Automatic generation of analysis class diagrams from use case specifications

Jitendra Singh Thakur · Atul Gupta

July 29, 2016

Abstract In object oriented software development, the analysis modeling is concerned with the task of identifying problem level objects along with the relationships between them from software requirements. The software requirements are usually written in some natural language, and the analysis modeling is normally performed by experienced human analysts. The huge gap between the software requirements which are unstructured texts and analysis models which are usually structured UML diagrams, along with human slip-ups inevitably makes the transformation process error prone. The automation of this process can help in reducing the errors in the transformation. In this paper we propose a tool supported approach for automated transformation of use case specifications documented in English language into analysis class diagrams. The approach works in four steps. It first takes the textual specification of a use case as input, and then using a natural language parser generates type dependencies and parts of speech tags for each sentence in the specification. Then, it identifies the sentence structure of each sentence using a set of comprehensive sentence structure rules. Next, it applies a set of transformation rules on the type dependencies and parts of speech tags of the sentences to discover the problem level objects and the relationships between them. Finally, it generates and visualizes the analysis class diagram. We conducted a controlled experiment to compare the correctness, completeness and redundancy of the analysis class diagrams generated by our approach with those generated by the existing automated approaches. The results showed that the analysis class diagrams generated by our approach were more correct, more complete, and less redundant than those generated by the other approaches.

Keywords Model transformation · Analysis modeling · Analysis class diagrams · Natural language processing · Automated approach

1 Introduction

The model driven object oriented software development process (Kleppe et al 2003; Kuznetsov 2007) can be visualized as a sequence of transformation steps where the first set of transformations is about constructing analysis models also called Platform Independent Models (PIM) from textual requirements. The the second set of transformations is about developing design models also called Platform Specific Models (PSM) from the analysis models. And the third set of transformations is about converting the design models into code. Understandably, an essential requirement and the underlying assumption here is that these transformations are loss-less for the requirements to be successfully transformed into a software product that will meet the user requirements. However, this is normally not the case as each of these transformations deals with a complex problem of identifying relevant model elements, their properties, and the contextual relationships between them. The complexity of the problem along with the human slip-ups (Norman [1981]) make these transformations error prone, non-repeatable and grossly influenced by the skills of these developers.

Here we focus on analysis modeling activity (also referred as object oriented analysis) which is about discovering the domain classes, attributes, operations and the relationships between the domain classes from the vocabulary of the problem domain to obtain the analysis models (Grady Booch [2010]). The Object Management Group’s (OMG) Model Driven
Architecture (MDA) refers these models as the Platform Independent Models (PIM) as they represent the functionality, behavior and structure of the system at the problem level. These models do not contain the platform specific details about the implementation of the system (Kuznetsov 2007). They represent different viewpoints of a problem domain, so as to allow the developers to talk and reason about the domain objects in order to enhance their understanding of the problem and avoid potential pitfalls (O’Docherty 2005).

The analysis modeling is usually carried by human analysts. This activity requires the analysts to read and analyse typically several hundred pages of software requirement specifications which involves significant efforts and time. Moreover, the chances of this transformation to be lossy are even more prominent as it requires understanding the user requirements documented as natural language (NL) texts and mapping these textual specifications to domain models like the analysis class diagrams. Naturally, the varying skills of human analysts, their understanding of domain knowledge and of the mapping process make the transformation non-repeatable. The different analysts may derive different analysis models (Harmain and Gaizauskas 2003; Deeptimahanti and Babar 2009) which may capture the domain information incompletely and may also be inconsistent with the problem specification. The problem gets even worse when it comes to keep the domain models in sync with changing requirements.

Sensing the problem, there have been some efforts in the direction of framing semi-automated and automated approaches that can help to generate the analysis models from textual specifications. The semi-automated approaches assist the user in deriving the analysis models but most of them are highly dependent on the user skills for identifying the elements of the analysis models (Overmyer et al 2001; Harmain and Gaizauskas 2003; Samarasinghe and Somé 2005). On the other hand the automated approaches (Harmain and Gaizauskas 2003; Liu et al 2004; Ilieva and Ormandjieva 2006; Popescu et al 2008; Yue et al 2013a) though do not require human interventions, but have some serious issues such as failing to identify the major elements of the analysis models, incomplete transformation of requirements into analysis models, identification of analysis model elements which are semantically incorrect, redundancy in the generated models and presence of many unconnected components in the models.

In this paper we propose an automated approach to generate the analysis class diagrams from use case specifications that overcomes many of the shortcomings of the existing approaches. The approach first parses the sentences of the input use case specification (UCS) using the Stanford NL parser to generate parts of speech tags (POS-tags) and type dependencies (TDs) (POS-tags represent the tagging of each word in a sentence with parts of speech such as noun, pronoun, verb, adjective, adverb etc. TDs (De Marneffe et al 2006; De Marneffe and Manning 2008) represent grammatical relationships between the words of a sentence. POS tags and TDs are discussed in more detail in Section 4.1.3). Then it applies the proposed set of comprehensive sentence structure rules on TDs and POS-tags of the sentences to identify their sentence structures. We have framed these comprehensive sentence structure rules based on the twenty five verb patterns proposed by A.S.Hornby (known for various achievements in linguistic and literature) (Hanks 2008) in Oxford Advanced Learners Dictionary of Current English (A.S. Hornby 1974, 2000) and in Hornby (1975). After identifying the sentence structures, the approach then applies the proposed set of comprehensive transformation rules on TDs and POS-tags to identify the potential elements for the generation of analysis class diagram. The transformation rules take into account the sentence structure of the sentences as well as the syntactic and semantic relationships between the words of the sentences to precisely identify these elements. The approach finally generates the analysis class diagram and using the GraphViz API visualizes the generated diagram. We have prototyped the proposed approach in a GUI based tool named AutoAMG (Automatic Analysis Model Generator).

As a validation of the proposed approach, we report the outcome of a controlled experiment conducted to compare the analysis class diagrams generated by our approach with those generated by the other two existing approaches - one proposed by Popescu et al (2008) and the other by Yue et al (2013a, 2015). The approaches were compared on the basis of correctness, completeness and redundancy of the analysis class diagrams generated by them for the forty UCSs taken from various software engineering books (O’Docherty 2005; Gomaa 2011; Rosenberg and Stephens 2008, 2007; Bruegge and Dutoit 1999) and research works (Liu et al 2004; Popescu et al 2008; Yue et al 2013a, 2015; Deeptimahanti and Sanyal 2011). The experiment design was a complete block design in which forty subjects evaluated the analysis class diagrams obtained from the three approaches. Each subject was randomly given one UCS, the analysis class diagrams obtained by the three approaches for the given UCS and a set of questionnaires for each class diagram. The questionnaires were based on the quality measures for analysis class diagrams presented in Section 6.2. For statistical analysis of the data obtained from answers to the questionnaires collected from the subjects, we first applied Kolmogorov-Smirnov test (Massey Jr 1951) and found that the data was non-normal. Then we applied the Friedman test (a non parametric test) (Corder and Foreman 2009; Sheskin 2003) on the data that showed significant differences between the analysis class diagrams generated using the

1 http://www.omg.org/mda/
2 http://nlp.stanford.edu/software/lex-parser.shtml
3 http://www.graphviz.org/; https://github.com/jabbalaci/graphviz-java-api
three approach. Next we applied post hoc Friedman tests on the data for pairwise comparison of the analysis class diagrams generated by the three approaches to find whether the class diagrams obtained from one of the approach are better in terms of correctness, completeness and redundancy than those obtained from the other approaches. The results clearly showed that the analysis class diagrams generated by the proposed approach were significantly better than those generated by other two existing approaches in terms of the correctness, completeness and redundancy. Specifically, the results for the forty UCSs showed that the analysis class diagrams generated by our approach were 46% more correct, 55% more complete and 31% less redundant than those generated by Popescu et al. approach, and were 33% more correct, 31% more complete and 31% less redundant than those generated by Yue et al. approach.

The main contributions of this paper are as follows:
1. We propose an automated approach to generate analysis class diagrams from UCSs.
2. We framed a comprehensive set of sentence structure rules to identify the sentence structures of the sentences, and a comprehensive set of transformation rules to identify the elements of the analysis class diagrams.
3. We developed a tool support AutoAMG for the above stated automation.
4. We report on a controlled experiment that we have conducted with 40 subjects and 40 UCSs to compare the correctness, completeness and redundancy of the analysis class diagrams generated by the proposed approach with those generated by other two exiting automated approaches.

The rest of the paper is organized as follows. Section 2 presents the background of the proposed work. Section 3 presents the motivation and the problem description. Section 4 presents the proposed approach. Section 5 presents the tool support that we have developed for the automation. Section 6 presents the details of the experiment conducted for the evaluation of the proposed approach. Section 7 discusses the strengths and limitations of the approach. Section 8 presents the related work in the literature. Section 9 presents the conclusion and future directions of the present work.

2 Background

The earliest attempt for analysis of textual requirements to identify classes and objects was made by Abbott (1983), in which he proposed extracting the classes and their operations using noun-verb analysis of textual description of the problem. The Nouns found in the text represent candidate classes and the verbs represent the candidate operations. The idea of simple noun-verb textual analysis was useful but suffers from dilemma of enlisting incorrect candidate objects and operations as any noun can be verbed, and any verb can be nouned (Grady Booch 2010).

Another technique to identify the potential classes proposed by Arango (1989), in which an analyst consults domain experts and identifies the candidate objects, operations, and relationships from the vocabulary that the domain experts use while speaking about the given problem. A domain expert may be a user of the system or any person who is familiar with all the details of the given system.

Shlaer and Mellor (1988) and Yourdon and Coad (1990) suggested other ways of identifying the prospective classes and objects from sources such as perceptible things (ATM card, customer record, sensors etc), roles (roles played by the people who interact with the system), events (an event that must be remembered), sentence structures (“is a” and “part of” relationships), other systems (other system with which this system interacts), devices (with which the system interacts) and organizational units (companies, departments, divisions etc. that the system must keep tract of) discussed in the problem description.

Beck and Cunningham (1989) suggested an idea of using simple cards that they named CRC (Class Responsibility Collaborator) cards for teaching the concepts of objects to novice programmers, and for introducing complicated existing designs to experienced programmers. CRC cards became an effective method for analyzing use case scenarios by a team of analysts to obtain analysis models.

The advancements in the field of natural language processing motivated researchers to automate the process of analysis modeling. Mich (1996), Mich and Garigliano (2002) proposed a CASE tool NL-OOPS (Natural Language - Object Oriented Production System) for identifying classes and relationships using a semantic network of words of a natural language processing system LOLITA (Large-scale Object-based Language Interactor, Translator and Analyser). The tool requires user intervention for deleting the extra nodes representing spurious classes, and to set the level of details for class hierarchy.

Overmyer et al (2001) proposed another semi automated approach supported by a tool named Linguistic Assistant for Domain Analysis (LIDA) that uses POS tags generated by NL parser to present the user with list of nouns, adjectives and verbs from which user can identify and mark the candidate classes, attributes and the methods. A Similar approach is proposed by Harmain and Gaizauskas (2003) named CM-Builder1, but there approach also assigns frequency of reference in the text to each class which helps the user to identify the candidate classes.
Further efforts have been made to remove the human intervention and to fully automate the transformation process. Liu et al. (2004) proposed an automatic approach named UCDA for generating analysis class diagrams using POS tags generated by a NL parser from UCS written using restricted grammar. Their approach processes the sentences based on the classification of sentences as transitive, intransitive, ditransitive, intensive, complex transitive, prepositional and non-finite given in Roberts (1956). The approach was dependent on using a glossary for identifying classes, it fails to identify attributes, relationship names, aggregation relationships and generalization relationships. Ilieva and Ormandjieva (2006) proposed an approach to generate domain models from unrestricted NL requirements. The approach was based on using a semantic network of words created from the POS-tags generated by parser. The semantic network is then transformed into the domain model. The domain model confine only the identified classes and the relationship between them, it lacks the attributes, operations, relationship names and relationship types (association, generalization and aggregation).

Popescu et al. (2008) proposed a different approach whose main objective was to identify inconsistencies in the requirement specifications but with the help of automatically generated domain models. The approach processes the requirement specifications using the constraining grammar proposed in Juristo et al. (2000). The approach uses link types generated by link grammar parser to extract the domain elements from such sentences in the requirement specification. The generated domain models can then be examined by the user to identify the inconsistencies. The analysis models obtained are highly incomplete with many unconnected components.

Recently, Yue et al. (2013a, 2015) proposed another automated approach for deriving analysis models from UCSs using parse trees and POS tags generated by a NL parser. The approach requires the input UCSs to be written using twenty six restrictions rules proposed in their previous work. Their approach processes the sentences on the basis of sentence structures formed using five basic English sentence patterns given in Greenbaum (1996). The approach does not recognize domain objects and attributes which are documented as a group of words (nouns). The generated class diagrams have many unconnected components. The approach dumps most of the operations in a single control class, whereas most of the other classes are assigned no operations at all.

3 Motivation and problem description

The analysis modeling is usually performed by human analysts in which the human analysts read the textual requirements to identify the elements for the generation of analysis class diagram. The analysis class diagram consists of the identified domain classes (entity, boundary and control) their attributes and operations, and the relationships (association, generalization and aggregation) between the domain classes. The huge gap between the software requirements and analysis models, along with human slip-ups (Norman 1981) inevitably make the transformation of the requirements into analysis models error prone. Because of the varying skills of human analysts, their understanding of domain knowledge and of the mapping process, the different analysts may derive different analysis models from the same set of requirements (Harman and Gaizauskas 2003; Deeptimahanti and Babar 2009), which makes the transformation non-repeatable. The problem gets even worse when the domain models are to be kept in sync with the changing requirements. The automation of this process can help in reducing these problems.

As there are many possible ways in which a same concept or thing can be expressed in a natural language such as English, the relevant elements to construct the analysis model are embedded at different positions in different sentences of the requirement specifications. Hence to extract the relevant elements from the text, the two foremost requirement for an automated approach are 1) It should be able to recognize the structures of the sentences in the specification, 2) It should be able to accurately obtain the syntactic and semantic relationships between the words in the sentences. An ideal automated approach should be able to interpret all the possible sentences of the natural language without any restrictions. The set of sentence patterns that an automated approach can interpret is referred in the paper as language model. The stronger the language model utilized by an approach makes the approach more useful and more appealing. But, the issues associated with the natural language such as ambiguity, variety of sentence types, anaphora (or pronoun) resolution problem and inconsistency (Kamsties and Peach 2000; Nuseibeh et al. 2001; Fabbrini et al. 2001; Yang et al. 2010) makes the interpretation of all the possible sentences hard for an automated approach. Therefore to overcome the problems associated with natural languages, the automated approaches impose restrictions on free usage of natural language to specify the requirements, but the requirements analysts (or use case authors) don’t want too much restrictions (too much restrictions prevent them in expressing their ideas).

As discussed in the last section there have been a few efforts in the direction of framing semi-automated and automated approaches to derive the analysis class diagrams from textual specifications. The major drawback of the semi automated approaches (Mich 1996; Mich and Garigliano 2002; Overmyer et al. 2001; Harman and Gaizauskas 2003) was their high dependency on the user to identify the elements for generating analysis class diagrams. The automated approaches (Liu et al. 2004; Ilieva and Ormandjieva 2006) though do not require any human intervention but they have some serious issues such
as the approaches do not identify the major elements of the analysis class diagrams (Table 1) and require glossary files (files containing list of domain specific terms) to identify the classes. The automated approaches proposed by Popescu et al (2008), and Yue et al (2013a, 2015) identify the major elements for generating analysis class diagrams (Table 1) without using any glossary file to identify the domain classes but they too have some serious issues which are listed in Table 2 along with the possible causes.

### Table 1 Analysis class diagram elements identified by the existing automated approaches

| Approach         | NLP constructs used | Input                        | Class Diagram Elements Identified (Output) | Class  | Attributes | Operations | Relationship Name | Association | Aggregation | Generalization |
|------------------|---------------------|------------------------------|--------------------------------------------|--------|------------|------------|-------------------|------------|-------------|--------------|
| Li et al (2006)  | POS tags            | Restricted NL Text          | ✓                                          | ✓      | ✓          | ✓          | ✓                 | ✓          | ✓          | ✓            |
| Aura and Ormandjieva (2006) | POS tags | Unrestricted NL Text | ✓                                          | ✓      | ✓          | ✓          | ✓                 | ✓          | ✓          | ✓            |
| Popescu et al (2008) | Link types | Restricted NL Text | ✓                                          | ✓      | ✓          | ✓          | ✓                 | ✓          | ✓          | ✓            |
| Yue et al (2013a) | Parse tree         | Restricted NL Text          | ✓                                          | ✓      | ✓          | ✓          | ✓                 | ✓          | ✓          | ✓            |

* Can not identify generalization relationship from text but obtains generalization relationship between control classes of UCSs directly from generalization field of the UCS

### Table 2 Issues in existing automated approaches

| Approach         | Issues                                                                 | Possible causes of the Issues                                                                 |
|------------------|------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| A1 Yue et al (2013a, 2015) | 1) The analysis class diagrams generated by the approach contains many isolated classes and unconnected components (chunks of classes and relationships). 2) The class diagrams lacks several relevant classes and relationships (domain classes and relationships which should be deduced from the problem specifications) 3) A number of identified domain classes are difficult to be semantically termed as domain classes as they seems to violate the encapsulation principle. 4) The class diagrams contain some redundant classes and relationships. | 1) The language model (set of sentence patterns used by the approach to interpret the sentences) is based on the five sentence patterns proposed in Popescu et al (2008), is unable to interpret the variety of simple sentences in English like those involving infinitive (e.g. “The system prompts to enter the password”), the system commands the motor to start), present participle (e.g. “The system prints the receipt showing transaction number and date”), past participle (e.g. “The system validates the record entered by the customer”), gerund (e.g. “The system starts printing the document”), etc. 2) The language model (set of sentence patterns used by the approach to interpret the sentences) is not comprehensive enough to interpret common complex sentences like those involving that clause (e.g. “The system checks that the password is correct”), conjunctive clause (e.g. “The system stops the motor when the tank is full”), “The motor stops when the tank is full.”) etc. 3) The transformation rules identify the entity classes only from those noun phrases in the sentences that either contains a single noun (e.g. “customer”), or a single noun with a determiner (e.g. “The customer”), or a possessive noun (e.g. “customer’s address”). 4) For transforming rules to extract aggregation and generalization relationships from the sentences in UCSs. |

In this paper we aim to propose a tool supported automated approach that can generate analysis class diagrams which are more correct, more complete and less redundant than those generated by existing automated approaches. The proposed approach takes as input the software requirements documented as use case specifications (UCSs) with a few restrictions shown in Table 6. To handle the diverse set of sentences in the specifications the approach systematically process the sentences on the basis of their sentence structures identified using the proposed comprehensive sentence structure rules. To correctly extract potential classes, their attributes, operations and the relationships, the approach uses the proposed set of comprehensive transformation rules. We have framed these transformation rules to precisely extract the relevant elements of analysis class diagram from the text through various drills on hundreds of sentences of each sentence pattern. For extracting the relevant elements from the text, the transformation rules take into account the sentence structure of the sentences as well as the syntactic and semantic relationships between the words of the sentences. For finding these relationships between the words of the sentences, the transformation rules use type dependencies (TDs) and part of speech tags (POS-tags) generated using Stanford Natural Language Parser AP[4] The approach finally visualizes the generated analysis class diagram in place with the help of GraphViz API.

