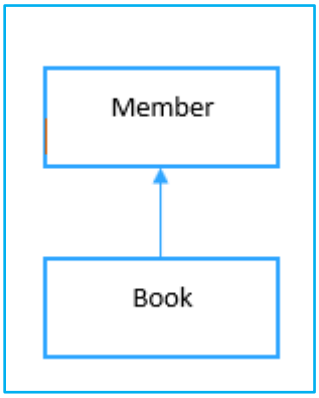

Figure 41 UML classes and direct association

4.7.3.1.2 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Requirement 04 in context of class identification and relationships are given in Figure *42*.

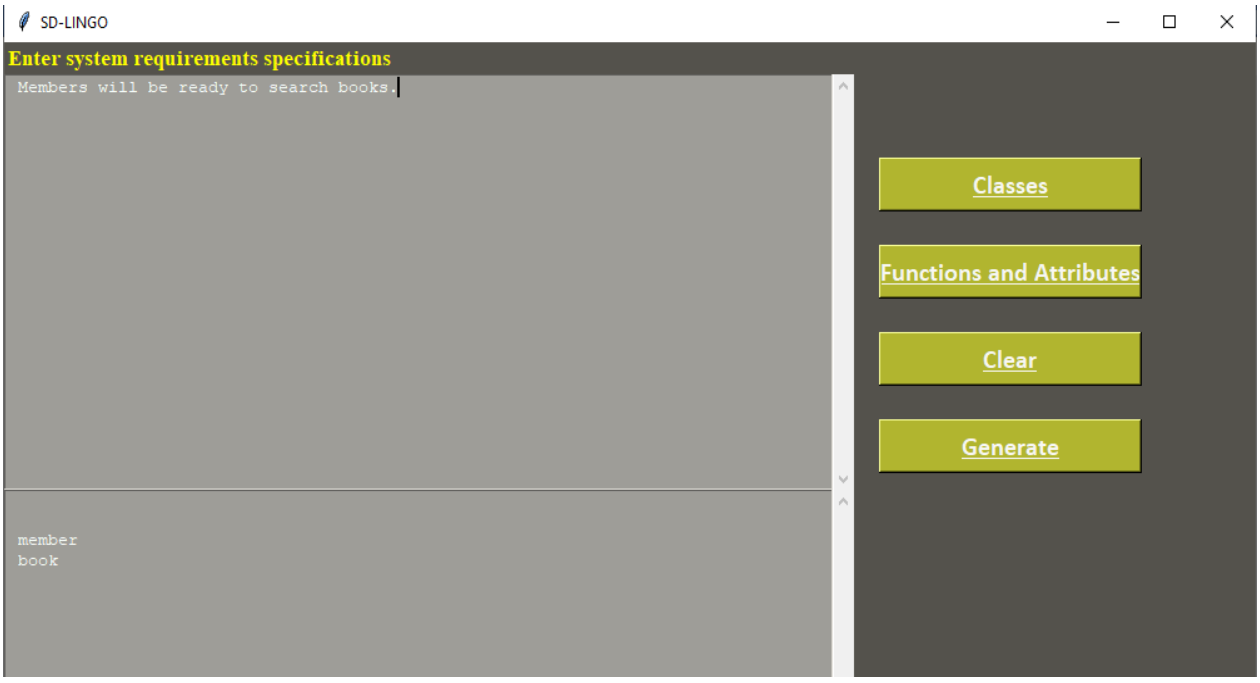

Figure 42 SD-Lingo screenshot for direct association

4.7.3.1.3 Class Diagram

Following class diagram is generated by SD-LINGO for Requirement 03 automatically from initial plain text as shown in Figure *43*.

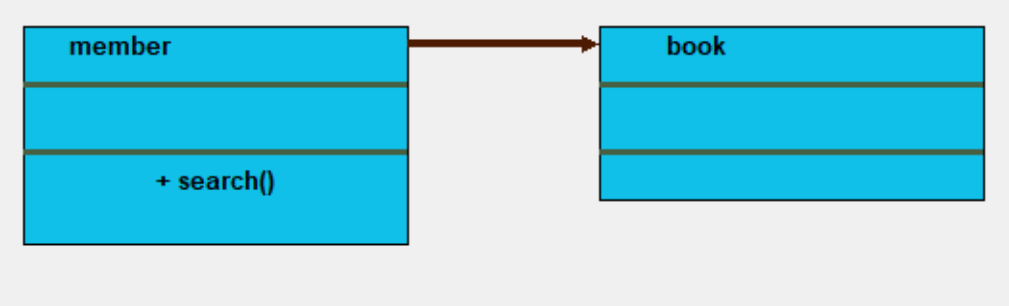

Figure 43 Classes show the direct association

4.7.3.1.4 Comparison of Actual Model and SD-LINGO Generated Class

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 20. SD-LINGO generated 3 correct classes and relationships, 0 incorrect class, 0 extra classes, 0 missing class, in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Precision, recall and over-specification of the selected case study is given below Table 20.

