Controlled English Commenting System

Pradeep Victor

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree o

Master of Science
in
Electrical and Computer Engineering

February 5, 2001
Blacksburg, Virginia

Key Words:
Controlled English, Comment generation, Parsing,
GUI, Documentation.

Controlled English Commenting System

Pradeep Victor

ABSTRACT

This thesis describes the implementation of a Controlled English Commenting (CEC) system that aids a VHDL modeler in entering controlled English comments. The CE system developed includes a graphical user interface (GUI). The interface permits modeler to submit comments for insertion at user selected points in a text file containing the model. A submitted comment is analyzed for vocabulary and syntax, and is then inserted if it is controlled English. If it is not, the CEC system extracts all possible controlled English comments that can be formed from the original comment and presents them to the user for selection and entry into the model. The interface then queries the user to complete any residual portions of the original comment until the user is satisfied. Until the user becomes familiar with the constraints of the controlled language, significant interaction is needed, particularly on complex comments. Preliminary experiments indicate that users rapidly learn the language's constraints and the need for interactive help declines.

Acknowledgement

I would like to thank Dr. Walling Cyre for making this thesis possible and for all the guidance he provided. I take this opportunity to thank Dr. Armstrong and Dr. Broadwater for serving as members of my committee.

I am also thankful to my parents and the rest of my family for all the support they provided me. Lastly, I wish to thank all my friends and all others who have directly or indirectly helped me in my efforts.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Goal

VHDL (VHSIC Hardware Description Language)[PelD] is a popular tool that helps to capture complex digital circuit designs for both simulation and synthesis. A language optimized for electronic circuit design, VHDL, is helpful in various levels of the circuit design process. It can be used in the high-level design stage to capture performance and interface requirements of a large system's various components. In the design capture phase, details of the system are entered in a computer-based design system using VHDL descriptions that are combined with other representations, such as schematics, to form the complete system. After design capture, the next step is to simulate the operation of th circuit to find out if it will meet the functional and timing requirements developed in the specification stage. The structured programming features of VHDL, along with its configuration management features make VHDL a natural form in which to model a larg and complex circuit.

Importance of VHDL Comments

The task of developing models is rather laborious and time consuming. If a circuit has been modeled with no comments, developing the circuit further will be very difficult. For example, assume that a designer returns after a long break to enhance a large circuit developed using VHDL, there is a high possibility that the designer may not remember why certain parts of the code exist. The possibility is even greater when a designer who ha never worked on the design is asked to enhance it.

Importance of comments

> *"Writing a comment makes one think harder about what his/her code is doing.*

A commenting style that requires a lot of busy work is a maintenance headache: If the comments are hard to change, they won't be changed; they'll become inaccurate and misleading, which is worse than having no comments at all.

Second, commenting might be difficult because the words to describe what the program is doing don't come easily. That's usually a sign that he/she doesn't really understand what the program does. The time one spends "commenting", is really time spent understanding the program better, time that needs to b spent regardless of whether you comment or not." [McCS93]

Importance of controlled English comments

A Controlled English (CE) is a subset of natural English that is restricted in its syntax and semantics for the purpose of readability and/or ease of processing by machine. A centra goal for Controlled English is to eliminate ambiguity and help in machine understandability and machine translation. Controlled English has been used in technical documentation, such as user manuals or maintenance manuals [WojR96] to make them easily understood. For example, the Boeing Simple English [WojR96], a research project funded by th Boeing Corporation aims at producing Controlled English maintenance documents for their airplanes. This approach of using CE for their maintenance documents, has helped Boeing render out its maintenance procedures in a non ambiguous manner, hence helping oversea clients to easily understanding the procedures, rather than depending on the parent company and hence saving both time and money. This research effort proposes the use of CE as a commenting language for VHDL code. As earlier mentioned, use of CE as a commenting language for the VHDL codes will help a user other than the program developer to easily understand what the code is doing and helps in maintenance of th code, either for bug fixing or enhancements. The Controlled English VHD Commenter (CEC) has been developed for this purpose, and hence enables the program developer to enter simple CE comments and ensure that the comments satisfy CE restrictions.

1.2 Approach

The CEC has been developed to help users write CE comments. The comments are firs parsed by a CE parser, which contains around 150 grammar rules and a vocabulary of 4900 words. The CE parser uses a bottom-up chart parse to generate all valid parse trees for its grammatical structure. If the CE parser fails to form a parse tree for a given comment, the comment is not a well-formed CE comment and has to be modified to conform.