---

[4] http://nlp.stanford.edu/software/lex-parser.shtml
We conducted a controlled experiment to compare the analysis class diagrams generated by the proposed approach with those generated by two existing automated approaches (Popescu et al. 2008) and (Yue et al. 2013a, 2015) for the forty UCSs obtained from various software engineering books and research papers. In the experimental forty subjects evaluated the analysis class diagrams generated by the three approaches for the forty UCSs in terms of correctness, completeness and redundancy. The results showed that the analysis class diagrams generated by our approach were significantly more correct, more complete, and less redundant than those generated by the other approaches.

4 Proposed approach

This section describes how the proposed approach obtains the analysis class diagrams from the input use case specifications. Section 4.1 describes the artifacts which are the building blocks of the proposed approach. Then Section 4.2 step by step describes how the input UCS is read and parsed, and how the various elements of analysis class diagram are identified from the UCS.

4.1 Building blocks of the proposed approach

This section presents the artifacts that are the backbone of the proposed approach. Section 4.1.1 presents the UCS template used as input in the approach. Section 4.1.2 presents the use case meta model used to store the elements of the input UCS, and class diagram meta model used to store the class diagram elements identified by the approach. Section 4.1.3 presents the Natural Language Processing (NLP) constructs (POS-tags and TDs) used by the approach to analyse the textual specifications, and to extract the class diagram elements from the text. Section 4.1.4 presents the language model used by the approach, and the restriction rules to be used for documenting the UCS.

4.1.1 Use case model

Use case diagrams (Example Figure 1) along with their textual specifications (Example Table 3) are commonly used for documenting the functional requirements of a system. A use case specification (UCS) defines the functional requirements of a system in terms of a sequence of actions performed by the system and the actor(s) of the system that provides the desired function for the actor(s).

Our approach takes as input the functional requirements documented as UCS (Example Table 3). The presented UCS template contains the most common elements (name, description, actor, basic flow and alternate flow) (Siqueira and Silva 2011) that are used for documenting the UCS. A part from these essential elements the template also contains a few more elements viz, parent actor name to specify actor generalization relationships which are shown in a use case diagram, parent use case name to specify use case generalization relationships which are shown in a use case diagram, and sub flow to specify the steps that can be performed in parallel with some steps in main flow (basic flow). Each flow step in our use case template can contain a pre-condition and a post-condition. The element alternate flow is divided into two sub elements viz. specific alternate flow and global alternate flow as done in Yue et al. (2013b) template, where the specific alternate flows specifies the steps which are alternatives to some specific step in the UCS, and the global alternate flow specifies the steps whose scope is all the other steps in the UCS (e.g. If a user press cancel or select cancel, when some other step is being carried out in the UCS.)

Fig. 1 Partial use cases of ATM system
Table 3 Use specification (UCS): Withdraw fund taken from Yue et al. [2013a, 2015], but a few modifications done to fit our template

| Use Case Name: | Withdraw Fund |
|---------------|---------------|
| Main System Name: | ATM |
| Description: | Represents steps to withdraw cash from ATM. |
| Constraints: | The system must process the transaction within 20 seconds |

| Actor No. | Actor Type | Actor Name | Parent Actor Name |
|-----------|------------|------------|-------------------|
| 1         | Primary    | ATM Customer | User              |
| 2         | Secondary  |             |                   |

| Pre Condition/Guard | FlowId | Step | Sub FlowId | Alternate FlowId | Post Condition |
|---------------------|--------|------|------------|-----------------|----------------|
| ATM customer has inserted an ATM card in the card reader | M1 | INCLUDE USE CASE Validate PIN. | | | |
| ATM customer selects Withdrawal. | M2 | | | | |
| ATM customer enters the withdrawal amount. | M3 | | | | |
| ATM customer selects account number. | M4 | | | | |
| The system validates that the account number is valid. | M5 | | | | |
| The system validates that ATM customer has enough funds in the account. | M6 | | | | |
| The system validates that the withdrawal amount does not exceed the daily limit of the account. | M7 | | | | |
| The system validates that the ATM has enough funds. | M8 | | | | |
| The system dispenses the cash amount. | M9 | | | | |
| The system prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance. | M10 | | S1-S2 | | ATM customer funds have been withdrawn. |

| FlowId | Step | Sub FlowId | Alternate FlowId | Post Condition |
|--------|------|------------|-----------------|----------------|
| S1     | The system ejects the ATM card. | | | |
| S2     | The system displays the Welcome message. | | | |

| Specific Alternate Flow Steps |
|-------------------------------|
| A1 | The system displays an apology message. | S1 | ATM customer funds have not been withdrawn. |
| A2 | The system shuts down. | | The system is shut down. |

| Global Alternate Flow Steps |
|----------------------------|
| IF ATM customer enters Cancel | GA1 | The system cancels the transaction. | S1-S2 | ATM customer funds have not been withdrawn. |

4.1.2 Metamodels

A metamodel describes how various elements in a model are arranged, related and constrained [Bézivin 2006]. Metamodels allow the model transformation tools to effectively apply the transformation operations on the models [Siqueira and Silva 2011] (A model is an instance of a metamodel).

Our approach stores the elements of the input UCS in an instance of UCS metamodel, and the elements of output class diagram in an instance of class diagram metamodel (Figure 2). As our approach generates analysis models (analysis class diagrams) which are platform independent models so the class diagram metamodel used in our approach is different from those in the literature (including OMG’s metamodel) which contain some platform specific details required for creating design models.

4.1.3 NLP models

The approach generates two NLP constructs i) Parts of speech tags (POS-tags) and ii) Type dependencies (TDs) from the sentences in the UCS using the Stanford NL parser API. The approach then uses these NLP constructs first to identify the sentence structure of the sentences, then to extract the class diagram elements from the text.

i) Parts of speech tags (POS-tags): The POS-tag refer to the words in a sentence tagged (or annotated) with parts of speech, such as noun, pronoun, verb, adjective, adverb, etc. When a sentence is given as input to the parser, it analyses the sentence, and assigns each word in the sentence with a POS-tag taken from a set of 36 such POS-tag. The output sentence generated by the parser is same as the input sentence, but each word in the output sentence is now tagged with a POS-tag.

Example 1 If a sentence: “The bank sends the customer an sms.” is given as input to the parser then the POS tagged sentence generated by the parser as output is:

http://nlp.stanford.edu/software/tagger.shtml
http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html
“[The/DT, bank/NNP, sends/VBZ, the/DT, customer/NN, an/DT, sms/NN, ./.]”

Where, the tag /DT after a word denote that the word is determiner, /NNP denote that the word is a singular proper noun, /NN denote that the word is a singular noun, and /VBZ denote that the word is a 3rd person singular present tense verb.

ii) Type dependencies (TDs): A type dependency (TD) represents grammatical dependency relationship (bi-lexical asymmetrical relationship) between the words of a sentence. The TDs can be used to obtain the semantic relationships between the words in a sentence (De Marneffe and Manning 2008), so they are used in our approach to disambiguate the extraction of relevant elements from the sentences. Our approach uses Stanford Parser APIs version 2.0.4 to generate TDs from the sentences. The Stanford TDs (De Marneffe and Manning 2008) are based on Lexical-Functional Grammar (Bresnan 2001). The dependency relationships and the naming schemes are inherent of two representations (Carroll et al 1999, King et al 2003) that also follow the Lexical-Functional Grammar. The present Stanford typed dependencies set can identify 53 such grammatical relationships. The parser generates a TD as a triplet structure tdName(head, dependent), where tdName represents the name of the dependency, head represents the head word and dependent represents the dependent word. More formally tdName depict that the dependent word is related to the head word by the dependency tdName.

Example 2 For the sentence “The bank sends the customer an sms.”, the TDs generated by the parser are given below.

TDs = \{det(bank-2, The-1), nsubj(sends-3, bank-2), root(ROOT-0, sends-3), det(customer-5, the-4), iobj(sends-3, customer-5), det(sms-7, an-6), dobj(sends-3, sms-7)\}

7 http://nlp.stanford.edu/software/dependencies_manual.pdf
The description of the TDs are shown in Table 4. The last column in this table presents the graphical view of some important dependencies in the sentence, and how these TDs help to disambiguate the extraction of desired elements of the analysis class diagram from the sentence. The TD root(ROOT-0, sends-3) depict that the main verb in the sentence is “sends”, the subject of the verb is “bank”, so a class can be created with name “bank”. The direct object of the verb is “sms” which is receiving the action “sends” so an another class can be created with name “sms”, “sends” can be added as an operation in it, and an association relationship with name “sends” can be created between the class “bank” and the class “sms”. The indirect object of the verb is “customer” which is indirectly receiving the action “sends” so it can be added as parameter to the to the operation “sends” in the “sms” class.

Table 4 TDs generated by Stanford parser for sentence: “The bank sends the customer an sms.”

| TDName | Head   | Dependent | Type dependencies (TDs) | Depicting semantic relationship between the words in the sentence | Disambiguating extraction of relevant elements |
|--------|--------|-----------|--------------------------|---------------------------------------------------------------|-----------------------------------------------|
| det    | Bank-2 | The-1     | nsubj                   | "The” is determiner of “Bank”                                 |                                               |
| nsubj  | sends-3| Bank-2    | iobj                    | “Bank” is subject of "sends"                                  |                                               |
| root   | ROOT-0 | sends-3   | dobj                    | "sends" is root of the sentence                                |                                               |
| det    | customer-5 | the-4     | dobj                    | "the" is determiner of "customer"                            |                                               |
| iobj   | sends-3 | customer-5 |           | "customer" is indirect object of "sends"                      |                                               |
| det    | sms-7   | an-6      | det                     | "an" is determiner of "sms"                                   |                                               |
| dobj   | sends-3 | sms-7     | det                     | "sms" is direct object of "sends"                             |                                               |

4.1.4 Language model

As a same concept or thing can be expressed in many ways (using sentences with different sentence structures) in a natural language like English, the essential elements for generating analysis class diagrams are embedded at different places in different sentences. To correctly interpret the sentences for extracting the essential elements, the structures of the sentences should to be identified.

Language model: We define the language model of an approach as the set of sentence patterns that an approach can interpret.

The language model of our approach consists of a comprehensive set of sentence patterns that include all the simple sentences (including sentences containing participles, infinitives and gerunds) and a few complex sentences (sentences specifying conditions and the sentences containing that clause and conjunctive clause) written in English. A simple sentence contains only one independent clause. A complex sentence contains one independent clause and one or more dependent clauses. (A clause consists a subject or noun phrase and a predicate or verb phrase. An independent clause is a clause that do not depend on any other clause to expresses a complete thought. A dependent clause is a clause which is dependent on some other clause or dependent clause to express a complete thought. For more details on types of sentences please refer [C].

Our language model is constructed using the twenty five verb patterns proposed by A. S. Hornby, known for various achievements in linguistic and literature (Hanks [2008], in Oxford Advanced Learners Dictionary of Current English [A.S. Hornby 1974, 2000] and in Hornby [1975]). These verb patterns define the twenty five possible ways in which a verb phrase in a sentence can be written. Using all these verb patterns, we framed the comprehensive set of sentence structure rules to recognize the sentences structure of the sentences. These rules use TDs of the sentences to identify the sentence structures. The first ten sentence structure rules are presented in Table 5, for complete set of rules please refer [A]. These rules are presented in Antecedent-Consequent format and they are ordered. To identify the sentence structure of a sentence, the approach one by one checks that all TDs shown in Antecedent part of the rule are found in the TDs of the given sentence, if this is true then the approach gets the sentence structure from Consequent part of the rule. (Note: The rules SSR8-SSR11 uses POS-tags along with TDs to identify the sentence structures. The rules SSR30-SSR33 identify sentence structures through keyword matching.)

The language model along with the need to avoid ambiguities (Kamsties and Peach [2000], Wiegers and Beatty [2013]) in the sentences, enforce the UCSs to be written in English language using a few restriction rules. The restriction rules along with their rationale are presented in Table 6.

The following example shows how the approach identifies the sentence structures.
Table 5 Sentence structure rules

| Rule | Antecedent (If the sentence contains TDCs) | Consequent (then the identified sentence structure is) | Example sentence | Type dependencies of Example sentence |
|------|-----------------------------------------|-------------------------------------------------|-----------------|-------------------------------------|
| SSR1 | nsubj(A,B), dobj(A,D)                   | iobj(A,C), xcomp(C,D)                           | The system sends the user an email. | [det(system-2, The-1), nsubj(sent-sends-3, system-2), root(ROOT-0, sends-3), det(user-5, the-4), dobj(sent-sends-3, user-5), det(email-7, an-6), dobj(sent-sends-3, email-7)] |
| SSR2 | nsubj(A,B), dobj(A,C), compl(D,E), nsubj(D,F) | (Subject-Verb-IndirectObject-Object)            | The system informs the user that the battery is full. | [det(system-2, The-1), nsubj(informs-3, system-2), root(ROOT-0, informs-3), det(user-5, the-4), dobj(informs-3, user-5), compl(full-10, that-6), det(battery-8, the-7), nsubj(full-10, battery-8), cop(full-10, is-9), compl(informs-3, full-10)] |
| SSR3 | nsubj(A,B), compl(C,D), nsubj(C,E)      | (Subject-Verb-ThatClause)                       | The system validates that the password is correct. | [det(system-2, The-1), nsubj(validates-3, system-2), root(ROOT-0, validates-3), compl(correct-8, that-4), det(password-6, the-5), nsubj(correct-8, password-6), cop(correct-8, is-7), compl(validates-3, correct-8)] |
| SSR4 | nsubj(A,B), neg(D,E), aux(D,F), infmod(C,D) | (Subject-Verb-IndirectObject-Not-To-Infinite) | The system warns the user not to restart the system. | [det(system-2, The-1), nsubj(warns-3, system-2), root(ROOT-0, warns-3), det(user-5, the-4), dobj(warns-3, user-5), neg(restart-8, not-6), aux(restart-8, to-7), infmod(user-5, restart-8), det(system-10, the-9), dobj(restart-8, system-10)] |
| SSR5 | nsubj(A,B), neg(D,E), aux(D,F), dobj(C,F)  | (Subject-Verb-Not-To-Infinite)                  | The customer selects not to fill the tank. | [det(customer-2, The-1), nsubj(selects-3, customer-2), root(ROOT-0, selects-3), neg(fill-6, not-4), aux(fill-6, to-5), xcomp(selects-3, fill-6), det(tank-8, the-7), dobj(fill-6, tank-8)] |
| SSR6 | nsubj(A,B), aux(C,E), xcomp(C,A)         | (Subject-Verb-To-be-Complement)                  | The system marks the errors to be red. | [det(system-2, The-1), nsubj(marks-3, system-2), root(ROOT-0, marks-3), det(errors-5, the-4), nsubj(red-8, errors-5), aux(red-8, to-6), cop(red-8, be-7), xcomp(marks-3, red-8)] |
| SSR7 | nsubj(A,B), aux(D,E), infmod(C,D)        | (Subject-Verb-Object-To-Infinite)                | The system commands the motor to start. | [det(system-2, The-1), nsubj(commands-3, system-2), root(ROOT-0, commands-3), det(motor-5, the-4), dobj(commands-3, motor-5), aux(start-7, to-6), infmod(motor-5, start-7)] |
| SSR8 | nsubj(A,B), dobj(C,F), POS-tag(D)="VGB" | (Subject-Verb-Object-Participle)                 | The system keeps the user waiting. | [det(system-2, The-1), nsubj(keeps-5, system-2), root(ROOT-0, keeps-5), det(user-5, the-4), dobj(keeps-5, user-5), partmod(user-5, waiting-6)] |
| SSR9 | nsubj(A,B), dobj(C,F), POS-tag(D)="VBN" | (Subject-Verb-Object-Participle)                 | The system validates the record entered by the customer. | [det(system-2, The-1), nsubj(validates-3, system-2), root(ROOT-0, validates-3), det(record-5, the-4), dobj(validates-3, record-5), partmod(record-5, entered-6), prep(entered-6, by-7), det(customer-9, the-8), dobj(by-7, customer-9)] |
| SSR10 | nsubj(A,B), xcomp(C,A), POS-tag(E)="ID" | (Subject-Verb-Object-Adjective-Adjective-Complement) | The system keeps the door open. | [det(system-2, The-1), nsubj(keeps-3, system-2), root(ROOT-0, keeps-3), det(door-5, the-4), nsubj(open-6, door-5), xcomp(keeps-3, open-6)] |

Example 3 Here we show how the approach identifies the sentence structure of a given sentence using the rules presented in Table 5. For sentence say “The system commands the motor to start.”, the TDCs generated by the parser are: [det(system-2, The-1), nsubj(commands-3, system-2), root(ROOT-0, commands-3), det(motor-5, the-4), dobj(commands-3, motor-5), aux(start-7, to-6), infmod(motor-5, start-7)]

Here, for each rule shown in Table 5 the approach one by one checks whether the TDCs present in the Antecedent part of the rule are found in the TDCs of the sentence. All the TDCs in the Antecedent part of the rule SSR7 are found in the TDCs of the sentence, hence the approach gets the sentence structure as SVDOTolInf from Consequent part of the rule.