Table 20 Comparison of GC/AM for Requirement#04

Sr. No.	Actual Model	Tool Generated	Status
1	Member	Member	Correct
2	Book	Book	Correct
3	Direct Association relationship between member and book	Direct Association relationship between member and book	Correct

Correct: 3, Incorrect: 0, Missing: 0, Extra: 0

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision=3/ (3+0) =1=100%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)

Recall= 3/ (3+0) =1=100%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)

Overspecification = 0/ (3+0) = 0=0%

4.7.4 Multiplicity

4.7.4.1 Requirement 05:

One student can register 5 courses.

4.7.4.1.1 Class Model of Requirement 05

Class model of Requirement 05 by Sommerville [1] is given Figure *44*.

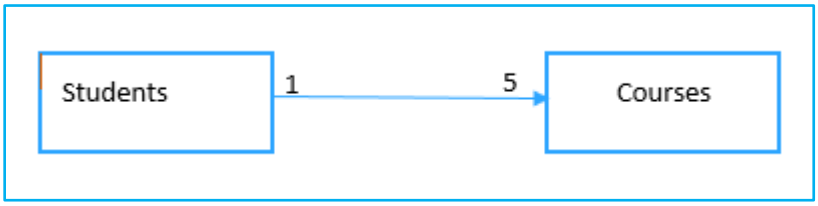

Figure 44 UML classes and Multiplicity for req#05

4.7.4.1.2 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Requirement 05 in context of class identification and relationships are given in Figure 45.

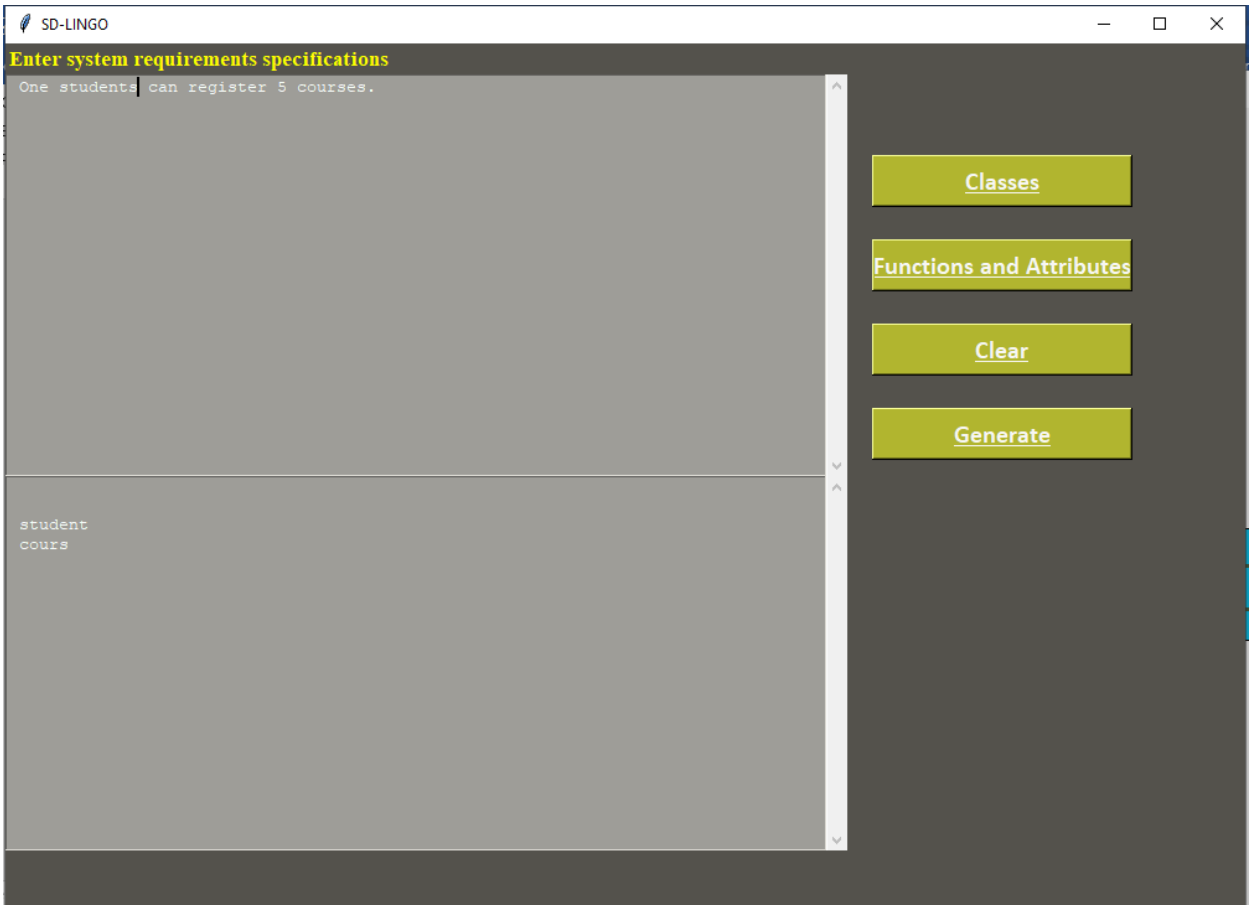

Figure 45 SD-Lingo tool screenshot for Req#05 multiplicity

4.7.4.1.3 Class Diagram

Following class diagram is generated by SD-LINGO for Requirement 05 automatically from initial plain text as shown in Figure *46*.