The CE parser analyses the user comment and produces a chart, which contains the different phrases that the CE parser constructed by its analysis. Each phrase is formed by a grammar rule. These phrases are first collected and then subjected to an extensive filtering process. (Refer to section 3.2.1 for more details). The filtered phrases are then used to form CE comments by recursively applying each CE rule allowed by the CE parser.

The CEC has a GUI (Graphical User Interface) system that helps the user to interact with the system to enter CE comments. It can also break a complex comment into a number o simpler CE comments that are then inserted into the VHDL code. The CEC has been developed using Microsoft Visual C++.

1.3 Contributions

This section lists the research contributions made by this author.

Chunk collector : The phrases in the chart file generated by the CE parser are selected and placed inside three vectors (Vector is a built in data type similar to an array provided by the standard template library in C++) depending on the type of phrase (nominal, predicate or adverbial) of phrase. These three vectors correspond to a nominal, predicate and adverbial. After this collection is completed an extensive filtering procedure (described later in chapter 3) decides which of the collected phrases should be retained and which of these phrases have to be rejected.

Comment structure analyzer Once the phrases previously collected have been selectively filtered, the comment structure analyzer forms a non-terminal equivalent o the ungrammatical comment (UC) entered by the user. For example the comment C1, is converted to the form
“Nominal / predicate / conjunction / predicate”.

The signal is generated by the processor P1 and is the input to the processor P2 (C1)

CE comment generator: A recursive algorithm has been developed for CE commen generation. This procedure applies each of the CE rules for a CE comment to the non-terminal equivalent structure of the ungrammatical comment formed earlier and tries to form all possible CE comments. For every ungrammatical comment there is a possibility of more than one CE comment being generated, from which the user can select a comment depending on the information he/she is trying to convey.

Interactive comment generation module: After the user has selected a CE commen formed by the CE comment generator, this module removes the phrases used in the selected comment and then tries to form CE comments from the remaining phrases. If it is unable to form any other CE comment, and there are still phrases which have not been used in any of the selected CE comments, it queries the user for information to form another CE comment. Hence this module makes sure that all of the information that the user is trying to convey is conveyed in a CE form.

Graphical User Interface (GUI): A graphical user interface has been developed to help users in entering CE comments. The GUI has comment entry field for the user to enter a comment. A list box to show all the generated CE comments has been provided. The user can select the CE comment by double clicking on the generated comment. An inser facility has been provided to insert the selected CE comments into the code being commented.

1.3 Document Organization

This section describes the organization of this thesis.

Chapter1 formally introduces the thesis. The motivation and the scope of the thesis ar presented in this chapter. This chapter also presents a brief overview of the CEC system developed in this thesis.

Chapter 2 summarizes the background and some other works and their relationship to this project. The reader is introduced to the notion of controlled English that has been used to develop the system. Works relevant to thi thesis, are also presented in this section.

Chapter 3 deals with the theoretical aspects of this thesis. This chapter explains the algorithm used in this thesis from a theoretical perspective. In chapter 4, we talk about th actual implementation of the system. Chapter 5 presents the user interface developed for the CEC system. In Chapter 6, we walk through the testing process conducted with th CEC system and the conclusions drawn from them. In chapter 7, we briefly explore the limitations of the system and the scope for further improvement of this system.

Chapter 2

Relevant Work

2.1 Introduction

In this chapter, other research relevant to the effort of developing the CEC system i reviewed. The field of controlled languages and in particular controlled English has been an area of active research. Many organizations and individuals have contributed to this effort. Some of the prominent contributions are discussed here along with their relationship to the CEC system.

The **AECMA** (European Association of Aerospace Industries) has developed a simple form of English that aids in the documentation of the various manuals in simple English. Another related research effort is the Attempto Controlled English (ACE) a languag designed to write specifications. ACE is more of a domain specific language as it has a different set of knowledge bases for different product functionalities and has its vocabulary developed for that particular domain. The ACE approach of cyclic activity (refer section 2.2) was helpful in designing the CEC interaction phase which helps the user to enter CE comments.

The **Boeing Simplified English** Checker was developed for technical writers to check their documents for compliance with AECMA (European Association of Aerospace Industries) Simplified English, a writing standard for aerospace maintenance documentation. Another closely related research effort is the Caterpillar Technical English developed to improve and modernize its delivery of service information. The goal of the Caterpillar Technica English is similar to the Boeing simplified English Checker as both try to produc unambiguous simple English documents. Another interesting research effort is the Requirements Sublanguage a research effort by the Automatic Design Research Group of Virginia Tech. It developed the CE parser, which is used by the CEC system for the analysis of user comments and in the generation of CE comments.