4.2 Working of the approach

The proposed approach works in five steps (Figure 5). The approach first reads the UCS, and stores the elements of the UCS into an instance of UCS metamodel that we call UCS_Instance (Step 1). It parses the sentences in the UCS_Instance using Stanford NL Parser API to generate TDCs and POS-tags (Step 2). It then applies the comprehensive set of proposed sentence structure rules on TDCs and POS-tags to identify sentence structures of the sentences (Step 3). Then it applies the proposed transformation rules on the TDCs and POS-tags to identify problem level classes, attributes, operations, the relationships between the classes (Step 4). Finally, it generates and visualizes the analysis class diagram (Step 5). The terms used in the transformation rules are shown in Table 7. The details of the steps are presented as follows:

Step 1: Read UCS and generate UCS_Instance
Automatic generation of analysis class diagrams from use case specifications

Step 1: Read UCS and store the elements of the UCS in an instance of UCS meta model called UCS_Instance

Step 2: Parse each sentence in UCS_Instance to obtain TDs and POS tags

Step 3: Identify sentence structure of each sentence from their TDs & POS-tags (using sentence structure rules SSR1-SSR33)

Step 4: Identify domain classes, their attributes and operations, and relationships between domain classes (using transformation rules TR1-TR54)

- Step 4.1: Identify entity terms (Rule TR1)
- Step 4.2: Identify control class (Rule TR2)
- Step 4.3: Identify boundary classes (Rules TR3)
- Step 4.4: Identify entity classes and their attributes (Rules TR4-TR10)
- Step 4.5: Identify class operations, class attributes (Rules TR11-TR43)
- Step 4.6: Identify more entity classes (Rule TR44-TR45)
- Step 4.7: Identify association relationships (Rule TR46)
- Step 4.8: Identify generalization relationships (Rule TR47-TR48)
- Step 4.9: Identify aggregation relationships (Rules TR51-TR53)
- Step 4.10: Eliminate extra classes (Rule TR54)

Step 5: Generate & Visualize analysis class diagram

Output – Analysis class diagram of the UCS

Fig. 3 Schematic diagram of the proposed approach
This step reads the UCS (Example Figure 3) and then stores the data from all the fields of the UCS into the respective fields in an instance of UCS metamodel (Figure 2) called UCS_Instance.

**Step 2: Parse sentences in UCS_Instance to obtain their TDs and POS tags**

This step reads the sentences of each section (Main Flow, Sub Flow, Alternate Flow, Global Alternate Flow, Pre-Condition, Post-Condition and Description section) of the UCS from UCS_Instance, and parses each sentence using Stanford NL Parser API to generate POS-tags and TDs. The TDs and POS-tags of the sentences are then used in Step 3 to identify the sentence structure of the sentences, and in Step 4 to identify the elements of the analysis class diagram.

**Example 4** For the sentence = “ATM customer enters the withdrawal amount.” of the UCS WithdrawFund (Table 3), the TDs and POS-tags generated by the are:

TDs=[nn(customer-2, ATM-1), nsubj(enters-3, customer-2), root(ROOT-0, enters-3), det(amount-6, the-4), nn(amount-6, withdrawal-5), dobj(enters-3, amount-6)]

POS-tags=[ATM/NNP, customer/NN, enters/VBZ, the/DT, withdrawal/NN, amount/NN, ./.]

**Step 3: Identify sentence structures of the sentences**

As a same concept or thing can be expressed in many ways, the essential elements for generating analysis class diagrams are embedded at different places in different sentences. To correctly interpret the sentences for extracting the essential elements, the structures of the sentences are to be identified. The approach uses the proposed set of comprehensive sentence structure rules SSR1-SSR33 to identify the sentence structures of the sentences as described in Section 4.1.4 (Table 5 presents first ten sentence structure rules, for full rule set please refer A). These rules are presented in Antecedent-Consequent format and they are ordered. To identify the sentence structure of a sentence, the approach one by one checks that all TDs shown in Antecedent part of the rule are found in the TDs of the given sentence, if this is true then the approach gets the sentence structure from Consequent part of the rule. If sentence structures of some sentences cannot be identified then those sentences are marked and presented to the user for modification. The user can modify sentences or may choose to continue without
modifying. If user chooses to continue without modifying the unidentified sentences then the unidentified sentences are skipped by the transformation process. The identified sentence structure of the sentences along with the TDs and POS-tags identified in previous step are then used in Step 4.5 to disambiguate the identification of class operations and/or class attributes. The identified operations are in turn used disambiguate the identification of entity classes in Step 4.6, and association relationships in Step 4.7.

Example 5 As described in Section 4.1.4 to identify the sentence structure of the sentence = “ATM customer enters the withdrawal amount.”, the approach applies the sentence structure rule SSR27 (shown in Appendix A) on TDs and POS-tags of the sentence (shown in Example 4) to identify the sentence structure of the sentence as SVDO.

Step 4: Identify elements of analysis class diagram

The nouns in the sentences are the prospects of classes and attributes, and the verbs are the prospects of operations and relationships. But every noun may not be a class or attribute and every verb may not be an operation or relationship. To disambiguate the process of identifying the right classes, attributes, operations and relationships of the analysis class diagram from the text, our approach uses various heuristics (the proposed transformation rules TR1-TR54) that in turn uses the syntactic and semantics relationships between the words in the sentences obtained from TDs and POS-tags. The following steps present how the approach identifies the elements of analysis class diagram.

Step 4.1: Identify entity terms

The noun phrases in the sentences may contain single nouns (Example: Transaction, Withdrawal) or a group of two or more consecutive nouns (group of nouns representing a single term, Example: Transaction number, ATM Card, ATM Card PIN number). These single nouns or group of two or more consecutive nouns are called entity terms. The entity terms are the prospects of potential classes or attributes. This step combines two or more consecutive nouns in the sentences to identify the entity terms (transformation rule-TR1). This is done by concatenating two or more consecutive words in the sentences whose POS-tag starts with “NN”. The TDs and POS-tags of the sentence are updated to reflect the changes (concatenation) done if any. A similar method for entity term extraction is used in an approach for entity disambiguation proposed by Misra and Das (2013).

Rule-TR1: For each sentence in the UCS
   Concatenate two or more consecutive words in the sentences whose POS-tags starts with “NN”.
   Update the TDs and POS-tags of the sentences to reflect the changes
   EndFor
   Scan POS-tags of each sentence, and store all the nouns in a set named setOfEntityTerms.

Example 6 For sentence “ATM customer enters the ATM Card Pin Number.”, the POS-tags generated by the parser are: [ATM/NNP, customer/NN, enters/VBZ, the/DT, ATM/NNP, Card/NNP, Pin/NNP, Number/NNP, ./] Here “ATM” and “customer” are the two consecutive words whose POS-tags start with “NN”, hence they are concatenated using the transformation rule TR1 to get “ATMcustomer” representing a single entity term. Similarly, the four consecutive words “ATM”, “Card”, “Pin” and “Number” are also concatenated using rule TR1 to get “ATMCardPinNumber” representing another single entity term.

The modified sentence is : “ATMcustomer enters the ATMCardPinNumber.”
The updated TDs are: [nsubj(enters-2, ATMcustomer-1), root(ROOT-0, enters-2), det(ATMCardPinNumber-4, the-3), dobj(enters-2, ATMCardPinNumber-4)]
The updated POS-tags are: [ATMcustomer/NNP, enters/VBZ, the/DT, ATMCardPinNumber/NN, ./] setOfEntityTerms = {“ATMcustomer”, “ATMCardPinNumber”}

Similarly, when two more sentences, “ATM customer selects Withdrawal.” and “ATM customer enters withdrawal amount.”, are processed, new entity terms are identified and the setOfEntityTerms is updated as:
setOfEntityTerms = {“ATMcustomer”, “ATMCardPinNumber”, “Withdrawal”, “withdrawalamount”}

Step 4.2: Identify control class (Rules TR2)

This step creates a control class from the name of the UCS stored in UCS_Instance (transformation rule-TR2).
**Rule-TR2**: `createClass(UCS.name,"<<control class>>")`;
For one UCS only one control class is created and from now all the references to the word “System” and to the name of UCS in the sentences refer to this control class.

*Example 7* From UCS shown in Table 3, the approach creates the control class named “WithdrawFunds <<control class>>”, where the stereotype “<<control class>>” denotes that the class is control class.

### Step 4.3: Identify boundary class(es) (Rules TR3)

This step creates a boundary class for each actor of the UCS stored in UCS_Instance (transformation rule-TR3).

**Rule-TR3**: For each actor of the UCS stored in UCS_Instance, `createClass(acceptor.name,"<<boundary class>>")`;

*Example 8* From UCS shown in Table 3, the approach creates the boundary class named “ATMcustomer <<boundary>><<primary>>", where the stereotype “<<boundary>><<primary>>” denotes that the class is boundary class of primary actor.

### Step 4.4: Identify entity classes and their attributes (Rules TR4-TR10)

The entity terms identified in Step 4.1 are the prospects of classes or attributes. But every entity term may not be a class or an attribute. The step applies heuristics (transformation rules TR4-TR10) on the identified entity terms and TDs of the sentences to identify the entity classes and attributes. Rule TR4 is given below, for rules TR5-TR10 please refer B

**Rule-TR4**: For every two entity terms t1 and t2 in `setOfEntityTerms` If (t2 startsWith t1) AND (t2 \(\neq\) t1) then

\[
\text{class = createClass(t1,"<<entity class>>")}; \text{class.addAttribute(t2)}; \\
\text{EndIf}
\]

EndFor

*Example 9* The `setOfEntityTerms` = {“ATMcustomer”, “ATMCardPinNumber”, “Withdrawal”, “withdrawalamount”}, shown in Example 6 contains two entity terms say t1 = “Withdrawal” and t2 = “withdrawalAmount”

Here the entity term t2 starts with entity term t1 hence by rule TR4 “Withdrawal” is identified as a class and “withdrawalAmount” is added as an attribute to class “Withdrawal”

### Step 4.5: Identify class operations and more class attributes (Rules TR11-TR43)

As the flow sentences (sentences in the main flow, sub flow and alternate flow sections of UCS) specifies the sequence of actions preformed by the system and the actor(s) of the system, the verbs representing these actions can be used to identify class operations. For each operation, three things are to be identified: i) the operation, ii) the source entity term that calls this operation and iii) the destination entity term that hosts this operation.

The verbs in the flow sentences are the prospects of class operations, but not all the verbs in a sentence represent valid operations. This step applies the heuristics (or transformation rules TR11-TR43) to identify the operations, the source entity terms and the destination entity terms from the flow sentences stored in UCS_Instance. These heuristics (rules) uses the sentence structure of the sentences, and the syntactic and semantic relationships between the words in the sentences depicted by TD to disambiguate the process of identifying these elements. The transformation rules TR11-TR20 are presented in Table 8 for full rule set please refer B. These rules are presented in Antecedent-Consequent format. The Antecedent part or If part (Column A) contains the sentence structure of the sentences to be matched for the rules to be fired, the Consequent part or then part has two columns (Column B and column C) where, column B presents the TDs of the sentence to be used for identifying operations/attributes and column C defines how the approach identifies class operations and attributes from these TDs when the rule is fired. To identify the class operations and attributes from a given sentence, the approach one by one compares the sentence structure of the given sentence with that present in column A, if match is found then the approach uses the TDs present in column B to identify the operations/attributes as shown in column C. These identified operations are further used in Step 4.6 for identifying more entity classes, and in Step 4.7 for identifying association relationships between the classes.

*Example 10* For sentence = “ATM customer enters the withdrawal amount.” shown in Example 4 the sentence structure as identified in Example 5 is SVDO, which that matches with the Antecedent part of rule TR37 (presented in Appendix B), hence the approach applies the transformation rule TR37 on the TDs nsubj(enters-2,ATMcustomer-1), dobj(enters-2,withdrawalAmount-4) of the sentence to identify following operation:
Table 8  Transformation rules to identify operations and attributes of classes

| Rule # | Antecedent ( If A) | Consequent ( then use B to identify C ) | C (operations/attributes:) |
|--------|---------------------|----------------------------------------|----------------------------|
| TR11   | SVIODO              | nsubj(A,B), iobj(A,C), dobj(A,D)       | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR12   | SVDOThatClause      | nsubj(A,B), dobj(A,C), complm(D,E), nsubj(D,F) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR13   | SVThatClause        | nsubj(A,B), complm(C,D), nsubj(C,E)     | op.SourceEntityTerm=B, op.DestEntityTerm=E, op.name=A |
| TR14   | SVDONotToInf        | nsubj(A,B), dobj(A,C), neg(D,E), aux(D,F), infmod(C,D) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR15   | SVNotToInf          | nsubj(A,B), neg(C,D), aux(C,E), xcomp(A,C), dobj(C,F) | op.SourceEntityTerm=B, op.DestEntityTerm=F, op.name=A |
| TR16   | SVDTobeComp         | nsubj(A,B), nsubj(C,D), aux(C,E), cop(C,F), xcomp(A,C) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR17   | SVDToInf            | nsubj(A,B), dobj(A,C), aux(D,E), infmod(C,D) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR18   | SVDOPresentPart     | nsubj(A,B), dobj(A,C), partmod(C,D) dobj(D,E) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR19   | SVDOPastPart        | nsubj(A,B), dobj(A,C), partmod(C,D) | op.DestEntityTerm.addAttribute(E) |
| TR20   | SVDODefStr          | nsubj(A,B), nsubj(C,D), xcomp(A,C) | op.DestEntityTerm= $$

Note: “op” is a meta object used to store the identified operation

op.SourceEntityTerm=“ATMcustomer”, op.DestEntityTerm=“withdrawalAmount”, op.name=“enters”

We can see that “ATM” and “customer” are consecutive nouns so they had already been concatenated to represent single entity term “ATMcustomer” using Rule TR1 in Step 4.1. Similar is the case for “withdrawal” and “amount”.

Step 4.6: Identify more entity classes (Rule TR44-TR45)

The source entity terms and destination entity terms of the operations identified in previous step are used in this step to identify more entity classes. A source entity term is the caller of the identified operation hence it is the prospect of only a class (if a class for the source entity term does not exist in ClassDiagram_Instance, a new entity class is created for it, and is added to ClassDiagram_Instance by rule TR44). Whereas, a destination entity term has two possibilities: it may either be an attribute of an existing class in which the identified operation is to be hosted or may be a prospect of a class (if a class for the destination entity term exists in ClassDiagram_Instance then the identified operation is hosted in that class, otherwise a new entity class is created, the identified operations is hosted in that class, and the class is added to ClassDiagram_Instance) (rule TR45).

**Rule-TR44**: If op.SourceEntityTerm is not present in ClassDiagram_Instance then createClass(op.SourceEntityTerm,“< <entity class>>”);

**Example 11** To the identified operation shown in Example 10

The approach apply rule TR44 to create a new entity class “ATMcustomer” and add it to ClassDiagram_Instance

| ATMcustomer |

**Rule-TR45**: For each class C in ClassDiagram_Instance

If (op.DestEntityTerm.name==C.name)AND(C does not contain operation op.name(op.Para)) then

C.addOperation(op.name(op.Para));

EndIf

EndFor
If no such class is found then
   For each class C in ClassDiagram_Instance
      If(op.DestEntityTerm.name==a.name for some attribute a of class C)AND
         (C does not contains operation op.name(op.Para)) then
            C.addOperation(op.name(op.Para));
            EndIf
      EndFor
   EndIf
If no such class is found then
   C=createClass(op.DestEntityTerm.name,"<<entity class>>"); C.addOperation(op.name(op.Para));
   EndIf

Example 12  To the same operation identified in Example 10
   the approach applies rule TR45 to add the operation “enters(withdrawalAmount)” to
class="Withdrawal", since op.DestEntityTerm="withdrawalAmount" is found as an attribute of
class="Withdrawal"

Step 4.7: Identify association relationships (Rule TR46)

From the source entity term, the destination entity term and the operation name of each operation identified in step 4.5,
the approach identifies association relationship between the class representing source entity term and the class representing
destination entity term. The navigability of the relationship is set from the class representing the source entity term to the
class representing the destination entity term. (transformation rule TR-46)

   For two classes say class A and class B, if class A calls operations op1 of class B then an association relationship
   named op1 is created between the class A and Class B. If class A also calls another operation op2 of class B then op2 is
   appended to the name of the association relationship to get the new name of the association relationship as op1op2 (Note: a
   new relationship op2 is not created between class A and class B, because this will result in redundant relationships between
   the classes i.e. two association relationship of navigability from class A to class B). The appending of such operation name
   to association relationship names will be helpful during the modification that may be done automatically or manually at later
   stage. Suppose at later stage we remove the operation op1 from class B, then this will require to remove the relationship
   named op1 between class A and class B, this can be done easily by removing the op1 from the name of association relationship
   between class A and class B to get the new relationship name as op2. If the relationships had not been renamed this
   way then the removal of the operation op1 from class B will also remove the only association relationship with name op1
   between the two classes.

   Rule-TR46: For each relationship r in ClassDiagram_Instance
   If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==r.class2)AND(r.name does not contains op.name)
      append op.name to r.name
   EndIf
   EndFor
   If (no such relationship found) then
      For each class c in ClassDiagram_Instance
         If(op.DestEntityTerm==c.Name)
            rName=op.name; createRelationship(op.SourceEntityTerm, c, rName, “association”);
         EndIf
      EndFor
      If(no such class is found) then
         For each class c in ClassDiagram_Instance
            If(op.DestEntityTerm==a.Name for some attribute a in class c)
               rName=op.name; createRelationship(op.SourceEntityTerm, c, rName, “association”);
            EndIf
         EndFor
         EndIf
      EndIf
   EndIf
   (Note: the operation createRelationship(), sets the navigability from the class representing the sourceEntityTerm to the class
   representing destEntityTerm when it creates association relationship between the classes.)
Example 13  To the operation op = “enters”, op.sourceEntityTerm = “ATMcustomer” and op.destEntityTerm = “withdrawalAmount” shown in Example [10] of Step 4.5, the approach applies rule TR46 to create an association relationship r with r.name = ‘enters”, r.endClass1 = “ATMcustomer”, r.endClass2 = “Withdrawal” (since op.DestEntityTerm = “withdrawalAmount” is found as an attribute of class = “Withdrawal”) and with navigability from “ATMcustomer” to “Withdrawal”

Step 4.8: Identify generalization relationships (Rules TR47-TR50)

The sentences in flows and description sections of UCS are scanned and the sentences containing sub strings of types “is a”, “kind of”, and all their synonyms are used to identify generalization relationships. We call such sub strings as Generalization Sub String (GenSubString). We categorise the sentences containing GenSubString into two kinds:

1. Child-GenSubString-Parent sentences: The sentences containing sub strings “is a”, “type of”, “kind of”, and all their synonyms are referred as Child-GenSubString-Parent sentences because in these sentences the child class/classes is/are present on the left of GenSubString and the parent class is present on the right of GenSubString. Rule TR47 is used to identify generalization relationships from these sentences.

2. Parent-GenSubString-Child sentences: The sentences containing sub strings “parent of”, “categorized into”, “has types”, “of types”, “classified into”, “classified as” and all their synonyms are referred as Parent-GenSubString-Child sentences because in these sentences the parent class is present on the left of GenSubString and the child class/classes is/are present on the right of GenSubString. Rule TR48 presented in Appendix B is used to identify generalization relationships from these sentences.