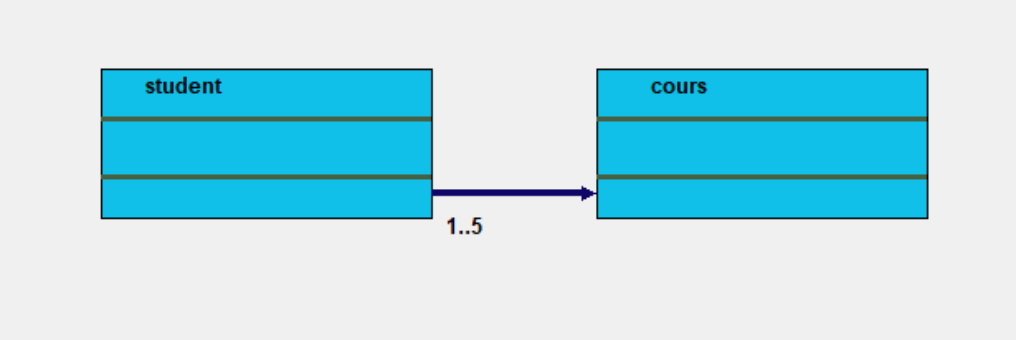

Figure 46 Classes show Req#05 multiplicity

4.7.4.1.4 Comparison of Actual Model and SD-LINGO Generated Class

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 21. SD-LINGO generated 3 correct classes and relationships, 0 incorrect class, 0 extra classes, 0 missing class, in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Precision, recall and over-specification of the selected case study is given below Table 21.

Table 21 Comparison of GC/AM for Requirement#05

Sr. No.	Actual Model	Tool Generated	Status
1	Student	Student	Correct
2	Courses	Courses	Correct
3	Multiplicity between student and courses	Multiplicity between student and courses	Correct

Correct: 3, Incorrect: 0, Missing: 0, Extra: 0

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision=3/ (3+0) =1=100%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)

Recall= 3/ (3+0) =1=100%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)

Overspecification = 0/ (3+0) = 0=0%

4.7.4.2 Requirement 06

Each patient has exactly 1 record.

4.7.4.2.1 Class Model of Requirement 06

Class model of Requirement 06 by Sommerville [1] is given Figure *47*.

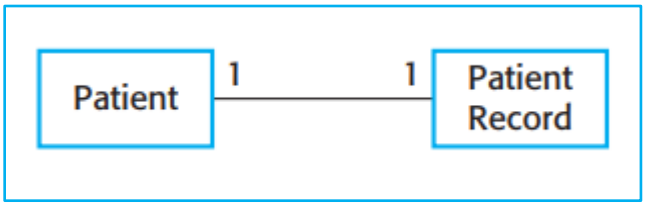

Figure 47 UML classes, Association, and multiplicity for req#06

4.7.4.2.2 SD-LINGO Screen Shot

Screenshot of SD-LINGO for Requirement 06 in context of class identification and relationships are given in Figure *48*.

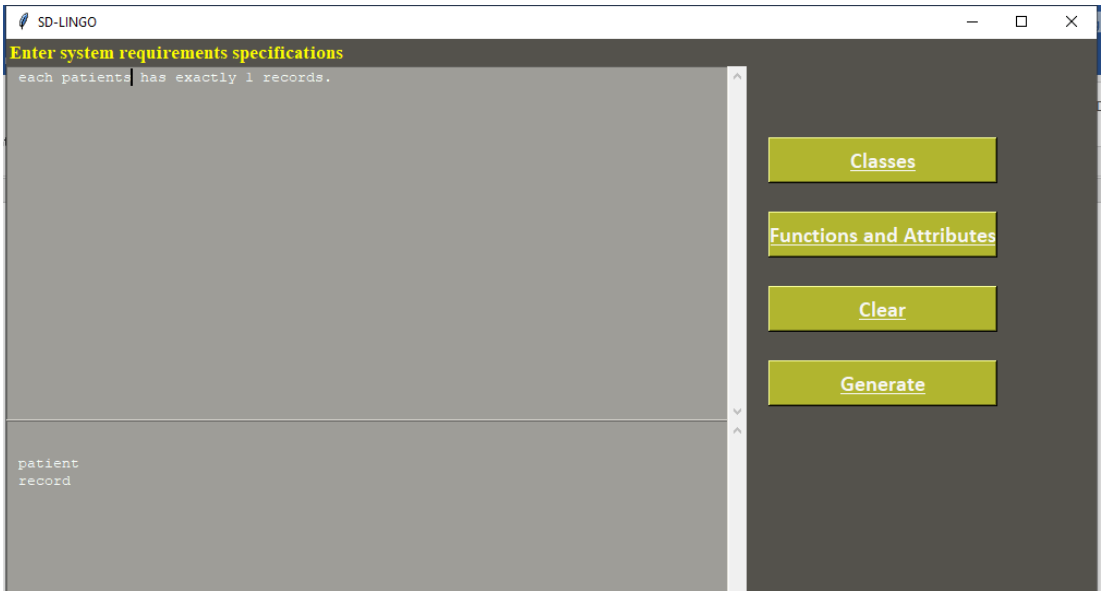

Figure 48 SD-Lingo tool screenshot for Req#06 multiplicity

4.7.4.2.3 Class Diagram

Following class diagram is generated by SD-LINGO for Requirement 06 automatically from initial plain text as shown in Figure *49*.