2.2 AECMA Simplified English

The AECMA Simplified English [FarG96] was developed by Airbus to produce all major manuals for their Aircraft models in simple English. All vendors for the aircraft' components also have to produce their component maintenance manuals in simplified English.

The reason for the development of the Simple English was the increase in the complexity of the aircraft that increased the size and complexity of the technical documentation. For example, the Concorde maintenance manual has 28 volumes containing 28,000 pages and Airbus offers its customers 48 different sets of manuals.

An increase in the number of non-English speaking customers also motivated the development of Controlled English. These customers have difficulty in understanding Standard English because of its large vocabulary and its feature of many synonyms for particular word. For example, consider the procedural statement - "Round (verb) the edges of the round (Adjective) cap. If it then turns round (Adverb) and round (Adverb) as it circles round (Preposition) the casing, another round (Noun) of tests is required". Th Oxford English Dictionary has one whole page to explain the word 'round'. AECMA proposed to reduce these problems with a specification that ensured fewer words and simplified structure.

AECMA uses approved words from the *SE guide, Technical names* and *Manufacturing process guide* [FarG96] as its sources for word . The AECMA simple English has restricted vocabulary of 950 words and 55 grammar rules. This idea of generating simple English documents has helped in the development of the CEC system for generating simple comments to aid in VHDL code documentation. CEC uses around 150 grammar rules and 4900 words in its vocabulary, offers more flexibility to the user and ensures that only CE comments are inserted into the VHDL code.

2.3 Attempto - Controlled English as a Specification Language

Attempto Controlled English (ACE) [FucN96] allows domain specialists to interactively formulate requirements specification in domain concepts. ACE is expressive enough to allow natural usage. The Attempto system translates specification texts in ACE into discourse representation structures and optionally into Prolog. Translated specificati texts are incrementally added to the knowledge base. The knowledge base can be queried in ACE for verification and executed for simulation, prototyping and validation of the specification. ACE is a computer processable subset of English for writing requirements specifications.

Structural ambiguity is eliminated from ACE by two methods. First, the language does not admit certain ambiguous sentences or provides unambiguous alternatives. However, all ambiguous sentences cannot be eliminated in order to keep the language as natural a possible. A second approach is then used to help in reducing the ambiguity. The sentence is parsed deterministically according to a small number of rules associated with syntactic constructions. A paraphrase is then generated to show how the sentence was parsed. If the user does not find the paraphrase to coincide with what was intended, the user reformats the sentence or decomposes the sentence into smaller unambiguous ones.

2.4 The Boeing Simplified English Checker

The Boeing Simplified English Checker [WojR96] helps technical writers check their documents for compliance with AECMA (European Association of Aerospace Industries) Simplified English.

The SE checker relies on a syntactic formalism that was inspired by the Generalized Phrase Structure Grammar (GPSG). It produces fully ambiguous parse forests by means of a Boeing-developed parser, a comprehensive English grammar and an extensive lexicon. As a syntax-based system, the Boeing SE checker can support only the use of AECMA SE. Using a word only in its approved meaning is a very important part of the standard, but the SE checker described here only gives feedback on the part of speech deviations. It does not help when a word is used in the correct part of speech but with an unapproved meaning.

The Boeing Simplified English Checker makes use of Boeing Technical English (BTE)[WojR98]. It deals more with the physical aspect of the system being discussed in the controlled English documents, whereas, the CEC was designed for describing the behavioral aspect. The approach taken by the CEC to check if a user comment is of the CE form, is similar to that performed by the Boeing Simplified English Checker.

2.5 Caterpillar Technical English(CTE)

Caterpillar Technical English [HayP96] was developed by Caterpillar, Inc. to improve and modernize its delivery of service information. The system consists of three parts, the Caterpillar Technical English (CTE), CTE Checker and the Machine Translation System.

CTE is a type of controlled English designed to express Caterpillar's service informati in a way that meets the requirements of the translation technology for accurate translation. CTE includes several thousand individual words, both general and technical, most of which are restricted to a singular interpretation, several tens of thousands of technical phrases, with only one unambiguous interpretation each and a collection of syntactic rules.

The CTE checker known as th *ClearChec* tells the Caterpillar authors whether what the write conforms to CTE and also helps them to make it conform if it does not. The machine translation system called the AMT translates the English SGML (Standard Generalized Markup Language) source documents into SGML source documents in other languages. The CTE is concerned with describing the physical aspect of the machinery discussed in the controlled English documents.