Rule-TR47: For each sentence of type Child-GenSubString-Parent, the POS-tags of the sentence are scanned and parentClass=createClass(noun nr on the right of GenSubString,”<entity class>”);
   For each noun nl on the left of GenSubString
       childClass=createClass(nl,”<entity class>”);
   createRelationship(parentClass,childClass,”generalization”);
   EndFor
   EndFor

Example 14  For sentence “The withdrawal, deposit, transfer and query are types of transaction.” POS-tags generated by the parser are:

[The/DT, withdrawal/NN, ,/, deposit/NN, ,/, transfer/NN, and/CC, query/NN, are/VBP, types/NNS, of/IN, transaction/NNS, ,/]

As this sentence contains GenSubString=“types of” hence it is Child-GenSubString-Parent sentence, therefore rule TR47 is applied. From POS-tags the nouns to the left of GenSubString “types of” are withdrawal, deposit, transfer and query, a child class is created for each of these nouns. And the noun to the right of GenSubString “types of” is transaction, a parent class is created for this noun. Generalization relationship is established between the identified child classes (withdrawal, deposit, transfer and query) and the identified parent class (transaction)

Step 4.9: Identify aggregation relationships (Rules TR51-TR53)

An aggregation relationship is a part-whole relationship between two classes, in which the part class is contained in the whole class (or part class is present as an attribute inside the whole class). The approach identifies aggregation relationships using two tactics: 1) When in ClassDiagramInstanceOf, a class say C1 is present as an attribute in another class say C2, then class C1 is recognized as part class and C2 is recognized as whole class, and aggregation relationship is created between the part class and whole class (transformation rule TR51 shown below). 2) The sentences in flows and description sections of
UCS containing sub strings such as “part of”, “consists of”, “contains” and all their synonyms are the prospects of aggregation relationships. The transformation rules TR52-TR53 (given is Appendix B) identify aggregation relationships from such sentences.

**Rule-TR51**: For each of the two classes c1 and c2 in ClassDiagram_Instance

```plaintext
If c2 is attribute of c1 then
  wholeClass=c1; partClass=c2;
  createRelationship(wholeClass,partClass,"aggregation");
EndIf
EndFor
```

*Example 15* Let ClassDiagram_Instance contains two classes say c1="Book" and c2="BookDetail" then by rule TR51, the approach creates an aggregation relationship between wholeClass="Book" and the partClass="BookDetail"

**Step 4.10: Eliminate extra classes (Rule TR54)**

Every noun or entity term may not be a class, if a noun or an entity term is identified as a class by the approach, then it must participate in relationship with some other class (or classes) in the class diagram. Hence, if some classes are not participating in any relationships then those classes are extra classes and needs to be removed from the class diagram. These extra classes are removed by applying rule TR54.

**Rule-TR54**: For each class c in ClassDiagram_Instance, if c is not present as EndClass of any relationship r in ClassDiagram_Instance then delete c from ClassDiagram_Instance.

**Step 5: Generate and visualize the class diagram**

The identified classes and relationships are then used to generate DOT language commands (commands that are used to create and visualize various graph elements such as lines, rectangles, triangles etc. in GraphViz). The class diagram is then visualized using GraphViz API.

**5 Tool support developed**

We have implemented the proposed approach in a prototype tool named Automatic Analysis Model Generator (AutoAMG) in Java 1.6 using Eclipse Indigo IDE release 3.7.0. AutoAMG uses Stanford NLP parser APIs version 2.0.4 for parsing the sentences. It uses Apache POI API 3.14 for reading the UCSs written in MS Excel files. For in-place visualizing of the generated class diagrams it uses Graphviz. It also uses a Java API to call dot (GraphViz) from Java program. The current implementation supports the input of one UCS at a time, and the generation of analysis class diagram for one UCS at a time. The interactions of the given UCS with other UCSs, as specified in the given UCS using INCLUDE, EXTEND keywords and Parent Use Case Name field of the UCS, are shown in the generated analysis class diagram with the help of INCLUDE, EXTEND and generalization relationships between the control class of given UCS and the control classes of the other UCSs respectively. The tool can be easily extended to take a set of UCS for a given problem as input, and to generate the analysis class diagram for the problem.

**User interface provided by AutoAMG**: AutoAMG provides a GUI interface, the menu bar provides options to:

1. Browse and select a UCS file for the input: Once the UCS file is selected it is read and parsed by the tool. The sentence structures of the sentences in UCS are identified and the UCS is displayed in a tab at center of screen. The sentences whose sentence structures cannot be identified are marked by the tool. The user can either modify those sentences or can continue by selecting Generate and Visualize Analysis Class Diagram from the menu bar.

---

8 [http://nlp.stanford.edu/software/lex-parser.shtml](http://nlp.stanford.edu/software/lex-parser.shtml)
9 [http://poi.apache.org/](http://poi.apache.org/)
10 [http://www.graphviz.org/](http://www.graphviz.org/)
11 [https://github.com/jabbalaci/graphviz-java-api](https://github.com/jabbalaci/graphviz-java-api)
2. Generate and visualize analysis class diagram: When user selects this options the tool automatically generates the analysis class diagram and displays it in a new tab at the center of screen.

3. Export class diagram as JPEG image: The user can browse to select location for storing the class diagram image file.

4. Export class diagram as XMI file: The user can browse to select location for storing the class diagram XMI file. The XMI file can be imported in an open source tool (e.g. ArgoUML[12]) for modifying the class diagram if required.

6 Experimental study

This section reports the controlled experiment that we conducted for comparing the analysis class diagrams generated by the proposed approach with those generated by the two existing approaches, one proposed by Popescu et al (2008) and the other proposed by Yue et al (2013a, 2015). In the experiment the analysis class diagrams generated by the three approaches for forty UCSs were evaluated by forty subjects on the basis of correctness, completeness and redundancy of the analysis class diagrams generated by them. The procedure followed for conducting the experiment were based the guidelines for experimental studies presented in (Wohlin et al 2003; Sjøberg et al 2005; Wohlin et al 2012)

Here, we present the details of the experiment. Section 6.1 presents the goal of the experiment and the research question. Section 6.2 presents the definition of the metrics used for assessing the correctness, completeness and redundancy of the analysis class diagrams. Section 6.3 states the hypothesis based on the research questions. Section 6.4 describes the variables used. Section 6.5 presents subject and object selection. Section 6.6 introduces the experimental design. Section 6.8 presents the study results and describes the analysis procedure. Section 6.9 discusses the validity considerations.

6.1 Goal and research question

The objective of the experiment was to compare the quality (correctness, completeness and redundancy) of the analysis class diagrams generated by the three approaches viz. the proposed approach, Popescu et al (2008) and Yue et al (2013a, 2015). Hence we framed following research question:

- RQ: Are the three approaches differ significantly in terms of the quality (class diagram correctness (CD_{cr}), class diagram completeness (CD_{cm}) and class diagram redundancy (CD_{rd})) of the analysis class diagrams generated by them? If yes, then what are the significant differences between them in terms of the quality?

6.2 Metrics used to assess the quality

The metrics used for assessing the quality of the analysis class diagrams along with how to use them for the assessment are as follows:

1. Class diagram correctness (CD_{cr}): Class diagram correctness of an analysis class diagram is found in terms of Average class correctness (AvgC_{cr}) and Average relationship correctness (AvgR_{cr}). The Average class correctness is the average of the Class correctness (C_{cr}) of all the classes in the class diagram. The Average relationship correctness is the average of the Relationship correctness (R_{cr}) of all the relationships in the class diagram.

   \[ CD_{cr} = \left( \frac{\text{AvgC}_{cr} + \text{AvgR}_{cr}}{2} \right) \]

   - Class correctness (C_{cr}): is the proportion of the correctness of various class elements like class name, class stereotype, class attributes and class operations.

   \[ C_{cr} = \left( C_{cr1} + C_{cr2} + C_{cr3} + C_{cr4} + C_{cr5} \right) / 5 \]

   So to calculate the correctness of a class, we need to find the values of variables C_{cr1}, C_{cr2}, C_{cr3}, C_{cr4} and C_{cr5}, which are found as follows:

   - Correctly identified as class (C_{cr1}) = 1, if the identified class represents some significant concept or thing of the problem domain for which a separate class is required in the analysis class diagram, 0 otherwise.

   - Correctly named (C_{cr2}) = 1, if the name assigned to the class is correct, 0 otherwise.

   - Correctly stereotyped (C_{cr3}) = 1, if a correct stereotype (<<entity>>, <<boundary>> or <<control>>) is assigned to the class, 0 otherwise.

   - Proportion of correctly identified attributes (C_{cr4}) = No. of correctly identified attributes in the class / Total no. of identified attributes in the class, if the Total no. of identified attributes in the class >0, 0 otherwise.

[12] http://argouml.tigris.org/
2. Class diagram completeness (CD_cm): Class diagram completeness of an analysis class diagram is measured as the average of Class operation completeness (CO_cm) and Relationship completeness (R_cm).
\[ CD_{cm} = \frac{(CO_{cm} + R_{cm})}{2} \]
- Class operation completeness (CO_cm): Class operation completeness is measured in terms of the No. of sentences in the functional requirements whose functionalities are assigned as operations to some class (or classes) in the class diagram (N_{sf}).
\[ CO_{cm} = \frac{N_{sf}}{N_s}, \text{where } N_s \text{ is the total number of sentences in the functional requirements} \]
- Relationship completeness (R_cm): Relationship completeness is measured in terms of the No. of separate groups of classes and relationships in the class diagram (N_{rg}).
\[ R_{cm} = \frac{1}{(N_{rg}-1)/N_f}, \text{if } N_f > 0, 0 \text{ otherwise (where } N_f \text{ is the total number of classes in the class diagram and } N_r \text{ is the total number of relationships in the class diagram)} \]

3. Class diagram redundancy (CD_rd): Class diagram redundancy of an analysis class diagram is measured as the average of Class redundancy (C_rd) and Relationship redundancy (R_rd).
\[ CD_{rd} = \frac{(C_{rd} + R_{rd})}{2} \]
- Class redundancy (C_rd): Class redundancy is found in terms of No. of redundant or extra classes in the class diagram (N_{rc}). A class is considered as a redundant class if it does not participate in any relationship with other classes in the class diagram or if it is an incorrectly identified class. Redundant classes are extra classes that are identified by an approach (those classes that are not needed). The extra classes can be of two types: i) the class that do not participate in any relationship to other class/classes in the class diagram. ii) An incorrect class (a class which was identified by an approach due to some mistake) is also an extra class (this class should not be identified by the approach hence it is extra or redundant class)
\[ C_{rd} = N_{rc} / N_c, \text{if } N_c > 0, 0 \text{ otherwise (where, } N_c = \text{Total no. of classes in the class diagram)} \]
- Relationship redundancy (R_rd): Relationship redundancy is found in terms of No. of redundant or extra relationships in the class diagram (N_{rr}). More than one relationships of same relationship type and navigability between the two same end classes are considered as redundant relationships. An incorrect relationship is also considered as redundant relationship because it is identified by the approach my mistake so it is also extra or redundant relationship.
\[ R_{rd} = N_{rr} / N_r, \text{if } N_r > 0, 0 \text{ otherwise (where, } N_r = \text{Total no. of relationships in the class diagram)} \]

6.3 Hypothesis formulation

We hypothesize that there are differences between the quality (correctness, completeness and redundancy) of the analysis class diagrams generated by the three approaches (Popescu et al. approach, Yue et al. approach and Our approach)(Table 9).
Table 9 Hypothesis formulation

| Hypothesis       | Null hypothesis                                                                 | Alternate hypothesis                                                                 |
|------------------|---------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Hypothesis 1     | H1-0: There are no significant differences between the correctness of the analysis class diagrams generated by the three approaches. | H1-1: The three approaches differ significantly in terms of the correctness of the analysis class diagrams generated by them. |
|                  | \( CD_{cr}(\text{Popescu et al.}) = CD_{cr}(\text{Yue et al.}) = CD_{cr}(\text{Our approach}) \) |                                                                                       |
| Hypothesis 2     | H2-0: There are no significant differences between the completeness of the analysis class diagrams generated by the three approaches. | H2-1: The three approaches differ significantly in terms of the completeness of the analysis class diagrams generated by them. |
|                  | \( CD_{cm}(\text{Popescu et al.}) = CD_{cm}(\text{Yue et al.}) = CD_{cm}(\text{Our approach}) \) |                                                                                       |
| Hypothesis 3     | H3-0: There are no significant differences between the redundancy in the analysis class diagrams generated by the three approaches. | H3-1: The three approaches differ significantly in terms of redundancy of the analysis class diagrams generated by them. |
|                  | \( CD_{rd}(\text{Popescu et al.}) = CD_{rd}(\text{Yue et al.}) = CD_{rd}(\text{Our approach}) \) |                                                                                       |

6.4 Variables

Following are the independent and the dependent variables of the experimental study

1. Independent variables: The analysis class diagrams generated using the three approaches.
2. Dependent variables: Class diagram correctness \( (CD_{cr}) \), Class diagram completeness \( (CD_{cm}) \) and Class diagram redundancy \( (CD_{rd}) \) of the analysis class diagrams generated by the three approaches.

6.5 Subject and object selection

**Subject selection:** The subjects (or participants) in the experiment were the forty students of Computer Science and Engineering discipline from Indian Institute of Information Technology Design & Manufacturing Jabalpur, India. The subjects were a mix of Ph.D. scholars, M.Tech. final year students and B.Tech final year students. All the subjects had done a basic course in Software Engineering as well as either Object-Oriented Software Engineering course or Object-Oriented Analysis and Design course, both of which required a course project which needed them to develop object-oriented analysis and design models for some specific problems. So the subjects were already familiar with object oriented analysis and modeling. Additionally, we conducted a training session of two hours to brush up their analysis modeling concepts. The training was not biased to any approach evaluated in the experimental study. The training given to the subject was very general in which the concepts for identifying the objects, their attributes and operations, and the relationships between the objects were based on the heuristics proposed by Abbott and Grady Booch et al. The concepts and understanding of the subjects were then tested through two different exercises each of two hours. In first exercise they were given a UCS and were asked to create an analysis class diagram for the UCS. In the second exercise they were given a UCS with its corresponding analysis class diagram, and were told to fill the same set of questionnaire for quality measures of analysis class diagrams, as used in the experiment. The answers to the questionnaires were checked and the mistakes done by the students were told to them, so that they don’t make the same mistakes in the experiment.

**Object selection:** The objects were the analysis class diagrams generated by the three approaches viz. \( \text{Popescu et al.}(2008), \text{Yue et al.}(2013a, 2015) \) and our approach for forty UCSs. We took these forty UCSs from various software engineering books \( \text{Grady Booch 2010, O’Docherty 2005, Gomaa 2011, Rosenberg and Stephens 2008, 2007, Bruegge and Dutoit 1999} \) and research works \( \text{Lu et al. 2004, Popescu et al. 2008, Yue et al. 2013a, 2015, Deeptimahanti and Sanyal 2011} \). Figure 4 presents the analysis class diagrams which were generated using the three approaches for one of the UCS named Withdraw Fund shown in Table 3.

6.6 Experimental design

The experiment design was a complete block design in which each subject evaluated the analysis class diagrams of all the three approaches for a UCS provided to him/her. In all there were forty subjects and forty UCSs, each subject was given one UCS, the class diagrams obtained from the three approaches for the given UCS and a set of questionnaires with each class diagram. These experimental materials were randomly distributed to the subjects. The blocking variable was the order in which the questionnaire for the approaches to be answered, so to nullify the ordering effect we asked the subjects to answer the questionnaires for the three approaches in the orders presented in Table 10. In the experiment the subjects first read the UCS, then examining the analysis class diagrams with respect to the UCS answered to the given questionnaires regarding the correctness, completeness and redundancy of the analysis class diagrams. The questionnaires were based on the quality
Fig. 4 Analysis class diagrams generated by Popescu et al. approach, Yue et al. approach and our approach for the UCS Withdraw Fund measures for evaluating correctness, completeness and redundancy of analysis class diagrams presented in Section 6.2.

Table 10 Order of the evaluation for the class diagrams of the three approaches to nullify the effect of blocking variable - Ordering effect

| Order of evaluation | Number of students | First       | Second       | Third       |
|---------------------|--------------------|-------------|--------------|-------------|
| 1 to 7              | 7                  | Popescu et al. | Yue et al.  | Our approach |
| 8 to 14             | 7                  | Popescu et al. | Our approach | Yue et al.  |
| 15 to 21            | 7                  | Yue et al.  | Popescu et al. | Our approach |
| 21 to 28            | 7                  | Yue et al.  | Our approach | Popescu et al. |
| 29 to 34            | 6                  | Our approach | Popescu et al. | Yue et al.  |
| 35 to 40            | 6                  | Our approach | Yue et al.  | Popescu et al. |

Instrumentation/Experimental-material: The experiment material provided to each subject was:
1. A use case specification (UCS). (A sample UCS is shown in Table 3)
2. The analysis class diagrams generated by the three approaches for the given UCS. (The samples of analysis class diagrams generated by the three approaches for the UCS Withdraw Fund is shown in Figure 4)
3. Questionnaires for evaluating correctness, completeness and redundancy of analysis class diagrams of each approach. The questionnaires were based on the class diagram quality measures presented in Section 6.2. The sample questionnaires for
correctness, completeness and redundancy of the analysis class diagram generated by one of the approach for UCS Withdraw Fund are given in Appendix ??.

6.7 Execution of the experiment

The experimental material (a UCS, three analysis class diagrams and three set of questionnaires one for each class diagram) were randomly distributed to the subjects. The subjects were clearly explained the steps for reading the UCS, examining the analysis class diagrams and filling the answers to the questionnaires. We provided the subjects maximum of three hours time for completing the experiment (In a prior mock experiment with four UCSs of different sizes taken from the 40 UCs used in the experimental study, we found that the time needed to answer the questionnaires was 1 hour 20 minutes for the UCS of smallest size and 2 hours 45 minutes for the UCS of largest size. So we decided the maximum time of 3 hours for the experiment.). Each subject had to answer the questionnaires for all the three analysis class diagrams provided to him/her. In the experiment each subject first read the UCS provided to him/her, then following the order shown in Table 10 examined the analysis class diagrams and filled the answers to the given questionnaires. The subjects were not allowed to talk and share the answers with each other during the experiment. As soon as a subject completed his/her experiment, the answers to questionnaire were collected from him/her, and the subject is allowed to move out.

6.8 Results & Analysis

This section analyzes the data set obtained from the experiment. Table 11 shows the data set for the correctness, completeness and redundancy of the analysis class diagrams of the three approaches for each UCS. This data set is obtained from the answers to the questionnaires collected from the subjects. The variability of the data for the correctness, completeness and redundancy of the analysis class diagrams generated by the approaches for the forty UCSs are shown using box plots in Figure 5.

![Box plot for the correctness, completeness and redundancy of the analysis class diagrams - showing the IQR, whiskers (1.5*IQR), and max/min outliers.](image)

**Fig. 5** Box plot for the correctness, completeness and redundancy of the analysis class diagrams - showing the IQR, whiskers (1.5*IQR), and max/min outliers.