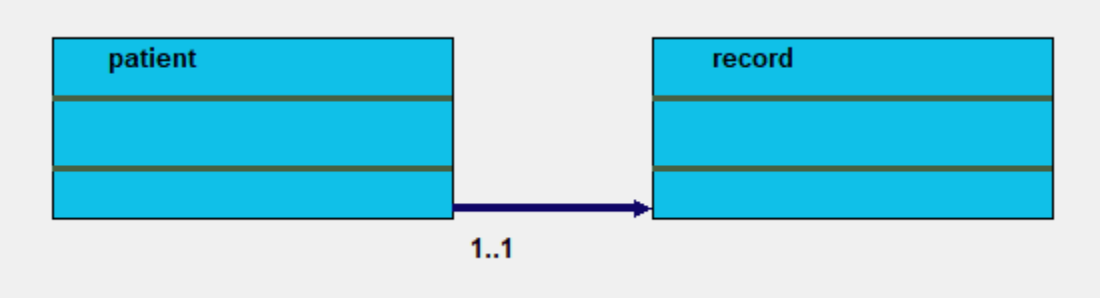

Figure 49 Classes show Req#06 multiplicity

4.7.4.2.4 Comparison of Actual Model and SD-LINGO Generated Class

Comparison of classes generated by SD-LINGO with the actual manual model is given in Table 22. SD-LINGO generated 3 correct classes and relationships, 0 incorrect class, 0 extra classes, 0 missing class, in comparison with the actual model where the number of correct instances made by the system is represented by '$N_{correct}$'. The number of incorrect responses made by the system refers to the '$N_{incorrect}$'. The number of elements missed by the system but still extracted by the human experts are represented by '$N_{missing}$'. Where the number of extra elements retrieved by the system is said to be 'N_{extra}'. Precision, recall and over-specification of the selected case study is given below Table 22.

Table 22 Comparison of GC/AM for Requirement#06

Sr. No.	Actual Model	Tool Generated	Status
1	Patient	Patient	Correct
2	Record	Record	Correct
3	Multiplicity between patient and record	Multiplicity between patient and record	Correct

Correct: 3, Incorrect: 0, Missing: 0, Extra: 0

Precision = $N_{correct}$ / ($N_{correct}$ + $N_{incorrect}$)

Precision=3/ (3+0) =1=100%

Recall = $N_{correct}$ / ($N_{correct}$+$N_{missing}$)

Recall= 3/ (3+0) =1=100%

Overspecification = N_{extra}/ ($N_{correct}$+$N_{missing}$)

Overspecification = 0/ (3+0) = 0=0%

CHAPTER 5
RESULTS AND EVALUATION

In order to test the performance of the proposed tool SD-LINGO, evaluation is conducted to different previous unseen natural language software requirements by the tool SD-LINGO. This performance parameter plays a vital role in software development for both stakeholders i.e., consumers and developers. Lynette et al [72] proposed three types of evaluation.

5.1 Adequacy Evaluation

It refers to the determination of the fitness of the system for some specific tasks. This type of evaluation is the basis to answer such types of questions: Will the system do, what is demanded? How good will it do the task? What is the cost related to completing a task? etc.

5.2 Diagnostics Evaluation

It refers to the type of evaluation that is used by the developers of the system to test their system during the phase of system development. For this kind of evaluation, a huge number of test data is required. The data is now as a basis to determine the system coverage and to fix all those founded flaws.

5.3 Performance Evaluation

It refers to the type of evaluation in which the performance of the system is measured in specific areas. In natural language processing, many concepts have been imported from quantitative performance evaluation to the development of evaluation methodologies. Lynette et al [72] addresses all above three measures of evaluation and all these concepts should be taken into account in every methodology of evaluation.

We are interested in the evaluation of speed, error rate and precision. The criteria of evaluation is mapped on the approach of Lynette et al [72] that how closely related the model developed by the analyst matched to the results proposed by our approach. However, no standard parameter regarding the evaluation criteria of software requirements in context of models does not exist. So not any model is categorized against correct or incorrect strictly. They may be categorized as good or bad requirements depending on the identified classes and their relationship. It is assumed in this research study that models given in OO text books are correct and good models so we took them all as our answer key. Given the chosen criteria, related to the system performance, i.e., hit to hit ratio and miss ratio, seconds to process ratio and % incorrect ratio. We have used just two metrics in this research study for the evaluation of our proposed system, Recall and Precision, developed for the evaluation of information retrieval system and also used as a measure to evaluate search strategies. In any ideal system precision and recall would be close to the 100%. In an information extraction system, recall is the completeness of the produced results

of the proposed system [73] . The relevant and the correct information produced by the proposed system is compared to the answer keys.