2.6 A Requirement Sublanguage for Automated Analysis [Cry95]

This work explains the development of a restricted natural language for expressing requirements. It supports readable specifications that can be analyzed for errors and automatically interpreted. The specific language designed here is for specification of digital systems. Semantics are represented in a type of semantic network constructed o concepts and relations. The semantic basis, consisting of concept and relation types together with semantic patterns is developed from an examination of several formal specification and design notations and natural language statements selected from product descriptions. The syntax of the language is developed from a syntactic analysis of natura language statements and the productions of the grammar are selected to maximize coverage of syntactic structures used in the selection. The grammar developed in this study is used by the CEC system.

Chapter 3

The CEC Design approach

This chapter introduces the theoretical background necessary for understanding the operation of the CEC system. In section 3.1, language theory and grammar are discussed to introduce the reader to controlled English. Section 3.2 gives an overview of the CE parser used by the CEC system and section 3.3 describes the algorithm and approach taken by th CEC system to generate CE comments from ungrammatical comments.

3.1 Language Theory and Gramma

The *alphabet* of a language is the set of all possible indivisible symbols of that language. A *language* is defined over an *alphabet* as a subset of the set of all strings obtained b concatenating one or more symbols from the alphabet. In the CE used here, the alphabet consists of {vocabulary, punctuation, numbers and identifiers} [WinT, LinP, ManM]. An identifier may be a name of a component such as S1 for signal, or a word not found in the vocabulary.

A formal language may be described by a grammar. A *grammar* G is defined as an quadruple G=<V, T, S, P>, where,

V is a finite set of variables (non-terminals),

T is a finite set of terminal symbols also called the alphabet,

S is a special non-terminal that denotes a *sentence* or a *top-level* construct and also represents a well-formed comment in this thesis, and

P is a finite set of productions (grammar rules).

A *grammar rule* or a *production* of a context free grammar (CFG) consists of a non-terminal symbol on the left-hand side, and a sequence of constituents (terminals and non-terminals) on the right-hand side. There are three categories of non-terminals. The firs category includes the high-level non-terminals such as nominals, predicates, adverbials, conjunctions and punctuation (stop, comma). The second category is the intermediate non-terminals that includes noun phrases, active verb sequences and sub-ordinate conjunctions. The third category is the part-of-speech (POS) non-terminals such as nouns, verbs, determiners and prepositions (The entire list of non-terminals can be found in the Appendix B). Terminals are composed of vocabulary and punctuation (‘.’, ‘,’).

3.2 The CE parser

The controlled English (CE) parser consists of two parts, the lexical analyzer and the parser. The lexical analyzer tokenizes the comment entered by the user. The CE parser attempts to form a parse tree for the commen **.**

3.2.1 Lexical analyzer

The ***lexical analyzer*** is responsible for tokenizing the input comment into individual words, numbers and punctuation that are then looked up in the dictionary to determine their corresponding part-of-speech non-terminals. The words in a comment are categorized by one or more part-of-speech non-terminal symbols. For example, the word ‘processor’, machine’, ‘input’, and ‘output’ are categorized as nouns, the words ‘sends’,

and ‘receives’ are categorized as verbs. A dictionary for example comment C2 is shown in Table 3.1. The first column is the token and the second column gives the parts of speech for the token. A token is a symbol of the alphabet, i.e. a word, number or punctuation. Comment C2, has 13 tokens. The output of the lexical analyzer is
“det / noun / verb / det / noun / noun / preposition / det / noun / preposition / det / noun / stop”

The processor sends the output signal to the input of the machine. (C2)

Table 3.1 Sample dictionary

Token	Part of speech
.	stop
input	noun
of	preposition
output	noun
processor	noun
machine	noun
sends	verb
signal	noun, verb, inf(Infinitive)
the	det
to	preposition

3.2.2 Parser

The parser generates one or more parse trees for each comment that is grammaticall correct (a well-formed CE comment) under the CE grammar. A parse tree is a directed tree in which each node is labeled with the left-hand side (variable) of a production and th successor nodes represent the constituents of the right-hand side of a production.

Every sentence in a context-free language has at least one parse tree. If there is more than one parse tree, it indicates an ambiguous grammar. Parse trees do not show the order in which productions are applied. A parse tree can be developed by bottom-up or top-down parsing. The CE parser uses bottom-up parsing. The parser scans the sequence of non-terminals produced by the lexical analyzer and applies the grammar rules repeatedly unti no new non-terminals can be produced. Parsing is successful if one or more 'S' type non-terminals are generated.

The CE parser used in the CEC system forms a chart from which it extracts the parse tree if the comment entered by the user is a well-formed CE comment. The chart generation process is explained below.