**Hypothesis Testing:** To choose the right statistical test for testing experimental hypotheses we applied Kolmogorov-Smirnov test (Massey Jr 1951) to check the normality of the obtained data, the results of the test give no evidence for the data to be normal. Hence we decided to apply a non parametric test for testing experimental hypotheses, we chose to apply Friedman test as our experimental design, procedure applied for the execution of experiment and the data collection methods fulfilled all the conditions for applying Friedman test on the data (Corder and Foreman 2009; Sheskin 2003).

**Testing of hypotheses 1, 2 and 3:** We applied the Friedman test on the data obtained from the experiment for correctness, completeness and redundancy of the analysis class diagrams of the three approaches. The results of the test as shown in Table 12 show that the significant value (or p-value) for class diagram correctness ($CD_{cr}$), class diagram completeness ($CD_{cm}$) and class diagram redundancy ($CD_{rd}$) of the analysis class diagrams of the three approaches obtained from the test are 0.000, 0.000 and 0.000, respectively which are all less than the 0.01 ($p < 0.01$), which gives clear evidence (with a confidence level of 99%) for the rejection of the null hypotheses H01, H02 and H03, respectively. Hence the analysis class diagrams of the three approaches significantly differ in terms of correctness (alternate hypothesis H11 hold), significantly
differ in terms of completeness (alternate hypothesis H12 hold) and significantly differ in terms of redundancy (alternate hypothesis H13 hold).

To find which approaches are significantly better than the other approaches in terms of correctness, completeness and redundancy of their analysis class diagrams, we applied the post hoc Friedman tests for pair wise comparison of the three approaches. The results of the post hoc tests for correctness, completeness and redundancy are shown in Tables 13, 14 and 15 respectively. The results showed that there is no significant difference between the correctness of analysis class diagrams of Popescu et al. (2008) and Yue et al. (2013a, 2015) approaches, whereas the correctness of the analysis class diagrams of our approach were more than those of Popescu et al. approach and Yue et al. approach. The completeness of analysis class diagrams generated by Yue et al. approach were more than those by Popescu et al. approach and Yue et al. approach. There was no significant difference between the redundancy in analysis class diagrams generated by Popescu et al. approach and Yue et al. approach, whereas the redundancy in the analysis class diagrams generated by our approach were less than those by Popescu et al. approach and Yue et al. approach. Hence the results clearly showed that for the forty UCSSs the analysis class diagrams generated by our approach were more correct, more complete and less redundant than those generated by the other two approaches. Specifically, from Table 11 one can note that the analysis class diagrams generated by the proposed approach were 91% correct, 98% complete and 0% redundant, whereas those generated by Yue et al. approach were 58% correct, 67% complete and 32% redundant, and those generated by Popescu et al. approach were 45% correct, 43% complete and 32% redundant.
for the UCS given to him/her, hence the threat to selection bias for the comparison groups was nullified. The three approaches

exercises. training sessions to brush up their analysis modeling concepts; and tested their concepts and understanding through some

Object Oriented Software Engineering

the subjects have studied at least two courses in object oriented concepts and analysis modeling. The subjects (or participants) in the experiment were the forty students of Computer Science and Engineering discipline from Indian Institute of Information Technology Design & Manufacturing Jabalpur, India. The subjects were a mix of Ph.D. scholars, M.Tech. final year students and B.Tech final year students. All of whom were selected through a voluntary participation procedure. The sample size was forty as it was sufficient to reveal the significant differences in the generated analysis class diagrams. The selection of the subjects was purely random.

Here we present the threats to validity in our experimental study and discuss the strategies used to deal with those threats (Wohlin et al. 2003; Sjøberg et al. 2005; Wohlin et al. 2012)

Table 12 The Friedman test for the class diagram completeness (CD_com) and redundancy (CD_red) of the three approaches

| Response variable | Null hypothesis | N  | $\chi^2$  | p value | Test Result | Conclusion |
|-------------------|-----------------|----|---------|--------|-------------|------------|
| CD_correctness    | H01: There are no significant differences between the correctness of the analysis class diagrams generated by the three approaches. | 40 | 60.843  | .000   | p < 0.01, Enough evidence to reject null hypothesis | The correctness of the analysis class diagram generated by the three approaches differ significantly |
| CD_com            | H02: There are no significant differences between the completeness of the analysis class diagrams generated by the three approaches. | 40 | 72.731  | .000   | p < 0.01, Enough evidence to reject null hypothesis | The completeness of the analysis class diagram generated by the three approaches differ significantly |
| CD_red            | H03: There are no significant differences between the redundancy in the analysis class diagrams generated by the three approaches. | 40 | 56.219  | .000   | p < 0.01, Enough evidence to reject null hypothesis | The redundancy of the analysis class diagram generated by the three approaches differ significantly |

Table 13 The post hoc Friedman test for the pairwise comparison of class diagram completeness (CD_com) of the three approaches

| Pairwise comparison (A vs B) | N  | Test Statistics | Mean Rank | Test Result | Conclusion |
|-----------------------------|----|-----------------|-----------|-------------|------------|
| Popescu et al. vs Yue et al. | 40 | 3.600 .058 | 1.35 1.65 | p > 0.01, No enough evidence for statistical difference | No statistical significant difference between the completeness of the class diagram (CD_com) of the two approaches |
| Popescu et al. vs Our approach | 40 | 40.000 .000 | 1.00 2.00 | p < 0.01, Enough evidence for statistical difference | mean rank for our approach > mean rank for Popescu et al. generated by the three approaches |
| Yue et al. vs Our approach | 40 | 39.000 .000 | 1.01 1.99 | p < 0.01, Enough evidence for statistical difference | mean rank for our approach > mean rank for Yue et al. generated by the three approaches |

Table 14 The post hoc Friedman test for the pairwise comparison of class diagram completeness (CD_com) of the three approaches

| Pairwise comparison (A vs B) | N  | Test Statistics | Mean Rank | Test Result | Conclusion |
|-----------------------------|----|-----------------|-----------|-------------|------------|
| Popescu et al. vs Yue et al. | 40 | 25.973 .000 | 1.11 1.89 | p < 0.01, Enough evidence for statistical difference | mean rank for Yue et al. > mean rank for Popescu et al, hence CD_com(Popescu et al.) > CD_com(Yue et al.) |
| Popescu et al. vs Our approach | 40 | 40.000 .000 | 1.00 2.00 | p < 0.01, Enough evidence for statistical difference | mean rank for our approach > mean rank for Popescu et al., hence CD_com(Our approach) > CD_com(Popescu et al.) |
| Yue et al. vs Our approach | 40 | 39.000 .000 | 1.01 1.99 | p < 0.01, Enough evidence for statistical difference | mean rank for our approach > mean rank for Yue et al., hence CD_com(Our approach) > CD_com(Yue et al.) |

6.9 Validity consideration

Here we present the threats to validity in our experimental study and discuss the strategies used to deal with those threats (Wohlin et al. 2003; Sjøberg et al. 2005; Wohlin et al. 2012)

Internal validity: One of the threats to internal validity was: the knowledge, understanding and experience of the subjects in object oriented concepts and analysis modeling. The subjects (or participants) in the experiment were the forty students of Computer Science and Engineering discipline from Indian Institute of Information Technology Design & Manufacturing Jabalpur, India. The subjects were a mix of Ph.D. scholars, M.Tech. final year students and B.Tech final year students. All the subjects have studied at least two courses in Object Oriented Software Engineering, one of which is Object Oriented Analysis and Design, they have also done a course project involving object oriented analysis. Additionally, we organised training sessions to brush up their analysis modeling concepts; and tested their concepts and understanding through some exercises.

As the design of the experiment was a complete block design so each subject had evaluated all the three approaches for the UCS given to him/her, hence the threat to selection bias for the comparison groups was nullified. The three approaches
were compared on the basis of the analysis class diagrams generated by them for the same set of forty UCSs and using the same quality measures. Hence there was no comparison bias.

The blocking variable was the order in which the approaches were evaluated by the subjects, so to nullify the ordering affect we divided the subjects into six groups and the subjects were asked to answer the questionnaires for the three approaches in the order shown in Table 10.

**Construct validity:** The threats to construct validity was: Whether the measures used for the evaluation of the class diagrams generated by the tools are able to depict the quality of the class diagrams? We minimized this threat by using the quality metrics presented in Section 6.2 for estimating the quality of analysis class diagrams generated by the tools. These quality metrics are based on a systematic literature review on model quality (Mohagheghi et al. 2009) and on the quality measures which were previously defined and used for estimating the correctness, completeness and redundancy of the class diagrams in Yue et al. (2013b). The quality measures presented in Yue et al. (2013b) were based on comparing an analysis class diagram with some reference class diagram, but it is difficult to get the reference class diagrams for every UCS. Moreover, different persons (or experts) may derive different analysis class diagram for the same UCS and all of them may be correct, so it is difficult to get a single reference class diagram for a given UCS. Therefore we modified and re-formulated those quality measures, so that the measures can be used to assess the quality of analysis class diagrams even if the reference class diagrams are not available. We present 15 sub metrics used for estimating the quality (correctness, completeness and redundancy) of the analysis class diagrams. Out of these 15 sub metrics, 13 are newly defined by us and only 2 sub metrics (correctly named and correctly stereotyped for class correctness) are taken from Yue et al. (2013b). These modifications were done based on the model quality goals presented in Mohagheghi et al. (2009) and the methodology a human analyst uses to identify problem level objects and relationships from the vocabulary of the problem domain (Booch 2006).

**Conclusion validity:** We minimized the threats to statistical conclusion validity by applying the suitable statistical tests on the data. First we check the normality of the data obtained from the experiment by applying Kolmogorov-Smirnov test, and found that the data was not normal. Then we applied Friedman test (a non parametric test) for hypothesis testing as our experimental design, procedure applied for the execution of experiment and the data collection methods fulfilled all the conditions for applying Friedman test on the data. We tested all hypotheses considering the significance level of 0.01 (p <0.01). Hence, we are confident that the correct statistical tests were applied, as the assumptions of the statistical tests were not violated.

**External validity:** The external validity of an experiment is related to the ability to generalize the results. In the reported experiment the students were used as subjects for comparing the approaches. One of the threats to external validity is whether the results obtained through the answers from the students generalize to software professionals. The subjects of the experiment were well trained in object oriented analysis and modeling through the courses in object oriented software engineering and the course projects done involving analysis and design. Additionally, we conducted training sessions to brush up their analysis modeling concepts. Their concepts and understanding were then tested through different exercises involving analysis modeling. Studies like Host et al. 2000, Holt et al. 1987, Arisholm and Sjoberg 2004 reported no significant difference between the students and software professionals used as subjects in the experiments. Porter et al. in their work Porter and Votta 1998 reported identical outcomes for the two experiments conducted by them, one involving students as subjects and the other replicated experiment that involved software professionals as subjects. Moreover, studies that involves object oriented design and modeling with UML like Arisholm et al. 2006 suggested that the students are better trained representatives than most professionals, specifically those who have not been taught OO modeling with UML in detail. A survey of

| Pairwise comparison (A vs B) | N | Test Statistics | Mean Rank | Test Result Conclusion | Conclusion valid |
|-----------------------------|---|----------------|-----------|------------------------|-----------------|
| Popescu et al. vs Yue et al. | 40 | .231 | .631 | 1.46 | 1.54 | p>0.01, No enough evidence for statistical difference | No statistical significant difference between the Class diagram redundancy (CDrd) of the two approaches |
| Popescu et al. vs Our approach | 40 | 36.000 | .000 | 1.95 | 1.05 | p<0.01, Enough evidence for statistical difference | mean rank for our approach < mean rank for Popescu et al., hence CDrd (Our approach) <CDrd (Popescu et al.) |
| Yue et al. vs Our approach | 40 | 40.000 | .000 | 2.00 | 1.00 | p<0.01, Enough evidence for statistical difference | mean rank for our approach < mean rank for Yue et al., hence CDrd (Our approach) <CDrd (Yue et al.) |
controlled experiment in software engineering (Sjøberg et al. 2005) reported that 81% percent of the subjects were students in a total of 113 experiments investigated in the study.

Other threats to external validity in the experiments like ours’ are the size of the study and the number of the representative of the population. In our study we used the forty UCSs from various domains such as control system, real time system, web applications etc. All these UCSs were from the standard software engineering books such as Grady Booch (2010), Bruegge and Dutoit (1999), Gomaa (2011), Rosenberg et al. (Rosenberg and Stephens, 2008, 2007) etc. and research works (Liu et al. (2004); Popescu et al. (2008); Yue et al. (2013a, 2015); Deeptimahanti and Sanyal (2011). Although, these UCSs were not the true representative of the industry level UCS, but are fairly good. Moreover, the same UCSs were used in all the three treatments so their effect if any would had been be same in the evaluation of all the approaches. Regarding the number of representative of the population, in the same survey (Sjøberg et al, 2005) as reported above in the total of 113 experiments investigated the number of subjects per experiment ranged from 04 to 266, with a mean value of 48.6. Hence we think the 40 subjects in our study were fair in number.

7 Discussion

The proposed approach for generating analysis class diagrams from UCSs is a fully automated approach which is supported by a GUI based tool support named AutoAMG. The approach provides in-place visualization of the generated analysis class diagrams. The results of the experimental study conducted for the evaluation of the approach showed that the analysis class diagrams generated by our approach are of better quality. Specifically the results has shown that the analysis class diagrams generated by our approach were 46% more correct, 55% more complete and 31% less redundant than those generated by Popescu et al. approach, and were 33% more correct, 31% more complete and 31% less redundant than those generated by Yue et al. approach.

Our approach has shown significant improvement over the other two existing approaches because i) The language model (or the set of sentence patterns) used in our approach is comprehensive enough to interpret and identify class diagram elements from possibly all the sentence patterns and a few complex sentences (sentences specifying conditions and sentences containing that clause and conjunctive clause) in English whereas the language model used in Popescu et al. can not interpret common simple sentences like those containing infinitives (e.g. “The system prompts to enter the password”, “The system commands the motor to start”), present participles (e.g. “The system prints the receipt showing transaction number and date”), past participles (e.g. “The system validates the record entered by the customer”), gerunds (e.g. “The system starts printing the document”), etc. (Table 2), and the language model used in Yue et al. approach can not interpret common complex sentences like those containing that clauses (e.g. “The system checks that the password is correct”) and conjunctive clauses (e.g. “The system stops the motor when the tank is full”, “The motor stops when the tank is full.”), etc. (Table 2). ii) The transformation rules in our approach takes into account both the sentence structures of the sentences and the semantic relationships between the words of the sentences obtained from TDS to disambiguate the extraction of the desired elements of the analysis diagrams (Section 4.1.3) whereas the Yue et al. approach uses parse trees that do not depict any semantic relationship between the words in the sentences to extract the elements of the analysis class diagram. iii) Our approach applies various heuristics on POS-tags and TDS of the sentences to identify entity terms (Step 4.1, Section 4.2), and to identify domain classes and attributes from entity terms (Step 4.2.4.9, Section 4.2) whereas Yue et al. approach is unable to identify the classes and attributes that are documented as group of words in the sentences. Their approach identify the entity classes only from those noun phrases in the sentences that either contains a single noun (e.g. “customer”), or a single noun with a determiner (e.g. “The customer”), or a possessive noun (e.g. “customer’s address”) (Rule B1.1 and B1.2 Yue et al (2013a, 2015)).

Table 16 presents a comparison of the three approaches on the basis of input, output, language model and NLP constructs used. From the existing approaches in literature for the generation of analysis class diagrams, we selected the two approaches viz. Popescu et al. (2008) and Yue et al. (2013a, 2015) for comparison with our approach due to the following reasons:

1. These are the latest approaches in the literature.
2. These approaches up to certain extent are able to identify most of the elements for generating analysis class diagrams that the proposed approach identifies such as classes, their attributes, their operations, association, generalization and aggregation relationships. Whereas the other approaches are unable to identify all these element, such as the approach CM-Builder2 (Harman and Gaužauskas, 2003) does not identifies class operations, aggregation and generalization relationships, the approach (Liu et al. 2004) does not identify attributes of the classes, aggregation and generalization relationships between the entity classes, and the approach (Ilieva and Ormandjeva, 2006) does not identify attributes, operations, relationship names and relationship types.
3. These approaches require only the UCSs as input, they do not require additional information along with UCSs like glossary files as required by the approach (Liu et al. 2004) for identifying the candidate classes and as required by the approach (Deeptimahanti and Babar 2009; Deeptimahanti and Sanyal 2009) for eliminating the redundant classes and attributes.

4. They do not require human intervention for eliminating irrelevant classes as required in Deeptimahanti and Babar (2009); Deeptimahanti and Sanyal (2009).

Table 16 Comparison with existing automated approaches

| Approach      | Input            | Sentence patterns recognized / Language model | Theoretical foundation of sentence patterns / language model | No. of Transformation rules | NLP constructs used | Output              |
|---------------|------------------|---------------------------------------------|-----------------------------------------------------------|----------------------------|-------------------|-------------------|
| Popescu et al (2008) | Textual specifications | 2 Simple 4 complex 5 specific 5 special | 5 sentence patterns based on constraining grammar proposed in Juristo et al (2000) | 12 | link types | Analysis class diagram |
| Yue et al. (2013a, 2015) | Textual specifications | 12 NIL 9 NIL | 5 sentence patterns proposed in English grammar book (Greenbaum 1996) | 14 | parse tree | Analysis class diagram |
| Our Textual specifications | 26 8 4 4 | 25 sentence patterns proposed in Oxford Advanced Learner’s Dictionary of Current English (A.S. Hornby 1974, 2000) and in Hornby (1975) that are successively followed by various researchers in linguistic domain (Hanks 2008) | 54 | TDs and POS-tags | Analysis class diagram |

The sentence structure rules used in our approach were crafted using the twenty five verb patterns originally proposed by A.S.Hornby et al. in Oxford Advanced Learner’s Dictionary of Current English (A.S. Hornby 1974, 2000) and in Hornby (1975), and successively followed by various researchers in linguistic domain (Hanks 2008). In a grammatical analysis of 152 UCSs, Cox (2002) identified 27 most common grammar structures that are used in documenting the UCSs. Our sentence structure rules include all these grammar structures. The proposed sentence structure can recognize possibly all the simple sentences (including sentences containing participles, infinitives and gerunds) as well as a few complex sentences viz. sentences specifying conditions, sentences containing that clause and conjunctive clause. The proposed sentence structure rules when applied on 2000 simple sentences in English taken from Tanaka corpus were able to correctly recognize 92% sentences (which is <100% because of incorrect POS-tags and TDs generated by the parser for some sentences).