Recall is defined in the equation as:

$$Recall = \frac{Ncorrect}{\text{Ncorrect} + \text{Nmissing}}$$

Where number of correct instances made by the system is represented by 'Ncorrect'. Number of elements missed by the system but still extracted by the human experts are represented by 'Nmissing'. Evaluation term precision shows that how much the extracted knowledge was correct.
For the calculation of precision following formula is used:

$$Precision = \frac{Ncorrect}{\text{Ncorrect} + \text{Nincorrect}}$$

Where number of correct instances made by the system is represented by 'Ncorrect'. Number of incorrect responses made by the system refers to the 'Nincorrect' [73].
For the determination of the proper value of the given system & measure, a manual evaluation method is used in which the results generated by the proposed system are compared with the answer keys. SD-LINGO determines the classes/objects, their relationships and attributes.
Each concrete correct answer matches the answer key is said to be correct. If the answer does not match to the answer key, it is said to be incorrect. If the answer element does not match to the answer key but still if it is valid information then it is said to be extra.
If an answer matches the answer key it is said to be correct if it matches exactly (exact matching of string of names) to the answer key. If answer does not match exactly to that is answer key, we use problem statement and on the basis of our own judgement, will find the element in the answer key (but the rule is: that element must be semantically identical to the answer key element, means both of them must be referred to the same entity).
If the answer does not match to the answer key, it is said to be incorrect. And both the problem statement and our manual judgement validate that it is wrong i.e., adverb, adjective, or a verb in case study is given in a class incorrectly.

When an element is in the proposed model but not in the answer key, it is said to be missing element. There is one more evaluation metric of over-specification which represents that how much extra information in the system is not in the answer key. The object-oriented community agreed upon this thing that in the initial stages of analysis it is recommended to over-specify rather than to miss the important information [61].

$$Over - specification = \frac{Nextra}{\text{Ncorrect} + \text{Nmissing}}$$

Where number of extra elements retrieved by the system is said to be 'Nextra'. Where 'A set of six case studies are selected for different domains and set of six single line requirements are extracted from a text book are used to measure the performance of proposed tool SD-LINGO. This is the dictated choice but based on the fact that all the case studies are well known to the software engineers with the available solution and of intermediate length. Not, any of the case study was examined prior to the final evaluation in detail and not the system runs to any of the case study before the evaluation. The natural language software requirements in the case studies range from the 100-550 words and the sentence length is 5-39 words per sentence and the average sentence length is 18 words. Following Table 23 depicts the performance of SD-LINGO on six case studies. Every row represents the results for one case study while the last three columns represent the performance measurement parameters recall, precision and over-specification of SD-LINGO. System average [recall is 90.1%, precision is 97.4% and over-specification is 5.5%]. As said earlier, that system recall and precision should be as high as possible and the system over-specification is as low as possible. We can say that system performance is good on over-specification because the class list produced by the SD-LINGO is as close to the answer model. Ncorrect' and 'Nmissing' are defined earlier in this section.

5.4 Comparative Analysis

Comparative analysis is performed on the proposed system as follow to validate the performance and accuracy measure of SD-LINGO.

5.4.1 Performance Measurement of SD-LINGO on Benchmark Case Studies

Table 23 Evaluation summary of results from all case studies

Sr. No.	Case study and requirements	N_{cor}	N_{inc}	N_{mis}	N_{ext}	Precision (SD-LINGO) %	Recall (SD-LINGO) %	Over-Specification (SD-LINGO) %
1	LHP	4	1	1	1	80%	80%	20%
2	BAMS	8	1	1	2	88.8%	22.2%	22.2%
3	ATM	7	0	0	0	100%	100%	0%
4	Library	6	0	0	0	100%	100%	0%
5	JRP	5	0	0	1	100%	100%	17%
6	Course registration	11	0	3	1	100%	79%	7%
7	Req#01	5	0	0	0	100%	100%	0%
8	Req#02	5	0	0	0	100%	100%	0%
9	Req#03	3	0	0	0	100%	100%	0%
10	Req#04	3	0	0	0	100%	100%	0%
11	Req#05	3	0	0	0	100%	100%	0%
12	Req#06	3	0	0	0	100%	100%	0%
Average						97.4%	90.1%	5.5%

As formal evaluation of other case tools has been there in the past research studies. So, we can also compare results of SD-LINGO with the other two papers. Comparison with [24] is given in the Table 24 for five case studies.

Table 24 Evaluation summary of results from Mosa Elbendak et al [24]

Sr. No.	Case Study	Precision (SD-LINGO) %	Precision	Recall (SD-LINGO) %	Recall	Over-Specification (SD-LINGO) %	Over-Specification
1	LHP	80%	80%	80%	80%	20%	60%
2	BAMS	88.8%	80%	22.2%	88%	22.2%	88%
3	ATM	100%	91%	100%	100%	0%	16%
4	Library	100%	80%	100%	80%	0%	50%
5	JRP	100%	83%	100%	100%	17%	16%
Average		93.76%	82.8%	80.44%	89.6%	11.84%	46%

Results of case studies ATM, and course registration in comparison with [30] are shown in Table 25:

Table 25 Evaluation summary of results from Vidhu Bhala et al [30]

Sr. No.	Case Study	Precision (SD-LINGO) %	Precision	Recall (SD-LINGO) %	Recall	Over-Specification (SD-LINGO) %	Over-Specification
1	ATM	100%	91.7%	100%	91.7%	0%	8.33%
2	Course Reg.	100%	81.8%	79%	100%	7%	22.22
Average		100%	87%	90%	96%	3.5%	15%

As the formal evaluation of other case tools has been there in the past research studies. So, we can also compare the results of SD-LINGO with the other two papers. Comparison with AR2DT [43] is given in for five case studies.