Chart generation process:

The CE parser used in the CEC system, first, collects the non-terminals generated by the lexical analyzer and assigns each of them a part number. For example, the node 'earlier' in comment C3 is tokenized by the lexical analyzer into an ' adv' and an 'adj'. These two tokens are given part numbers 1 and 2 respectively. Next, new parts are generated applying the appropriate CE rules on the previously formed parts. For this example, part 3 ('d') is generated by applying the CE rule 'd $\rightarrow$adv' on part number 1. Similarly, part 4 ('adjs') is generated by applying the CE rule ' adjs$\rightarrow$adj' on part number 2. The part, 'the processor' (part 10), is formed by combining parts 5 and 7. The chart generated by the parser for the comment C3 using the CE grammar is shown in Table 3.2 and is illustrated diagrammatically in Figure 3.1, where each node represents the part number in Table 3.2.

Earlier the processor resets the register. (C3)

Table 3.2 Sample chart

Node	Part	Non-terminal	Rule applied	Constituent parts
earlier (chunk)	1	adv	adv → earlier	-
	2	adj	adj → earlier	-
	3	d	d → adv	1
	4	adjs	adjs → adj	2
the	5	det	det → the	-
processor	6	noun	noun → processor	-
	7	head	head→ noun	6
	8	np	np → head	7
	9	n	n → np	8
the processor (chunk)	10	np	np→ det head	5,7
	11	n	n → np	10
resets	12	verb	verb → resets	-
	13	avs	avs → verb	12
	14	pred	pred → avs	13
earlier the processor resets	15	ss	ss →d pred	3,11,14
the processor resets	16	ss	ss →n pred	11,14
the	17	det	det → the	-
register	18	noun	noun → register	-
	19	head	head→ noun	18
	20	np	np → head	19
	21	n	n → np	20
the register	22	np	np→ det head	17,19
	23	n	n → np	22
resets the register (chunk)	24	pred	pred → avs n	13,23
.	25	stop	stop → .	-
earlier the processor resets the register	26	ss	ss → d pred	3,11,24
the processor resets the register	27	ss	ss → n pred	11,24
processor resets the register	28	ss	ss → n pred	9,24
earlier the processor resets the register.	29	s	s → ss stop	26,25
the processor resets the register.	30	s	s → ss stop	27,25
processor resets the register.	31	s	s → ss stop	28,25

The abbreviations for the various non-terminals used in the sample chart in Table 3.2 are shown in Table 3.3.

Table 3.3 Abbreviations used in the sample chart

Abbreviation	Non-terminal
adv	adverb
adj	adjective
adjs	adjective string
d	adverbial
np	noun phrase
n	nomina
avs	active verb sequence
ss	simple sentence
s	simple sentence