To find the appropriate quality metrics for evaluating the analysis class diagrams we investigated the existing quality metrics, we found that a number of quality metrics to validate design class diagrams (or design models) have been proposed in literature (Rosenberg and Hyatt 1997; Genero et al. 2002; Manso et al. 2003; Genero et al. 2005; McQuillan and Power 2007; Ma et al. 2010) but the quality metrics for validating analysis class diagrams (or analysis models) have been rarely explored. As the analysis class diagram is the first structural representation of the problem level objects and the relationships between them. The quality metrics for validating analysis class diagrams must be able to estimate how much correctly and how much completely the transformation from the requirements to the analysis models has been done (Mohagheghi et al. 2009). In Yue et al. (2013b) Yue et al. defined and used a set of measures to compare the quality of analysis class diagrams developed by the subjects in their experiment with the reference class diagrams (the reference class diagrams were taken from software engineering books, and those developed by the student experts). But, it is difficult to get the reference analysis class diagrams for every UCS. Moreover, different experts may derive different analysis class diagrams from the same set of requirements so it is also difficult to get a single reference class diagram for a given UCS. Therefore we modified and re-formulated those quality measures, so that the measures can be used to assess the quality of analysis class diagrams even if the reference class diagrams are not available. We present 15 sub metrics used for estimating the quality (correctness, completeness and redundancy) of the analysis class diagrams. Out of these 15 sub metrics, 13 are newly defined by us and only 2 sub metrics (correctly named and correctly stereotyped for class correctness) are taken from Yue et al. (2013b). These modifications are done based on the model quality goals obtained through a systematic literature review on model quality (Mohagheghi et al. 2009) and the methodology a human analyst uses to identify problem level objects and relationships from the vocabulary of the problem domain (Booch 2006).

13 http://tatoeba.org/eng/
Some limitations of the proposed approach are as follows:

1. The approach requires the UCSs to be written using a few restrictions shown in Table 6. The restrictions are required for handling the issues that are associated with the natural language such as ambiguity, variety of sentence types, anaphora (or pronoun) resolution problem and inconsistency. At present the approach can process and interpret all the simple sentences (including sentences containing participles, infinitives and gerunds) as well as a few complex sentences viz. sentences specifying conditions, the sentences containing that clause and conjunctive clause. In a grammatical analysis of 152 UCSs, Cox (2002) identified 27 most common grammar structures that are used in documenting the UCSs. These grammar structures are the subset of the sentence structures that our approach can process and interpret.

2. As the approach uses NL parser to process the sentences in UCS, hence the accuracy of the approach is inherently bounded by the accuracy of the NL parser. The Stanford NL parser used in our approach generates POS tags with accuracy of about 97% (Manning 2011) and TDs with accuracy of about 84.2% (Cer et al 2010), these figures are for all the type of sentences, the accuracy may be more for the simple sentences and a few complex sentences that our approach handles at present. However, the parser accuracy in generating TDs can further be improved from 84.2% to 89.1% by using Charniak-Johnson re-ranking parser (Cer et al 2010; McClosky et al 2006) for generating the dependencies and converting them to Stanford TDs.

3. The proposed approach does not use entity disambiguation techniques such as misspelling identification, abbreviation identification, alias identification etc. (Misra and Das 2013)

4. The current implementation of AutoAMG supports the input of one UCS at a time, and the generation of analysis class diagram for one UCS at a time. But, the interactions of the given UCS with other UCSs, as specified in the UCS using INCLUDE, EXTEND keywords and Parent Use Case Name field of the UCS, are shown in the generated analysis class diagram with the help of INCLUDE, EXTEND and generalization relationships between the control class representing given UCS and the control classes representing the other UCSs respectively. In future we will extend our tool to take a set of UCSs for a given problem as input, and to generate the analysis class diagram for the problem.

The experimental study reported in this paper is to the best of our knowledge the first one to be conducted for comparing different automated approaches used to generate analysis class diagrams. We made the following experimental material along with other details available at [14] for other researchers of this domain to replicate such experiments in future viz. the forty UCSs used in the experimental study, the analysis class diagrams generated by the three approaches (our approach, Popescu et al. approach and Yue et al. approach) for the forty UCSs, the questionnaires for the evaluation of the analysis class diagrams and the tool AutoAMG.

8 Related Work

This section presents the existing approaches in the literature to automate the process of generating analysis models from software requirements. The available approaches can be classified into semi automated approaches and automated approaches. The semi automated approaches require human intervention for identifying the elements for the generation of the analysis class diagrams whereas the automated approaches do not require any human intervention.

Mich (1996); Mich and Garigliano (2002) proposed a semi automated approach supported by a CASE tool NL-OOPS (Natural Language - Object Oriented Production System) for generating object oriented models from unrestricted natural language requirements documents. The tool uses as a core the semantic network of a Natural Language Processing System LOLITA (Large-scale Object-based Language Interactor, Translator, and Analysers) for identifying classes and relationships. The knowledge of the requirements document is stored in the knowledge base of LOLITA, which adds new nodes to its semantic network. The class model which includes classes and associations is derived from the nodes in the semantic network. The tool requires user intervention for deleting the extra nodes representing spurious classes and to set the level of details for class hierarchy.

Overmyer et al (2001) proposed a semi automated approach supported by a tool named Linguistic Assistant for Domain Analysis (LIDA) that helps an analyst in deriving analysis models. The tool first reads the textual requirements, then it presents the user with list of nouns, adjectives and verbs from which user has to identify and mark the candidate classes, attributes and the methods. The tool requires user intervention at every step to identify the elements for generating the class diagram.

Harmain and Gaizauskas (2003) proposed a semi automated approach supported by a tool named CM-Builder1 for the generation of analysis class diagrams. The approach first reads the textual requirements and provides the lists of

[14] http://serg.iiitdmj.ac.in/tools/AutoAMG/
candidate classes, attributes and relationships to the user. It also assigns the frequency of reference in the text to each class. The classes having more frequency of references are highly suggested classes. The user has to select the required classes and their attributes and relationships between them from the corresponding candidate lists.

Samarasinghe and Söme (2005) proposed a semi-automated approach supported by a tool named UCEd to generate domain models from use case models. The tool takes as input a UCS written using the grammar provided by them. The tool then makes every possible combinations of the words in the sentences of the UCS and present them to the user. The user has to select the required classes, their operations, their attributes etc. from the presented word combinations.

Harmain and Gaizauskas (2003) proposed an approach supported by a tool named CMBuilders2 that obtains a first cut domain models from the NL requirements automatically. The tool considers all the nouns as candidate classes, all the non copular verbs as candidate relationships; it obtains attributes from possessive relationships and adjectives. The candidate classes having low frequency in the requirements and also do not participating in any relationships are discarded by the tool. The tool fails to identify class operations, aggregation and generalization relationships. Moreover, the generated models contain many unconnected components.

Liu et al (2004) proposed an automated approach named UCDA for generating analysis class diagrams from UCS written using restricted grammar. Their approach processes the sentences based on the classification of sentences as transitive, intransitive, ditransitive, intensive, complex transitive, prepositional and non-finite given in Roberts (1956). To identify domain classes, the approach uses POS-tags generated by a NL parser and a glossary that defines specific terms of the domain. The approach fails to identify attributes of the classes, and aggregation and generalization relationships between the entity classes.

Ilieva and Ormandjiev (2006) proposed an approach that obtains domain models from unrestricted NL requirements. The approach uses a POS-tagger to mark the words in the NL requirements with parts of speech. Then it identifies three roles from the sentences: subject, predicate and object. Using these roles, a semantic network of the words is created. The semantic network is then transformed into a hybrid activity model and a domain model. The domain model represents only the identified classes and the relationship between them, it lacks the attributes, operations, relationship names and relationship types. Any tool supporting this approach was not presented by the authors.

Popescu et al (2008) proposed an approach supported by a prototype tool named Dowser to identify inconsistencies in requirement specifications with the help of automatically generated domain models. The approach uses constraining grammar proposed in Juristo et al (2000) that allows only five sentence structures: classification (bottom up, top down and multiple) that represents generalization relationship, composition (component and content) that represents aggregation relationship, identification that represents attribute, complement enumeration and adjacent complements that represent association relationship. The tool first parses the requirements based on the constraining grammar. Then using a NL parser it generates a link grammar parse of the sentences. It then uses link types to identify the classes, methods, variables and associations, and generates a textual object-oriented analysis model of the specified system. The textual model is then visualized using GraphViz (this needs human involvement). The approach is unable to identify many relationships between the classes that results in many unconnected components in the class diagram.

Deeptimahanti and Babar (2009), Deeptimahanti and Sanyal (2009) proposed an approach supported by tools named UML Model Generator from analysis of Requirements (UMGAR) to generate class models from restricted natural language requirements. It identifies classes from nouns, attributes from adjectives, and methods from verbs. With the help of a glossary it eliminates the redundant classes. Attribute classes are eliminated by using a text file which contains a list of words that takes values or indicate status. For eliminating irrelevant classes it requires human intervention. The generated class diagram has many unconnected component. The approach requires the requirements to be stated in either subject-verbobjekt-object format. It also does not allow passive voice sentences.

Yue et al (2013a,b) proposed an automated approach to derive analysis models from use case models. The approach first uses a NL parser to obtain the parse tree and TDs of the sentences. The approach then uses parse tree and TDs to identify sentence structures. It identifies the sentences on the basis of sentence structures formed using five basic English sentence patterns proposed in Greenbaum (1996). It then uses the parse tree to identify the elements for generating the class diagram. The approach fails to recognize domain objects and attributes which are documented as a group of words (or nouns). It also fails to distinguish between a class and an attribute, resulting in many incorrectly identified relationships. The generated class diagrams have many unconnected components. Moreover the approach dumps most of the operations in a single control class, whereas most of the other classes are assigned no operations at all, hence the division of responsibilities among the classes is not properly done by the approach.

To summarize the earlier efforts done in the direction of obtaining the analysis models from software requirements: the semi-automatic approaches in literature Overmyer et al (2001), Samarasinghe and Söme (2005) highly relies on users for identifying the elements to derive the analysis models. Due to a very limited set of grammar rules the approaches Samarasinghe and
Some [2005; Popescu et al 2008] fail to analyze the diverse set of NL sentences resulting in an incomplete transformation of software requirements into analysis class diagrams. The analysis class diagram obtained by the approaches [Ilieva and Ormandjieva 2006; Subramaniam et al 2004] lacks either attributes, operations, relationship names, generalization or aggregation. Many relationships identified between the objects by approaches [Yue et al 2013a, 2015, 2010; Fliedl et al 2007] are incorrect. The approach [Yue et al 2013a, 2015, 2010] fails to recognize problem level objects and attributes which are documented as a group of words, it fails to distinguish between a class and an attribute, and also fails to distribute responsibilities among the classes.

In comparison the proposed approach attempts to handle the diverse set of sentences in the requirements by using a set of comprehensive sentence structure rules and a set of comprehensive transformation rules. These comprehensive sentence structure rules are based on the twenty five verb patterns proposed by A.S.Hornby in Hornby (1975) and in Oxford Advanced Learners Dictionary of Current English (A.S. Hornby 1974, 2000). To correctly extract potential classes, their attributes, operations and the relationships, the comprehensive transformation rules take into account the sentence structure of the sentences as well as the syntactic and semantic relationships between the words of the sentences. These relationships between the words are found with the help of type dependencies (TDs) and part of speech tags (POS-tags) generated using the Stanford NL parser API.

9 Conclusion and future work

In this paper we have proposed an automated approach for the generation of analysis class diagrams from software requirements documented as use case specifications (UCSs). The approach takes as input a UCS written in English language with a few restrictions, and then parses it using the Stanford NL parser APIs to generate TDs and POS-tags from the sentences in the UCS. To systematically process and interpret the sentences the approach first identifies their sentence structure by applying the proposed set of comprehensive sentence structure rules on their TDs and POS-tags. Based on the identified sentence structures it then applies the proposed set of comprehensive transformation rules on TDs and POS-tags of the sentences to identify the elements for the generation of analysis class diagram. The approach finally generates the analysis class diagram, and visualizes it using GraphViz APIs. We have implemented the proposed approach in a GUI based tool support named AutoAMG.

For the validation of the proposed approach we reported the outcome of a controlled experiment that we have conducted to compare the analysis class diagrams generated by the approach with those generated by the two existing automated approaches, one proposed by Popescu et al [2008] and the other proposed by Yue et al [2013a, 2015]. In the experiment forty subjects evaluated the correctness, completeness and redundancy of the analysis class diagrams generated by the three approaches for forty UCSs. The results of the experiment clearly showed that the analysis class diagrams generated by the proposed approach were significantly better in terms of correctness, completeness and redundancy than those generated by the other approaches. Specifically, the results for the forty UCSs showed that the analysis class diagrams generated by our approach were 46% more correct, 55% more complete and 31% less redundant than those generated by Popescu et al. approach, and were 33% more correct, 31% more complete and 31% less redundant than those generated by Yue et al. approach.

The presented work has many future directions such as to generalize the approach so that along with the simple sentences and a few complex sentences that the approach can currently interpret and transform, it can also interpret and transform other complex and compound sentences. Other issues to deal with are to handle anaphora or pronoun, to apply entity disambiguation techniques [Misra and Das 2013] such as misspelling identification, abbreviation identification, alias identification etc. The approach can be extended to further generate platform specific models (PSM) from the platform independent models (PIM), and to generate the template code. The validation study of the approach can be done in the industrial settings that may help in reenforcing the outcomes of the approach.

Appendix A  Sentence Structure Rules

Table presents the sentence structure rules.
### Table 17: Sentence Structure Rules

| Rule # | Antecedent (If the sentence contains TDs:) | Consequent (then the identified sentence structure is:) | Example sentence | Type dependencies of Example sentence |
|--------|-------------------------------------------|------------------------------------------------------|-----------------|----------------------------------------|
| SSR1   | nsubj(A, B), dobj(A, C), dobj(A, D)       | SVDDEO (Subject-Verb-IndirectObject-DirectObject)   | The system sends the user an email. | [det(system-2, The-1), nsubj(sends-3, system-2), root(ROOT-0, sends-3), det(user-5, the-4), iobj(sends-3, user-5), det(email-7, an-6), dobj(sends-3, email-7)] |
| SSR2   | nsubj(A, B), dobj(A, C), complm(D, E), nsubj(D, F) | SVDOThatClause (Subject-Verb-DirectObject-ThatClause) | The system informs the user that the battery is full | [det(system-2, The-1), nsubj(informs-3, system-2), root(ROOT-0, informs-3), det(user-5, the-4), dobj(informs-3, user-5), complm(full-10, that-6), det(battery-8, the-7), nsubj(full-10, battery-8), cop(full-10, is-9), complm(informs-3, full-10)] |
| SSR3   | nsubj(A, B), complm(C, D), nsubj(C, E)    | SWTThatClause (Subject-Verb-ThatClause)              | The system validates that the password is correct | [det(system-2, The-1), nsubj(validates-3, system-2), root(ROOT-0, validates-3), complm(correct-8, that-4), det(password-6, the-5), nsubj(correct-8, password-6), cop(correct-8, is-7), complm(validates-3, correct-8)] |
| SSR4   | nsubj(A, B), neg(D, E), aux(D, F), infmod(C, D) | SVDODInotInf (Subject-Verb-DirectObject-Not-To-Infinite) | The system warns the user not to restart the system. | [det(customer-2, The-1), nsubj(selects-3, customer-2), root(ROOT-0, selects-3), neg(fail-6, not-4), aux(restart-8, to-7), infmod(user-5, restart-8), det(system-10, the-9), dobj(restart-8, system-10)] |
| SSR5   | nsubj(A, B), neg(D, E), aux(D, F), xcomp(A, C) | SVNotToInf (Subject-Verb-Not-To-Infinite)           | The system warns the user not to restart the system. | [det(customer-2, The-1), nsubj(selects-3, customer-2), root(ROOT-0, selects-3), neg(fail-6, not-4), aux(restart-8, to-7), xcomp(selects-3, fail-6), det(tank-8, the-7), dobj(fail-6, tank-8)] |
| SSR6   | nsubj(A, B), nsubj(C, D), aux(C, E), xcomp(A, C) | SVDoLaterComp (Subject-Verb-DirectObject-to-be-Complement) | The system marks the errors to be red. | [det(system-2, The-1), nsubj(marks-3, system-2), root(ROOT-0, marks-3), det(errors-5, the-4), nsubj(red-8, errors-5), aux(red-8, to-6), cop(red-8, be-7), xcomp(marks-3, red-8)] |
| SSR7   | nsubj(A, B), dobj(A, C), aux(D, F), infmod(C, D) | SVDODInf (Subject-Verb-DirectObject-To-Infinite)    | The system commands the motor to start. | [det(system-2, The-1), nsubj(commands-3, system-2), root(ROOT-0, commands-3), det(motor-5, the-4), dobj(commands-3, motor-5), aux(start-7, to-6), infmod(motor-5, start-7)] |
| SSR8   | nsubj(A, B), dobj(A, C), partmod(C, D) and POS-tag(D)=="VBG" | SVDOPresentPart (Subject-Verb-DirectObject-PresentParticle) | The system keeps the user waiting. | [det(system-2, The-1), nsubj(keeps-3, system-2), root(ROOT-0, keeps-3), det(user-5, the-4), dobj(keeps-3, user-5), partmod(user-5, waiting-6)] |
| SSR9   | nsubj(A, B), dobj(A, C), partmod(C, D) and POS-tag(D)=="VBN" | SVDOPastPart (Subject-Verb-DirectObject-PastParticle) | The system validates the record entered by the customer. | [det(system-2, The-1), nsubj(keeps-3, system-2), root(ROOT-0, keeps-3), det(user-5, the-4), dobj(keeps-3, user-5), partmod(user-5, waiting-6)] |
| SSR10  | nsubj(A, B), nsubj(D, E), xcomp(A, C) and POS-tag(C)=="JJ" | SVDODJAdj (Subject-Verb-DirectObject-Adjective-Complement) | The system keeps the door open. | [det(system-2, The-1), nsubj(keeps-3, system-2), root(ROOT-0, keeps-3), det(door-5, the-4), nsubj(open-6, door-5), xcomp(keeps-3, open-6)] |
| SSR11  | nsubj(A, B), nsubj(C, D), xcomp(A, C) and POS-tag(C)=="NN" | SVDODNJnoun (Subject-Verb-DirectObject-NounComplement) | The system makes the user an administrator. | [det(system-2, The-1), nsubj(makes-3, system-2), root(ROOT-0, makes-3), det(user-5, the-4), nsubj(keeps-3, user-5), adminmod(keeps-3, administrator-7)] |
| SSR12  | nsubj(A, B), dobj(A, C), advmod(D, E), aux(D, F), xcomp(A, D) | SVDODConjInf (Subject-Verb-DirectObject-Conjunctive-To-Infinite) | The system tells the user where to go. | [det(system-2, The-1), nsubj(tells-3, system-2), root(ROOT-0, tells-3), det(user-5, the-4), dobj(tells-3, user-5), advmod(tells-3, go-8), aux(go-8, to-7), xcomp(tells-3, go-8)] |
| SSR13  | nsubj(A, B), dobj(A, C), advmod(D, E), nsubj(D, F) | SVDODConjClause (Subject-Verb-DirectObject-Conjunctive-Clause) | The system stops the motor when the tank is full. | [det(system-2, The-1), nsubj(stops-3, system-2), root(ROOT-0, stops-3), det(motor-5, the-4), dobj(stops-3, motor-5), advmod(full-10, when-6), det(tank-8, the-7), nsubj(full-10, tank-8), cop(full-10, is-9), dep(motor-5, full-10)] |
| SSR14  | nsubj(A, B), dobj(A, C), advmod(A, D) | SVDODAdverbial (Subject-Verb-DirectObject-Adverbial) | The server responds the query quickly. | [det(server-2, The-1), nsubj(responds-3, server-2), root(ROOT-0, responds-3), det(query-5, the-4), dobj(responds-3, query-5), advmod(responds-3, quickly-6)] |
| SSR15  | nsubj(A, B), aux(A, C), dobj(A, D)       | SAuxVDO (Subject-AuxiliaryVerb-DirectObject)         | The system will eject the ATM card. | [det(system-2, The-1), nsubj(eject-4, system-2), aux(eject-4, will-3), root(ROOT-0, eject-4), det(card-7, the-5), nn(card-7, ATM-5), dobj(eject-4, card-7)] |
| SSR16  | nsubj(A, B), dobj(A, C), prep(A, D), dobj(D, E) | SVDOPPO (Subject-Verb-DirectObject-PrepositionObject) | The system sends the message to the customer. | [det(system-2, The-1), nsubj(sends-3, system-2), root(ROOT-0, sends-3), det(message-5, the-4), dobj(sends-3, message-5), prep(sends-3, to-6), det(customer-8, the-7), prep(to-6, customer-8)] |
| SSR17  | nsubj(A, B), advmod(C, D), aux(C, E), xcomp(A, C) | SVConjInf (Subject-Verb-Conjunctive-To-Infinite)    | The system guides where to go. | [det(system-2, The-1), nsubj(guides-3, system-2), root(ROOT-0, guides-3), advmod(6, where-4), aux(6, to-5), xcomp(guides-3, go-6)] |
| SSR18  | nsubj(A, B), advmod(C, D), nsubj(C, E), advcl(C, A) | SVConjClause (Subject-Verb-Conjunctive-Clause)      | The motor stops when the tank is full. | [det(motor-2, The-1), nsubj(stops-3, motor-2), root(ROOT-0, stops-3), advmod(full-8, when-4), det(tank-6, the-5), nsubj(full-8, tank-6), cop(full-8, is-7), advcl(stops-3, full-8)] |
### Sentence Structure Rules (Appendix A) – continued from previous page