Table 26 Evaluation summary of results from AR2DT [43]

Sr. No.	Case Study	Precision (SD-LINGO) %	Precision	Recall (SD-LINGO) %	Recall	Over-Specification (SD-LINGO) %	Over-Specification
1	LHP	80%	80%	80%	80%	20%	0%
2	BAMS	88.8%	72.7%	22.2%	88.9%	22.2%	88%
3	ATM	100%	100%	100%	91.7%	0%	16%
4	JRP	100%	83.3%	100%	100%	17%	16%
5	Course Reg.	100%	92.8%	79%	100%	7%	22.22%
Average		93.76%	85.76%	76%	92%	13.24%	28%

The results of the performance evaluation for the proposed tool SD-LINGO are very encouraging and proved efficient.

CHAPTER 6
CONCLUSION AND FUTURE RECOMMENDATION

This research thesis presents a Novel Natural Language Processing (NLP) approach to automatically generate a conceptual class model from initial software requirements. This research is carried out in three steps as follows:

In the first step, a comprehensive study is performed to investigate the application of NLP for primary software development phases i.e., requirement, design, and testing. A Systematic Literature Review (SLR) is carried out to select 29 articles published during 2014-2021. After quality Evaluation, only 17 articles consider that fully fulfills the objective of our research. Subsequently, 14 combinations of main NLP activities (i.e., Tokenization, POS tagging, Chunking, and Parsing) and 12 NLP algorithms are identified. Furthermore, 23 existing tools are identified that are further divided into two categories tools utilized by the researchers are 11 and purposed by researchers are 12. Finally, a comprehensive analysis is performed to investigate the automation level of NLP applications for the generation of the class diagrams and test cases from early plain text requirements. This SLR leads to identifying significant research gaps like the compulsion of manual pre-processing steps while automatically generating a conceptual class model from plain text through NLP approaches. Also, the existing not generate the fully automated class diagram.

In the second step, a novel fully automated NLP approach is proposed to generate a conceptual class model from initial software requirements. The proposed approach comprises 20 novels NLP rules to fully automate the class generation from initial requirements without requiring any manual pre-processing steps. As a part of the research, the SD-LINGO tool is developed.SD-LINGO generate different elements of class diagram such as classes, attributes, relationships.SD-LINGO is written in python with approximately twenty-four hundred lines of code. The tool used for front end development is Spyder and for drawing class diagrams we used Tkinter.

In the final step, we evaluate the performance of our proposed approach through six benchmarks studies and for validation of relationships; we used single line requirements used in different books. System average [precision is 97.4%, recall is 90.1% and over-specification is 5.5%]. For further investigation, the comparative analysis is performed with two high-quality journals and one tool AR2DT and our research performance shows many propitious results in almost every aspect. It is concluded that the proposed approach not only removes the compulsion of manual preprocessing steps but also outperforms the existing approaches with respect to performance. In the future, we will enhance SD-LINGO to automatically generate a complete class diagram from the raw text by adding more rules to achieve high accuracy.

REFERENCES

1. Sommerville, I., Software Engineering. 2015: Pearson Education.
2. Kneuper, R., Sixty years of software development life cycle models. IEEE Annals of the History of Computing, 2017. 39(3): p. 41-54.
3. Akinsola, J.E., et al. Comparative analysis of software development life cycle models (SDLC). in Computer Science On-line Conference. 2020. Springer.
4. Saravanan, T., et al. Comparative Analysis of Software Life Cycle Models. in 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN). 2020. IEEE.
5. Akbar, R., M. Haris, and M. Naeem. Requirement gathering and tracking process for distributed agile based development. in Proceedings of the 8th Conference on Recent Advances in Computer Engineering. 2008.
6. Yau, S.S. and J.J.-P. Tsai, A survey of software design techniques. IEEE Transactions on Software Engineering, 1986(6): p. 713-721.
7. Robillard, M.P., Sustainable software design, in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016, Association for Computing Machinery: Seattle, WA, USA. p. 920–923.
8. Chemuturi, M., Requirements Management Through SDLC, in Requirements Engineering and Management for Software Development Projects. 2013, Springer New York: New York, NY. p. 169-175.
9. Emadi, S. and F. Shams. From UML component diagram to an executable model based on Petri Nets. in 2008 International Symposium on Information Technology. 2008. IEEE.
10. Bell, D., UML basics: An introduction to the Unified Modeling Language. The Rational Edge, 2003.
11. Torchiano, M. Empirical assessment of UML static object diagrams. in Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004. 2004. IEEE.
12. Tonella, P. and A. Potrich, Package Diagram. Reverse Engineering of Object Oriented Code, 2005: p. 133-154.
13. Wei, R., et al., Automatic generation of UML profile graphical editors for Papyrus. Software and Systems Modeling, 2020. 19(5): p. 1083-1106.
14. Backman, S., Code Generationfor UML Composite Structure Diagrams. 2018.
15. Elallaoui, M., K. Nafil, and R. Touahni, Automatic transformation of user stories into UML use case diagrams using NLP techniques. Procedia computer science, 2018. 130: p. 42-49.
16. Ahmad, T., et al., Model-based testing using UML activity diagrams: A systematic mapping study. Computer Science Review, 2019. 33: p. 98-112.
17. Panthi, V., et al. Functionality testing of object-oriented software using UML state machine diagram. in 2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET). 2018. IEEE.
18. Alshayeb, M., et al., Improving the security of UML sequence diagram using genetic algorithm. IEEE Access, 2020. 8: p. 62738-62761.
19. Cvetković, J. and M. Cvetković, Evaluation of UML diagrams for test cases generation: Case study on depression of internet addiction. Physica A: Statistical Mechanics and Its Applications, 2019. 525: p. 1351-1359.