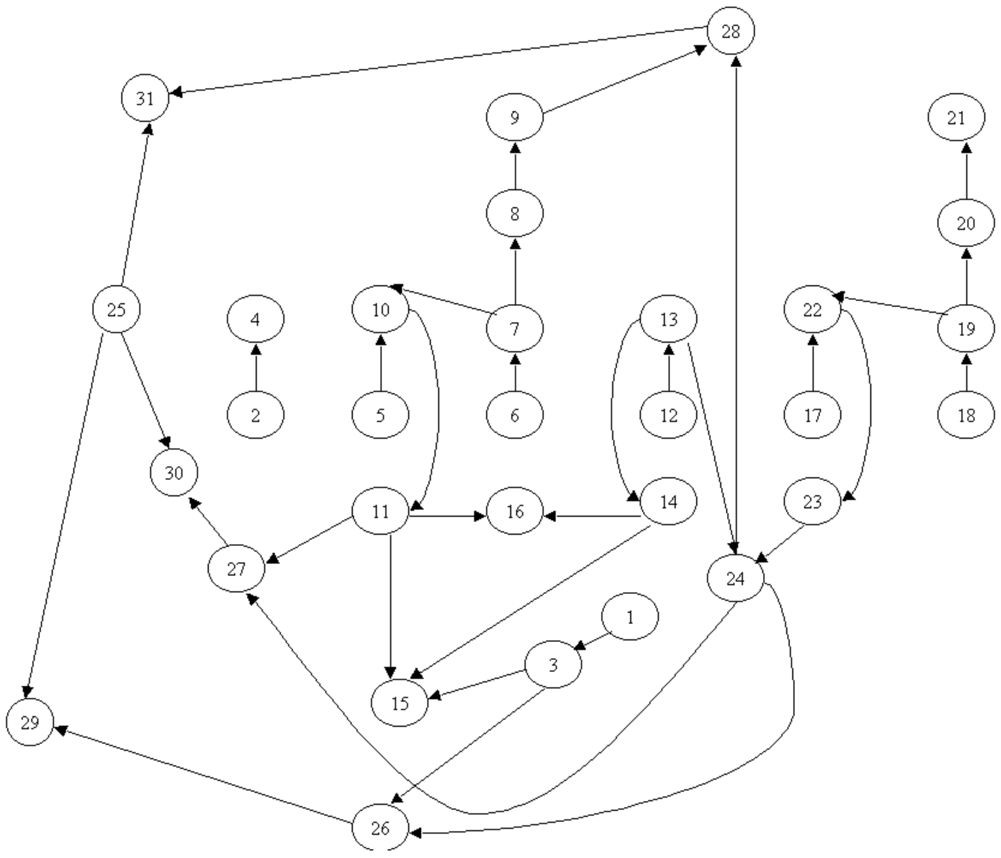


Figure. 3.1 Construction of different parts in the chart generation phase

The parser extracts the parse tree after the chart has been formed. The example commen C3 used here is a well-formed CE comment and a parse tree can be extracted from the chart as shown in Figure 3.2.The part numbers are shown in parentheses.

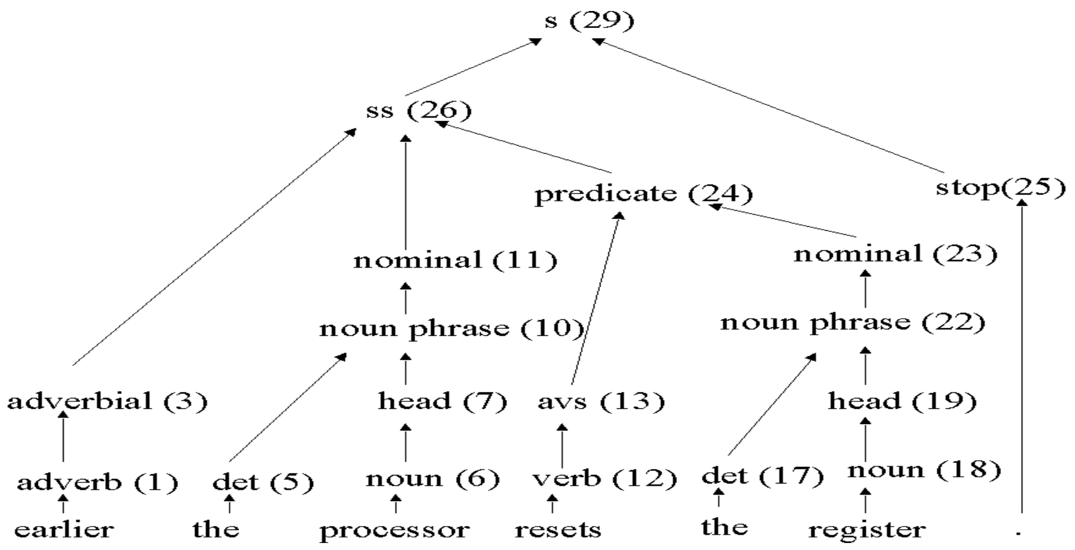


Figure. 3.2 Parse tree formed by using a bottom-up parser.

3.3 Analysis of ungrammatical comments

This analysis phase is required only if the CE parser fails to form a parse tree. This is possible if the comment entered by the user is not a well-formed controlled English comment. The comment analysis phase attempts to form all controlled English comments that are possible from the UC and presents it to the user. The user can choose one of the controlled English comments generated by the analysis phase, or re-phrase the UC. If the user accepts a controlled English comment generated by the comment analysis phase, the CEC system selects the remaining parts of the UC and runs the analysis phase again to determine if it is able to form other controlled English comments. This process continues until no other controlled English comments can be formed.

For the ungrammatical comment C4, the CE comments generated by the system are shown in Table 3.4. Table 3.5 shows the CE comments generated after the user selects the first CE comment in Table 3.4.

A bit INTERRUPT 0 is an interruption signal and, when it is set, an interruption reques signal is applied to the central processing unit. (C4)

Table 3.4 Set of CE comments generated by the CEC system for comments C4

CE Comments
a bit INTERRUPT 0 is an interruption signa
a bit INTERRUPT 0 is applied to the central processing unit
a bit INTERRUPT 0 is an interruption signal and is applied to the centra processing unit
an interruption signal and is applied to the central processing unit
when it is set, an interruption request signal is applied to the central processing unit
when it is set an interruption request signal is applied to the central processing unit

Table 3.5 Set of CE comments generated after the user selects the first CE comment in Table 3.4

CE Comments
an interruption signal and is applied to the central processing unit
when it is set, an interruption request signal is applied to the central processing unit
when it is set an interruption request signal is applied to the central processing unit

Sections 3.3.1 through 3.3.3 explain the various steps involved in generation of CE comments from ungrammatical comments. Section 3.4 explains the interaction process that aids in the generation of another set of CE comments after the user has selected a CE comment from the previously generated set CE comments.