| Rule # | Antecedent (If the sentence contains TDS) | Consequent (then the identified sentence structure is) | Example sentence | Type dependencies of Example sentence |
|-------|------------------------------------------|------------------------------------------------------|------------------|---------------------------------------|
| SSR19 | nsubj(A,B), aux(C,D), xcomp(A,C)         | SVToInf (Subject-Verb-To-Infinitive)                 | The system starts to fill the tank. | [det(system-2, The-1), nsubj(starts-3, system-2), root(ROOT-0, starts-3), aux(fill-5, to-4), xcomp(starts-3, fill-5), det(tank-7, the-6), dobj(fill-5, tank-7)] |
| SSR20 | nsubj(A,B), xcomp(A,C)                   | SVGerund (Subject-Verb-Gerund)                       | The printer starts printing the document | [det(printer-2, The-1), nsubj(starts-3, printer-2), root(ROOT-0, starts-3), xcomp(starts-3, printing-4), det(document-6, the-5), dobj(printing-4, document-6)] |
| SSR21 | nsubj(A,B), advmod(A,C)                  | SVAdverbialAdjunct (Subject-Verb-AdvervialAdjunct)  | The elevator moves up or down. | [det(elevator-2, The-1), nsubj(moves-3, elevator-2), root(ROOT-0, moves-3), advmod(moves-3, up-4), cc(up-4, or-5), conj(up-4, down-6)] |
| SSR22 | nsubj(A,B), cop(A,C)                     | SVPredicateic (Subject-Verb-Predicative)             | The customer is employee. | [det(customer-2, The-1), nsubj(employee-4, customer-2), cop(employee-4, is-3), root(ROOT-0, employee-4)] |
| SSR23 | nsubj(A,B), prep(A,C), num(D,E), pobj(C,D) | SVForComp (Subject-Verb-For-Complement)             | The system waits for 5 seconds. | [det(system-2, The-1), nsubj(waits-3, system-2), root(ROOT-0, waits-3), prep(waits-3, for-4), num(seconds-6, 5-5), pobj(for-4, seconds-6)] |
| SSR24 | nsubjpass(A,B), auxpass(A,C), prep(A,D), pobj(D,E) | SVPassPO (Subject-AuxiliaryVerbPassive-PrepositionObject) | The ATM card is ejected by the system. | [det(card-3, The-1), nn(card-3, ATM-2), nsubjpass(ejected-5, card-3), auxpass(ejected-5, is-4), root(ROOT-0, ejected-5), prep(ejected-5, by-6), det(system-8, the-7), pobj(by-6, system-8)] |
| SSR25 | nsubjpass(A,B), aux(A,C), auxpass(A,D), prep(A,E), pobj(E,F) | SAuxVPassPO (Subject-AuxiliaryVerbPassive-PrepositionObject) | The ATM card will be ejected by the system. | [det(card-3, The-1), nn(card-3, ATM-2), nsubjpass(ejected-6, card-3), aux(ejected-6, will-4), auxpass(ejected-6, be-5), root(ROOT-0, ejected-6), prep(ejected-6, by-7), det(system-9, the-8), pobj(by-7, system-9)] |
| SSR26 | nsubj(A,B), prep(A,C), pobj(D,E)         | SVPO (Subject-Verb-PrepositionObject)                | The system prompts for password. | [det(system-2, The-1), nsubj(prompts-3, system-2), root(ROOT-0, prompts-3), prep(prompts-3, for-4), pobj(for-4, password-5)] |
| SSR27 | nsubj(A,B), dobj(A,C)                    | SVDO (Subject-Verb-DirectObject)                     | ATM customer enters the withdrawal amount. | [det(customer-2, ATM-1), nsubj(enter-3, customer-2), root(ROOT-0, enter-3), det(amount-6, the-4), nn(amount-6, withdrawal-5), dobj(enter-3, amount-6)] |
| SSR28 | mark(A,B)                                | Conditional                                          | If the ATM card is valid. | [mark(valid-6, If-1), det(card-4, the-3), nn(card-4, ATM-3), nsubj(valid-6, card-4), cop(valid-6, is-5), root(ROOT-0, valid-6)] |
| SSR29 | nsubj(A,B)                               | SV (Subject-Verb)                                   | The system restarts | [det(system-2, The-1), nsubj(restarts-3, system-2), root(ROOT-0, restarts-3)] |

Note: N/A = Not Applied, M4-M8 represents the step Nos. M4 to M8 of UCS.

### Appendix B Transformation Rules

1) Rule TR1 is used to identify entity terms

---

**Rule-TR1**: For each sentence in the UCS

1. Concatenate the consecutive words in the sentences whose POS-tags starts with “NN”.
2. Parse the modified sentences again and Update the TDs and POS-tags of the sentences to reflect the changes.

**Example**: For sentence “The system ejects the ATM card.”, the POS-tags generated by the parser are shown in Table 4, here the consecutive words whose POS-tag starts with “NN” are “ATM” and “card”, hence they are combined or concatenated using the transformation rule TR1 to get “ATMcard” representing a single entity or a single attribute.

The modified sentence is: “The system ejects the ATMcard.”

The updated TDs are: [det(system-2, The-1), nsubj(ejects-3, system-2), root(ROOT-0, ejects-3), det(ATMcard-5, the-4), dobj(ejects-3, ATMcard-5)]

The updated POS-tags are: [The/DT, system/NN, ejects/VBZ, the/DT, ATMcard/NN, /]
2) Rules TR2-TR3 are used to identify control and boundary classes

**Rule-TR2:** createClass(UCS.name, "<<control class>>");

**Example:** From UCS shown in Table 3, the approach identifies the control class named “WithdrawFunds <<control class>>”, where the stereotype “<<control class>>” denotes that the class is control class.

**Rule-TR3:** For each actor of the UCS, createClass(actor.name, "<<boundary class>>");

**Example:** From UCS shown in Table 3, the approach identifies the boundary class named “ATMcustomer <<boundary class>> <<primary class>>”, where the stereotype “<<boundary class>>” <<primary class>>” denotes that the class is boundary class of primary actor.

3) Rules TR4-TR9 are used to identify entity classes and their attributes

**Rule-TR4:** POS-tags of each sentence in the UCS are scanned, and all the nouns are stored in a list named ListOfNouns.

For every two nouns n1 and n2 in ListOfNouns

If n2 startsWith n1 then

class = createClass(A, "<<entity class>>"); class.addAttribute(B);

EndIf

EndFor

**Example:** Let ListOfNouns contains two nouns say n1=“Withdrawal” and n2=“withdrawalAmount” then by the rule TR4 “Withdrawal” is identified as a class and “withdrawalAmount” is added as an attribute of class “Withdrawal”.

**Rule-TR5:** If TDs of the sentence contain TDs nsubj(has, A) and dobj(has, B) then

class = createClass(A, "<<entity class>>"); class.addAttribute(B);

EndIf

**Example:** The TDs generated by the parser for sentence M8 of UCS-WithdrawFunds are:

det(system-2, The-1), nsubj(validates-3, system-2), root(ROOT-0, validates-3), complm(has-7, that-4), det(AMT-6, the-5), nsubj(has-7, AMT-6), ccompt(validates-3, has-7), amodfunds-9, enough-8), dobj(has-7, funds-9)

Applying the rule TR5 on TDs nsubj(has-7, AMT-6) and dobj(has-7, funds-9), the approach identifies “AMT” as class and “funds” as attribute of class “AMT”.

**Rule-TR6:** If TDs of the sentence contain TDs prep(A, “in”), and pobj(“in”, B)

If(isNoun(B)) then

class = createClass(B, "<<entity class>>"); class.addAttribute(A);

For all TDs of type conj(A, C) before prep(A, “in”)

class.addAttribute(C);

EndIf

EndFor

**Example:** For sentence “The system prints the transactionNumber and balance on the receipt.” The TDs generated by the parser are:

det(system-2, The-1), nsubj(prints-3, system-2), root(ROOT-0, prints-3), det(transactionNumber-5, the-4), dobj(prints-3, transactionNumber-5), cc(transactionNumber-5, and-6), conj(transactionNumber-5, balance-7), prep(transactionNumber-5, in-8), det(receipt-10, the-9), pobj(on-8, receipt-10)

From the TDs prep(transactionNumber-5, on-8) and pobj(on-8, receipt-10), receipt is identified as class and transactionNumber is identified as attribute of class receipt. Also from TDs conj(transactionNumber-5, balance-7) and prep(transactionNumber-5, in-8), balance is also identified as attribute of class receipt.

**Rule-TR7:** If TDs of the sentence contain TDs prep(A, “of”), and pobj(“of”, B)

If(isNoun(B)) then

class = createClass(B, "<<entity class>>"); class.addAttribute(A);

For all the consecutive TDs of type conj(A, C) before prep(A, “of”)

class.addAttribute(C);

EndIf

EndFor

**Example:** For sentence “The system prompts for the userName and password of the customer.” the TDs generated by the parser are:

det(system-2, The-1), nsubj(prompts-3, system-2), root(ROOT-0, prompts-3), prep(prompts-3, for-4), det(userName-6, the-5), pobj(for-4, userName-6), cc(userName-6, and-7), conj(userName-6, word-8), prep(userName-6, of-9), det(customer-11, the-10), pobj(of-9, customer-11)

From the TDs prep(userName-6, of-9) and pobj(of-9, customer-11), customer is identified as class and userName is identified as attribute of class customer. Also from TDs conj(userName-6, word-8) and prep(userName-6, of-9), password is also identified as attribute of class customer

**Rule-TR8:** If TDs of the sentence contain TDs poss(A, B) then

class = createClass(B, "<<entity class>>"); class.addAttribute(A);

EndIf

**Example:** For sentence “The system prompts for customer’s address.” the TDs generated by the parser are:

det(system-2, The-1), nsubj(prompts-3, system-2), root(ROOT-0, prompts-3), prep(prompts-3, for-4), poss(address-7, customer-5), possessive(customer-5, ’s-6), pobj(for-4, address-7)

From TD poss(address-7, customer-5), customer is identified as class and address is identified as the attribute of class customer
Rule-TR9: If TDs of the sentence contain TD amod(A, B) then class = createClass("<<entity class>>"), class.addAttribute(B); EndIf

Example: For sentence “The system informs the interested user.” the TDs generated by the parser are:
[det(system-2, The-1), nsubj(informs-3, system-2), root(ROOT-0, informs-3), det(user-6, the-4), amod(user-6, interested-5), dobj(informs-3, user-6)]

From TD amod(user-6, interested-5) user is identified as an entity class and interested is identified as attribute of class user.

4) Rules TR10-TR42 are used to identify class operations and more class attributes, shown in Table 18.

Table 18: Transformation Rules to identify class operations and more class attributes

| Rule # | Antecedent (IF A)                                                                 | Consequent (then use B to identify C) | C (operations/attributes)  |
|-------|-----------------------------------------------------------------------------------|--------------------------------------|-----------------------------|
| TR11  | SVIODO                                                                           | nsubj(A, B), iobj(A, C), dobj(A, D)  | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR12  | SVDODthatClause                                                                  | nsubj(A, B), dobj(A, C), complm(D, E), nsubj(D, F) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR13  | SVThatClause                                                                     | nsubj(A, B), complm(C, D), nsubj(C, E) | op.SourceEntityTerm=B, op.DestEntityTerm=E, op.name=A |
| TR14  | SVDODonToInf                                                                     | nsubj(A, B), dobj(A, C), neg(D, E), aux(D, F), infmod(C, D) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR15  | SVNNotToInf                                                                      | nsubj(A, B), neg(C, D), aux(E, F), xcomp(A, C), dobj(C, F) | op.SourceEntityTerm=B, op.DestEntityTerm=F, op.name=A |
| TR16  | SVDODobComp                                                                      | nsubj(A, B), nsubj(C, D), aux(E, F), xcomp(C, F) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR17  | SVDODInf                                                                         | nsubj(A, B), dobj(A, C), aux(E, F), infmod(C, D) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
|       |                                                                                  |                                      | If (TDs of the sentence contains TD dobj(D,F)) then op.DestEntityTerm2=C, op.DestEntityTerm2=F; op.name2=D EndIf |
| TR18  | SVDODPresentPart                                                                 | nsubj(A, B), dobj(A, C), partmod(C, D), dobj(D, E) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
|       |                                                                                  |                                      | If (TDs of the sentence contains TD dobj(D,E)) then op.DestEntityTerm.addAttribute(E) For each TD=conj(X,Y) and (X==E) after dobj(D,E) destClass.addAttribute(Y) EndFor EndIf |
| TR19  | SVDODPastPart                                                                    | nsubj(A, B), dobj(A, C), partmod(C, D) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR20  | SVDODAdj                                                                         | nsubj(A, B), nsubj(C, D), xcomp(A, C) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR21  | SVDODNoun                                                                        | nsubj(A, B), nsubj(C, D), xcomp(A, C) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR22  | SVDODConToInf                                                                   | nsubj(A, B), dobj(A, C), advmod(D, E), aux(D, F), xcomp(A, D) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR23  | SVDODConClause                                                                   | nsubj(A, B), dobj(A, C), advmod(D, E), nsubj(D, F) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR24  | SVDODAdverbial                                                                  | textitnsubj(A, B), dobj(A, C), advmod(A, D) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR25  | SAuxVDO                                                                          | nsubj(A, B), aux(A, C), dobj(A, D) | op.SourceEntityTerm=B, op.DestEntityTerm=D, op.name=A |
| TR26  | SVDOPO                                                                           | nsubj(A, B), dobj(A, C), prep(D, A), pobj(D, E) | op.SourceEntityTerm=B, op.name=A, If(E=="to" or "from" or "on" or "in" or "into" or "through" or "of") then op.DestEntityTerm=E else op.DestEntityTerm=C |
| TR27  | SVDConToInf                                                                      | nsubj(A, B), advmod(C, D), aux(C, E), xcomp(A, C) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR28  | SVDConClause                                                                     | nsubj(A, B), advmod(C, D), nsubj(C, E), advcl(A, C) | op.SourceEntityTerm=B, op.DestEntityTerm=B, op.name=A |
| TR29  | SVToInf                                                                          | nsubj(A, B), aux(C, D), xcomp(A, C) | op.SourceEntityTerm=B, op.name=A, If found(dobj(C,E)) then op.DestEntityTerm=E Else op.DestEntityTerm=B |
| TR30  | SVGerund                                                                          | nsubj(A, B), xcomp(A, C) | op.SourceEntityTerm=B, op.name=A, If found(dobj(C,D)) then op.DestEntityTerm=D else op.DestEntityTerm=B |
Transformation Rules to identify class operations and more class attributes (Appendix B) –Continued from previous page

| Rule # | Antecedent (If A) | Consequent (then use B to identify C) | C (operations/attributes:) |
|--------|------------------|--------------------------------------|---------------------------|
| TR31   | SVA  |  | op.SourceEntityTerm=B, op.DestEntityTerm=B, op.name=A |
| TR32   | SVP   | nsbj(A, B), advmod(A, C) | If(isAdjective(C)) then B.addAttribute(C) |
| TR33   | SVFC  | nsbj(A, B), prep(A, C), num(D, E), pobj(C, D) | op.SourceEntityTerm=B, op.name=A, If(A==E) then op.DestEntityTerm=B, else op.DestEntityTerm=E |
| TR34   | SVPPO | nsbj(A, B), auxpass(A, C), prep(A, D), pobj(D, E) | op.SourceEntityTerm=E, op.DestEntityTerm=B, op.name=A |
| TR35   | SAnxPPO | nsbj(A, B), aux(A, C), auxpass(A, D), prep(A, E), pobj(E, F) | op.SourceEntityTerm=F, op.DestEntityTerm=B, op.name=A |
| TR36   | SVO   | nsbj(A, B), pobj(A, C) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR37   | SVDO  | nsbj(A, B), dobj(A, C) | op.SourceEntityTerm=B, op.DestEntityTerm=C, op.name=A |
| TR38   | Conditional | mark(A, B) | |
| TR39   | SV    | nsbj(A, B) | op.SourceEntityTerm=B, op.DestEntityTerm=B, op.name=A, op.Para="" |
| TR40   | Include or Extend | nn(A, B), nn(C, D) | op.SourceEntityTerm="system", op.name="<<A>>", op.DestEntityTerm="" |
| TR41   | Resume | nn(A, B) | op.SourceEntityTerm="system", op.DestEntityTerm="system", op.name=B, op.Para=A |
| TR42   | Loop  | nn(A, B) | op.SourceEntityTerm="system", op.DestEntityTerm="system", op.name=B, op.Para=A |
| TR43   | All except SV, SVDO, Conditional, Include, Extend, Resume and Loop | root() | op.Para="" |

5) Rules TR44-TR45 are used to identify more entity classes

**Rule-TR44:** If op.SourceEntityTerm is not present in ClassDiagram_Instance then class = createClass(op.SourceEntityTerm,"<<entity class>>");

**Rule-TR45:** For each class C in ClassDiagram_Instance

- If (op.DestEntityTerm.name==C.name) AND (C does not contain operation op.name(op.Para)) then
  - C.addOperation(op.name(op.Para));

6) Rule TR46 is used to identify association relationships

**Rule-TR46:** For each operation op

- For each relationship r in ClassDiagram_Instance
  - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==r.class2) AND (r.name does not contain op.name)
    - append op.name to r.name
  - EndIf
  - EndFor
- If no such relationship found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
  - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
- If no such class is found then
  - For each relationship r in ClassDiagram_Instance
    - If(op.SourceEntityTerm==r.class1 and op.DestEntityTerm==a.name for some attribute a of class r.class2) AND (r.name does not contain op.name)
      - append op.name to r.name
    - EndIf
  - EndFor
- If no such relationship found then
  - C.createClass(op.DestEntityTerm.name,"<<entity class>>");
  - C.addOperation(op.name(op.Para));

- EndIf
- EndFor
7) Rules TR47-TR50 are used to identify generalization relationships

**Rule-TR47**: For each sentence of type Child-GenSubString-Parent, the POS-tags of the sentence are scanned and

- parentClass=createClass(noun nr on the right of GenSubString, "<<entity class>>");
- For each noun nl on the left of GenSubString
  - childClass=createClass(nl, "<<entity class>>");
  - createRelationship(parentClass, childClass, "generalization");
- EndFor

Example: For sentence “Memory has types RAM and ROM.” POS-tags generated by the parser are:

[Memory/NN, has/VBZ, types/NNS, RAM/NNP, and/CC, ROM/NNP, /.