20. Mishra, A. Dynamic Slicing of UML Interaction Overview Diagram. in 2019 IEEE 9th International Conference on Advanced Computing (IACC). 2019. IEEE.
21. A. Maheshwari, A.T.a.D.S.K., A New Design Based Software Coupling Metric, in 2014 International Conference on Information Technology. 2014, IEEE: Bhubaneswar, India. p. 351-355.
22. Moreira, G.Y.d.O. and J.A.M. Santos. Applying coupling and cohesion concepts in object-oriented software: a controlled experiment. in 19th Brazilian Symposium on Software Quality. 2020.
23. Kanakaraddi, S.G. and S.S. Nandyal. Survey on parts of speech tagger techniques. in 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT). 2018. IEEE.
24. Elbendak, M., P. Vickers, and N. Rossiter, Parsed use case descriptions as a basis for object-oriented class model generation. Journal of Systems and Software, 2011. 84(7): p. 1209-1223.
25. Kulkarni, N., et al. Automated Analysis of Textual Use-Cases: Does NLP Components and Pipelines Matter? in 2012 19th Asia-Pacific Software Engineering Conference. 2012. IEEE.
26. Geogy, M. and A. Dharani. Prominence of each phase in Software development life cycle contributes to the overall quality of a product. in 2015 International Conference on Soft-Computing and Networks Security (ICSNS). 2015. IEEE.
27. Kitchenham, B., Procedures for performing systematic reviews. Keele, UK, Keele University, 2004. 33(2004): p. 1-26.
28. Conte, D.A. and J.C. Hauck. Automated Identification of Classes Using Natural Language Processing. in Proceedings of the XII Brazilian Symposium on Information Systems on Brazilian Symposium on Information Systems: Information Systems in the Cloud Computing Era-Volume 1. 2016.
29. Nasiri, S., Y. Rhazali, and M. Lahmer, Towards a generation of class diagram from user stories in agile methods, in Advancements in Model-Driven Architecture in Software Engineering. 2021, IGI Global. p. 135-159.
30. Sagar, V.B.R.V. and S. Abirami, Conceptual modeling of natural language functional requirements. Journal of Systems and Software, 2014. 88: p. 25-41.
31. Santos, C.J., S. Moshtari, and M. Mirakhorli. An Automated Approach to Recover the Use-case View of an Architecture. in 2020 IEEE International Conference on Software Architecture Companion (ICSA-C). 2020. IEEE.
32. Sanyal, R. and B. Ghoshal. Automatic extraction of structural model from semi structured software requirement specification. in 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS). 2018. IEEE.
33. Jaiwai, M. and U. Sammapun. Extracting UML class diagrams from software requirements in Thai using NLP. in 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE). 2017. IEEE.
34. Sharma, R., P.K. Srivastava, and K.K. Biswas. From natural language requirements to UML class diagrams. in 2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE). 2015. IEEE.
35. Abdelnabi, E.A., et al. Generating UML Class Diagram using NLP Techniques and Heuristic Rules. in 2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA). 2020. IEEE.