3.3.1 Chunk collecting phase

A chunk is a high level non-terminal of type nominal, adverbial, predicate conjunction or punctuation, that is part of the actual UC under analysis. These chunks are required by the comment generation phase for the generation of CE comments and are present in the chart generated by the CE parser.

The chart produced by the parser is scanned and chunks corresponding to the three non-terminals – *nominal*, *predicate*, and *adverbial* are collected. Nominal chunks that are already accounted for as part of another nominal are ignored. For example, the phrase “A bit INTERRUPT 0 " found in comment C4 is a nominal and "INTERRUPT 0" is a nomina that is ignored as it is part of the nominal "A bit INTERRUPT 0” that has already been collected.

The predicate chunks are then collected in the same fashion as the nominal chunks. Nominal chunks that are part of the predicate chunks are removed from the previously collected list of nominal chunks. In the example discussed above, “is applied to the centra processing unit” is a predicate chunk. The chunk “the central processing unit” is a nomina chunk that is removed from the previously formed nominal chunk list.

The chunk collector then collects all adverbial chunks and those adverbial chunks that ar already part of a larger adverbial chunk are removed. The adverbial chunks that are inside nominal and predicate chunks are also removed. For example, in the chunk "a high signal", "high" is an adverbial chunk that is ignored. The nominal and predicate chunks that ar already part of the larger adverbial chunks are removed from the previously collected list o nominal and predicate chunks.

The chunk collector then examines the comment to see if there are any commas or conjunctions that were not included inside the chunks that have been already selected. If there are any commas and conjunctions that need to be handled, they are collected separately.

It is possible in some situations, that a nominal chunk is also a predicate chunk, then, two separate lists of chunks are formed with one list treating the chunk as a nominal and th other list treating the chunk as a predicate. For example, consider the chunk “signal RD from dmac”. This chunk can be interpreted in two ways depending on the usage of th word ‘signal’ as a verb or as a noun. If the word ‘signal’ is a noun, the chunk under consideration is a nominal, if it is a verb, the chunk is then considered a predicate

The different chunks formed for the complex comment C5 are shown in Table 3.6.

The output signal of the processor is received by the system and when the input signal is received by the system, it goes to th saturation state. (C5

Table 3.6 Different chunks collected in the chunk collection phase

Chunk type	Chunks
nominal(n)	the output signal of the processor it
predicate (pred)	is received by the syste goes to the saturation stat
conjunction (conj)	and
comma (,)	,
adverbial (d)	when the input signal is received by the syste

3.3.2 Ungrammatical comment (UC) structure analysis

This phase processes the chunks collected in the previous phase to determine the chunk types and the location of the chunk in the actual comment. The data collected by the analysis phase for the comment C6 is shown in Table 3.7. The first column is the chunk type, i.e., nominal, predicate, adverbial, conjunction or comma. The second column gives the actual chunks of the comment. The third column is the location (the character location at which the chunk starts in the comment) of the chunks in the ungrammatical commen entered by the user.

The processor tries to send a high input signal and it tries to do it quickly. (C6)

Table 3.7 Data collected by the analysis phase.

Chunk type	Chunks	Location
nomina	the processor	0
predicate	tries to send a high input signal	13
conjuncti	and	45
nomina	it	48
predicate	tries to do it quickly	50

3.3.3 Generation of CE comments

In this phase the CEC generates all possible controlled English comments for a given ungrammatical comment. The information collected by the analysis phase is used by this phase to generate the CE comments. Each controlled English rule for a simple commen is successively applied. The CE comment generation is a recursive process and is discussed below.

The recursive algorithm

This section explains the algorithm used by the CEC system to form controlled English comments from a UC. The algorithm makes use of two tables, table ‘ CommentParts’ and table ‘locPtr’. Table ‘CommentParts’ has three columns. The first column indexes the location of the chunks in this table. The second column is the chunk type of the different chunks in the UC entered by the user. The third column contains the chunks themselves. Table ‘locPtr’ has three columns. The first column is the index of the entries in this table. The second column is the rule size (Rule size is the number of constituents that form the rule. For example the rule size is 2 for the rule s $\rightarrow$ n pred) and the third column contains the location of the non-terminals in ‘Comment Parts’ table. For the example comment C7, the ‘Comment Parts’ contents are shown in Table 3.8, and Table 3.9 shows the contents o ‘locPtr’ for the rule “s $\rightarrow$ n pred”.