As this sentence contains GenSubString="types of" hence it is Child-GenSubString-Parent sentence, therefore rule TR47 is applied. From POS-tags the nouns to the left of GenSubString “types of” are Memory, RAM and ROM a child class is created for each of these nouns. And the noun to the right of GenSubString “types of” is transaction, a parent class is created for this noun. Generalization relationship is established between the identified child classes (withdrawal, deposit, transfer and query) and the identified parent class (transaction)

**Rule-TR48**: For each sentence of type Parent-GenSubString-Child, the POS-tags of the sentence are scanned and

- parentClass=createClass(noun nl on the left of GenSubString, "<<entity class>>");
- For each noun nr on the right of GenSubString
  - childClass=createClass(nr, "<<entity class>>");
  - createRelationship(parentClass, childClass, "generalization");
- EndFor

Example: For sentence “The withdrawal, deposit, transfer and query are types of transaction.” POS-tags generated by the parser are:

[The/DT, withdrawal/NN, ,/, deposit/NN, ,/, transfer/NN, and/CC, query/NN, are/VBP, types/NNS, of/IN, transaction/NNS, ./.

As this sentence contains GenSubString="types of” hence it is Parent-GenSubString-Child sentence, therefore rule TR48 is applied. A parent class is created for noun Memory on the left of GenSubString has types and for each noun RAM and ROM a child class is created. A generalization relationship is established between parent class (Memory) and the child classes (RAM and ROM)

**Rule-TR49**: The use case generalization relationships are used to identify the generalization relationships between their control classes.

Example: For UCS Withdraw Fund the generalization relationship between control classes of Withdraw Fund and ATM is directly obtained from the Parent-UseCase-Name field of UCS.

**Rule-TR50**: The actors generalization relationships are used to identify the generalization relationships between the actors boundary classes.

Example: For UCS Withdraw Fund the generalization relationship between boundary classes of ATM Customer and User is directly obtained from the Parent-Actor-Name field of UCS.

8) Rules TR51-TR53 are used to identify aggregation relationships

**Rule-TR51**: For each of the two classes c1 and c2 in ClassDiagram_Instance

- If c2 is attribute of c1 then
  - aggregateClass=createClass(c1, partClass=c2);
  - createRelationship(aggregateClass, partClass, "aggregation");
- EndIf

EndFor

The sentences in flows and description sections of UCS are scanned and the sentences containing sub strings of types “part of”, “consists of” and all their synonyms are used to identify aggregation relationships. We call such sub strings as AggSubString (or Aggregation Sub String). We categories the sentences containing AggSubString into two kinds:

1. Part-AggSubString-Whole sentences: The sentences containing sub strings "part of", "unit of", "member of" and all their synonyms are referred as Part-AggSubString-Whole sentences because in these sentences the part class/classes is/are present on the left of AggSubString and the whole class is present on the right of AggSubString.

2. Whole-AggSubString-Part sentences: The sentences containing sub strings "consist of", "made of", "contains" and their all synonyms are referred as Whole-AggSubString-Part sentences because in these sentences the whole class is present on the left of AggSubString and the part class/classes is/are present on the right of AggSubString.

The following two transformation rules (TR52 and TR53) are applied to identify aggregation relationships:

**Rule-TR52**: For each sentence of type Part-AggSubString-Whole, the POS-tags of the sentence are scanned and

- wholeClass=createClass(nl, "<<entity class>>");
- For each noun nr on the left of AggSubString
  - partClass=createClass(nr, "<<entity class>>");
  - createRelationship(wholeClass, partClass, "aggregation");
- EndFor

Example: For sentence “CardReader, CashDispenser and ReceiptPrinter are parts of ATM.” The POS-tags generated by the parser are:
As the sentence contains AggSubString="parts of" hence it is Part-AggSubString-Whole sentence therefore rule TR52 is applied. A class representing the part is created for each nouns CardReader, CashDispenser and ReceiptPrinter to the left of AggSubString "parts of" and a class representing the whole is created for the noun ATM on the right of AggSubString. An aggregation relationship is established between the classes representing the parts (CardReader, CashDispenser and ReceiptPrinter) and the class representing the whole (ATM).

Rule-TR53: For each sentence of type Whole-AggSubString-Part, the POS-tags of the sentence are scanned and wholeClass=createClass(noun nl on the right of AggSubString, "<<entity class>>");
For each noun nr on the left of AggSubString partClass=createClass(nr, "<<entity class>>");
createRelationship(wholeClass, partClass, "aggregation");
EndFor
EndFor

Example: For sentence “A Computer is composed of Hardwares and Softwares.” The POS-tags generated by the parser are:
[A/DT, Computer/NN, is/VBZ, composed/VBN, of/IN, Hardwares/NNPS, and/CC, Softwares/NNPS, ./.

As the sentence contains AggSubString="composed of" hence it is Whole-AggSubString-Part sentence therefore rule TR53 is applied. A class representing the whole is created for the noun Computer on the left of AggSubString “composed of” and a class representing the part is created for each noun Hardwares and Softwares on the right of AggSubString “composed of”. An aggregation relationship is created between the class representing the whole (Computer) and the classes representing the parts (Hardwares and Softwares)

9) The rule TR54 is used to eliminate extra classes

Rule-TR54: For each class c in ClassDiagram_Instance
If c is not present as EndClass in any relationship then
    c is deleted from ClassDiagram_Instance
EndIf
EndFor

Appendix C Types of sentences in English

There are 4 types of sentences in English:\[15\]

1. **Simple Sentence** A simple sentence consists of only one independent clause, that has subject and predicate, and expresses a complete thought.
   Example:
   i) The ATM Customer enters the withdrawal amount.
   ii) The system commands the motor to start.
   iii) The system validates the record entered by the customer.
   iv) The ATM card is ejected by the system.

2. **Compound Sentence** A compound sentence consists of two independent clauses joined by a coordinator (conjunction).
   Example:
   i) The Card reader ejects the ATM card, and the printer prints the receipt.
   ii) The system debits the customers account, but the cash is not dispensed.

3. **Complex Sentence** A complex sentence consists of an independent and one or more dependent clauses.
   Example:
   i) The system checks that the card is valid.
   ii) While the cash dispenser is counting the cash, the card reader ejects the card.
   iii) The ATM card which the customer inserted was invalid.

4. **Compound Complex Sentences** A compound complex sentence consists of a complex and a simple sentence joined by coordinator (conjunction).
   Example:
   i) The Customer inserts the card, and the system checks that the card is valid.
   ii) While the cash dispenser is counting the cash, the card reader ejects the card, and the system starts displaying the message to collect the card and cash.

Appendix D Sample questionnaires and data collection

Following are the sample questionnaires for the analysis class diagram generated by one of the approach for WithdrawFund UCS. Each subject in the experiment was given 3 sets of such questionnaires one for the analysis class diagram generated by each approach.

The questionnaires were so designed that the data collection from the answers to the questionnaires is straight forward, and the correctness, completeness and redundancy of the analysis class diagram can be easily calculated by simply putting the collected data in the formulas for correctness, completeness and redundancy of analysis class diagram presented in Section [15].

D.1 Questionnaires for class diagram correctness

The sample questionnaires for class correctness and relationship correctness are shown in Table [19] and Table [20] respectively.

---

[15] http://www.oxforddictionaries.com/words/sentences
Table 19 Questionnaire for class correctness

| S.No. | Class                  | Correctly identified as class \(C_{rc1}\) | Correctly identified attributes \(R_{rc1}\) | Relationship Name | Correctly assigned End-Class1 \(R_{sg1}\) | Correctly assigned relationship \(R_{rc2}\) | Correctly identified attributes \(R_{rc2}\) | Repetition of correctly identified attributes \(R_{sg2}\) | Repetition of correctly identified operations \(R_{op}\) |
|-------|-----------------------|------------------------------------------|-------------------------------------------|-------------------|------------------------------------------|---------------------------------------------|------------------------------------------|-----------------------------------------------|-----------------------------------------------|
| 1     | WithdrawFunds          | < < control class > >                    |                                           | Message           | displays                                 |                                             |                                           |                                               |                                               |
| 2     | Message                | < < control class > >                    |                                           | ATMCustomer       | < < boundary > >                         |                                             |                                           |                                               |                                               |
| 3     | ATMCustomer            | < < boundary > >                         |                                           | Receipt           | prints                                   |                                             |                                           |                                               |                                               |
| 4     | Receipt                | < < control class > >                    |                                           | ATM               | < < control class > >                    |                                             |                                           |                                               |                                               |
| 5     | ATM                    | < < control class > >                    |                                           | ATMControl        | < < control class > >                    |                                             |                                           |                                               |                                               |
| 6     | ATMControl             | < < control class > >                    |                                           | ATMCustomer       | < < control class > >                    |                                             |                                           |                                               |                                               |
| 7     | ATMCustomer            | < < boundary > >                         |                                           | WithdrawFunds     | < < control class > >                    |                                             |                                           |                                               |                                               |
| 8     | WithdrawFunds          | < < control class > >                    |                                           | ATMReceipt        | < < control class > >                    |                                             |                                           |                                               |                                               |
| 9     | ATMReceipt             | < < control class > >                    |                                           | ATMCustomer       | < < control class > >                    |                                             |                                           |                                               |                                               |
| 10    | ATMCustomer            | < < control class > >                    |                                           | ATMControl        | < < control class > >                    |                                             |                                           |                                               |                                               |

Note: Write 1 if the element is correctly identified, 0 otherwise.

Table 20 Questionnaire for relationship correctness

| S.No. | Relationship Name                  | Correctly identified as relationship \(R_{rc}\) | Correctly assigned End-Class1 \(R_{sg1}\) | Correctly assigned End-Class2 \(R_{sg2}\) | Correctly identified relationship type \(R_{rc}\) | Correctly identified navigability \(R_{nav}\) |
|-------|-----------------------------------|-----------------------------------------------|------------------------------------------|---------------------------------------------|-----------------------------------------------|-----------------------------------------------|
| 1     | Message                            | displays                                      |                                           |                                             |                                               |                                               |
| 2     | ATMCustomer                        | < < boundary > >                              |                                           |                                             |                                               |                                               |
| 3     | ATMReceipt                         | < < control class > >                        |                                           |                                             |                                               |                                               |
| 4     | ATMControl                         | < < control class > >                        |                                           |                                             |                                               |                                               |
| 5     | ATMCustomer                        | < < boundary > >                              |                                           |                                             |                                               |                                               |
| 6     | ATMReceipt                         | < < control class > >                        |                                           |                                             |                                               |                                               |
| 7     | ATMControl                         | < < control class > >                        |                                           |                                             |                                               |                                               |
| 8     | ATMReceipt                         | < < control class > >                        |                                           |                                             |                                               |                                               |
| 9     | ATMControl                         | < < control class > >                        |                                           |                                             |                                               |                                               |
| 10    | ATMCustomer                        | < < control class > >                        |                                           |                                             |                                               |                                               |

Note: Write 1 if the element is correctly identified, 0 otherwise.

D.2 Questionnaires for class diagram completeness

1) No. of sentences in the functional requirements whose functionalities are assigned as operations to some class (or classes) in the class diagram \(N_{o}\) = ?

2) No. of separate groups of classes and relationships in the class diagram \(N_{sg}\) = ?

D.3 Questionnaires for class diagram redundancy

1) No. of redundant or extra classes in the class diagram \(N_{c}\) = ?
   (Note: A class is considered as a redundant class if it does not participate in any relationship with other classes in the class diagram or if it is an incorrectly identified class)

2) No. of redundant or extra relationships in the class diagram \(N_{r}\) = ?
   (Note: More than one relationships of same relationship type and navigability between the two same end classes are considered as redundant relationships. An incorrect relationship is also considered as redundant relationship)

References

Abbott RJ (1983) Program design by informal english descriptions. Communications of the ACM 26(11):882–894
Arango G (1989) Domain analysis: From art form to engineering discipline. ACM Sigsoft software engineering notes 14(3):152–159
Arendt T, Taentzer G (2010) Uml model smells and model refactorings in early software development phases. Universitat Marburg
Arisholm E, Brudin LC, Hore SE, Labiche Y (2006) The impact of uml documentation on software maintenance: An experimental evaluation. Software Engineering, IEEE Transactions on 32(6):365–381
AS Hornby ACG Anthony Paul Cowie (1974) Oxford advanced learner’s dictionary of current English. Oxford University Press
 AS Hornby SW Michael Ashby (2000) Oxford advanced learner’s dictionary of current English. Oxford University Press
Beck K, Cunningham W (1989) A laboratory for teaching object oriented thinking. In: ACM Sigplan Notices, ACM, vol 24, pp 1–6
Bézivin J (2006) Model driven engineering: An emerging technical space. In: Generative and transformational techniques in software engineering, Springer, pp 36–64
Booch G (2006) Object Oriented Analysis & Design with Application. Pearson Education India
Bresnan J (2001) Lexical-functional syntax, vol 16. Blackwell Oxford
Bruegge B, Dutoit AA (1999) Object-oriented software engineering; conquering complex and changing systems. Prentice Hall PTR
Carroll J, Minnen G, Briscoe T (1999) Corpus annotation for parser evaluation. In: Proceedings of the EACL workshop on Linguistically Interpreted Corpora (LINC)
Cer DM, De Marneffe MC, Jurafsky D, Manning CD (2010) Parsing to stanford dependencies: Trade-offs between speed and accuracy. In: LREC
Corder GW, Foreman DI (2009) Nonparametric statistics for non-statisticians: a step-by-step approach. John Wiley & Sons
Popescu D, Rugaber S, Medvidovic N, Berry DM (2008) Reducing ambiguities in requirements specifications via automatically created object-oriented models. In: Innovations for Requirement Analysis. From Stakeholders Needs to Formal Designs, Springer, pp 103–124
Porter A, Votta L (1998) Comparing detection methods for software requirements inspections: A replication using professional subjects. Empirical software engineering 3(4):355–379
Roberts P (1956) Patterns of English. Harcourt, Brace
Rosenberg D, Stephens M (2007) Use case driven object modeling with uml. APress, Berkeley, USA
Rosenberg D, Stephens M (2008) Use Case Driven Object Modeling with UML: Theory and Practice. Springer Science & Business
Rosenberg LH, Hyatt LE (1997) Software quality metrics for object-oriented environments. Crosstalk journal 10(4)
Samarasinghe N, Somé SS (2005) Generating a domain model from a use case model. In: IASSE, p 278
Sheskin DJ (2003) Handbook of parametric and nonparametric statistical procedures. crc Press
Shlaer S, Mellor SJ (1988) Object-oriented systems analysis: modeling the world in data
Siqueira FL, Silva PSM (2011) An essential textual use case meta-model based on an analysis of existing proposals. In: WER
Sjøberg DI, Hannay JE, Hansen O, Kampenes VB, Karahasanovic A, Liborg NK, Rekdal AC (2005) A survey of controlled experiments in software engineering. Software Engineering, IEEE Transactions on 31(9):733–753
Subramaniam K, Liu D, Far BH, Eberlein A (2004) Ucda: Use case driven development assistant tool for class model generation. In: SEKE, pp 324–329
Wiegers K, Beatty J (2013) Software requirements. Pearson Education
Wohlin C, Höst M, Henningsson K (2003) Empirical research methods in software engineering. In: Empirical Methods and Studies in Software Engineering. Springer, pp 7–23
Wohlin C, Runeson P, Håt M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer Publishing Company, Incorporated
Yang H, De Roeck A, Gervasi V, Willis A, Nuseibeh B (2010) Extending nocuous ambiguity analysis for anaphora in natural language requirements. In: Requirements Engineering Conference (RE), 2010 18th IEEE International, IEEE, pp 25–34
Yourdon E, Coad P (1990) Object-oriented analysis. Prentice Hall
Yue T, Briand LC, Labiche Y (2010) Automatically deriving a uml analysis model from a use case model. Tech. rep., Simula Research Laboratory
Yue T, Briand L, Labiche Y (2013a) Automatically deriving a uml analysis model from a use case model. Tech. Rep. 2010-15 (Version 2), Simula Research Laboratory
Yue T, Briand LC, Labiche Y (2013b) Facilitating the transition from use case models to analysis models: Approach and experiments. ACM Transactions on Software Engineering and Methodology (TOSEM) 22(1):5
Yue T, Briand LC, Labiche Y (2015) atoucan: An automated framework to derive uml analysis models from use case models. ACM Transactions on Software Engineering and Methodology (TOSEM) 24(3):13