36. Tripathy, A. and S.K. Rath. Application of natural language processing in object oriented software development. in 2014 International Conference on Recent Trends in Information Technology. 2014. IEEE.
37. Thakur, J.S. and A. Gupta. Identifying domain elements from textual specifications. in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016.
38. Bashir, N., et al. Modeling Class Diagram using NLP in Object-Oriented Designing. in 2021 National Computing Colleges Conference (NCCC). 2021. IEEE.
39. Saini, R., et al. Towards queryable and traceable domain models. in 2020 IEEE 28th International Requirements Engineering Conference (RE). 2020. IEEE.
40. Narawita, C.R. and K. Vidanage. UML generator-an automated system for model driven development. in 2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer). 2016. IEEE.
41. Abdelnabi, E.A., A.M. Maatuk, and M. Hagal. Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques. in 2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA. 2021. IEEE.
42. Bousmaha, K.Z., et al. A Platform for the Conceptualization of Arabic Texts Dedicated to the Design of the UML Class Diagram. in International Conference on Applications of Natural Language to Information Systems. 2016. Springer.
43. Ahmed, M.A., et al. A novel natural language processing (NLP) approach to automatically generate conceptual class model from initial software requirements. in International Conference on Information Science and Applications. 2017. Springer.
44. Utama, A.Z. and D.-S. Jang, An Automatic Construction for Class Diagram from Problem Statement using Natural Language Processing. Journal of Korea Multimedia Society, 2019. 22(3): p. 386-394.
45. Wang, C., et al., Automatic generation of system test cases from use case specifications, in Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015, Association for Computing Machinery: Baltimore, MD, USA. p. 385–396.
46. Bahaweres, R.B., et al. Behavior-driven development (BDD) Cucumber Katalon for Automation GUI testing case CURA and Swag Labs. in 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS). 2020. IEEE.
47. Jali, N., D. Greer, and P. Hanna. Class Responsibility Assignment (CRA) for Use Case Specification to Sequence Diagrams (UC2SD). in 2014 8th. Malaysian Software Engineering Conference (MySEC). 2014. IEEE.
48. Jellali, I., et al. GSM model construction from enterprise models. in 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE). 2015. IEEE.
49. Sawprakhon, P. and Y. Limpiyakorn. Model-driven approach to constructing uml sequence diagram. in 2014 International Conference on Information Science & Applications (ICISA). 2014. IEEE.
50. Mahmood, A., et al. Natural language processing based interpretation of skewed graphs. in 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). 2014. IEEE.
51. Omer, M.A.M. and D. Wilson. New rules for deriving formal models from text. in 2016 International Conference for Students on Applied Engineering (ICSAE). 2016. IEEE.

52. Mosca, N., et al., Object-Based Similarity Assessment Using Land Cover Meta-Language (LCML): Concept, Challenges, and Implementation. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2020. 13: p. 3790-3805.
53. Andrianjaka, R.M., et al. Restructuring extended Lexical elaborate Language. in 2019 23rd International Conference on System Theory, Control and Computing (ICSTCC). 2019. IEEE.
54. Luc, R.J., et al. Semantic aspect derivation of the Praxème methodology from the elaborate lexicon extended language. in 2016 20th International Conference on System Theory, Control and Computing (ICSTCC). 2016. IEEE.
55. Mahmood, A., I.S. Bajwa, and K. Qazi. An automated approach for interpretation of statistical graphics. in 2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics. 2014. IEEE.
56. Autili, M., et al., A tool-supported methodology for validation and refinement of early-stage domain models. IEEE Transactions on Software Engineering, 2015. 42(1): p. 2-25.
57. Meteer, M., et al. MedLingMap: A growing resource mapping the Bio-Medical NLP field. in BioNLP: Proceedings of the 2012 Workshop on Biomedical Natural Language Processing. 2012.
58. Umber, A., I.S. Bajwa, and M.A. Naeem. NL-based automated software requirements elicitation and specification. in International Conference on Advances in Computing and Communications. 2011. Springer.
59. Deeptimahanti, D.K. and M.A. Babar. An automated tool for generating UML models from natural language requirements. in 2009 IEEE/ACM International Conference on Automated Software Engineering. 2009. IEEE.
60. Kumar, D.D. and R. Sanyal. Static UML model generator from analysis of requirements (SUGAR). in 2008 Advanced Software Engineering and Its Applications. 2008. IEEE.
61. Larman, C., Applying UML and patterns: an introduction to object oriented analysis and design and interative development. 2012: Pearson Education India.
62. Booch, G., I. Jacobson, and J. Rumbaugh, Object-Oriented Analysis and Design with Applications Third Edition. 2007, Pearson Education, Inc.
63. Leffingwell, D., Agile software requirements: lean requirements practices for teams, programs, and the enterprise. 2010: Addison-Wesley Professional.
64. Ambler, S.W., The elements of UML (TM) 2.0 style. 2005: Cambridge University Press.
65. Lindland, O.I., G. Sindre, and A. Solvberg, Understanding quality in conceptual modeling. IEEE software, 1994. 11(2): p. 42-49.
66. Falleri, J.-R., et al. Using natural language to improve the generation of model transformation in software design. in 2009 International Multiconference on Computer Science and Information Technology. 2009. IEEE.
67. Shipman, J.W., Tkinter 8.4 reference: a GUI for Python. New Mexico Tech Computer Center, 2013. 54.
68. Duffy, D., From Chaos to Classes: Object-Oriented Software Development in C++. 1995: McGraw-Hill Companies.
69. Blaha, M. and J. Rumbaugh, Object-oriented modeling and design with UML. 2005: Pearson Education India.
70. Kruchten, P., The rational unified process: an introduction. 2004: Addison-Wesley Professional.

71. Yilmaz, R., et al., Object-Oriented Programming in Computer Science, in Encyclopedia of Information Science and Technology, Fourth Edition. 2018, IGI Global. p. 7470-7480.
72. Hirschman, L. and H.S. Thompson, Overview of evaluation in speech and natural language processing. 1997.
73. Grishman, R. and B.M. Sundheim. Message understanding conference-6: A brief history. in COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics. 1996.