Table 3.8 Contents of CommentParts

Ordinal	Chunk type	Chunks
0	nomina	the processor
1	predicate	tries to send a high input signal
2	conjuncti	and
3	nomina	it
4	predicate	tries to do it quickly

Table 3.9 Contents of ‘loctPtr’

Index	Non-terminal of rule	Non-terminal location in CommentParts (Lentries)
0	n	0, 3
1	pred	1, 4

In Table 3.8, the index value of 0 corresponds to the non-termina *nominal* and index value of 1 corresponds to the *predicate* non-terminal. The information in 'locPtr' states that nominals are found at locations 0, 3 and predicates are found at locations 1, 4 in table 'CommentParts'. The recursive algorithm uses this data to form all combinations of nominals and predicates to satisfy the rule s $\rightarrow$ n pred”.

The CEC system uses the recursive algorithm to form CE comments. The recursive algorithm combines the chunks at locations 0 and 1 of the 'CommentParts' table to form the CE comment “The processor tries to send a high input signal”, chunks at locations 0 and 4 to form the CE comment “The processor tries to do it quickly” and chunks at locations 3 and 4 to form the CE comment “it tries to do it quickly”. The comment formed by chunks at locations 3 and 1 is not used as it results in a nominal that occurs in the latter part of the sentence being used as a subject for an earlier predicate.

Table 3.10 Recursive algorithm used for generation of CE comments

```
Recurse(index, prevPos)
// For the first call index = 0, prevPos = -1
// index represents the row number for the ‘loctPtr’ table
{
          string S[x]   // Dynamic array of CE comments being formed for this rule

          Iterator = 0  // Used to iterate through the entries in the ‘Non-termina
                         // location in CommentParts’ column (Lentries) in each row o
                        // the ‘loctPtr’ table.

          While Iterator is less than the number  of entries in the ‘Non-termina
          Location in CommentParts’ column in index’th row of the ‘loctPtr’ table.

                     if location  value a  Lentries [Iterator] is greater than the
                     previous location 'prevPos' of the chunk that was appended
                     to the  new CE comment under construction

                                append to string 'S' the chunk form the ‘CommentParts’
                                table  corresponding to location value indicated by
                                Lentries [Iterator]

                                prevPos = 'Location'
                    endif

                    if index does not point to the last non-terminal (i.e., the last entry in
                    the ‘loctPtr table’) of the rule

                              recurse(index + 1, prevPos) // Call recurse again
                    endif
                    Iterator = Iterator + 1
          Endwhile
}
```

Example:

The controlled English comments formed by the comment generation phase using the recursive function *recurs* and the simple English rules used to form them for th ungrammatical comment C7 are shown in Table 3.11.

When CPU 511 receives the signal 524 from DMAC 512, it saves PC and PSW toward a stack area, and starts the interruption processing program routine (C7)

Table 3.11 Generated CE comments

CE comments generated	**CE Rule used**
when cpu 511 receives the signal 524 fro dmac 512 it saves pc and psw toward a stack area	s→d n pred
when cpu 511 receives the signal 524 fro dmac 512 it starts the interruption processing progra routine	s→d n pred
when cpu 511 receives the signal 524 fro dmac 512 , it saves pc and psw toward a stack area	s→d , pred
when cpu 511 receives the signal 524 fro dmac 512 , it starts the interruption processing progra routine	s→d , pred
it saves pc and psw toward a stack area	s→n pred
it starts the interruption processing program routine	s→d n pred
when cpu 511 receives the signal 524 fro dmac 512 it saves pc and psw toward a stack area and starts the interruption processing program routine	s→d pred conj pred
it saves pc and psw toward a stack area and starts the interruption processing program routine	s→d pred conj pred
when cpu 511 receives the signal 524 fro dmac 512 it saves pc and psw toward a stack area , and starts the interruption processing program routine	s→d pred , conj pred
when cpu 511 receives the signal 524 fro dmac 512 , it saves pc and psw toward a stack area and starts the interruption processing program routine	s→d pred conj pred

3.3.4 Interaction phase

In the interaction phase, the system interacts with the user to help enter controlled English comments. When the user selects a controlled English comment produced by the CE generation phase, the interaction phase collects the chunks that are not present in the controlled English comment selected by the user. For example, if the user entered the ungrammatical comment C8, the information collected by the analysis phase is shown in Table 3.12.

The input signal for the system is the result of the output signal of the signal generator and it tries to generate a high value output (C8

Table 3.12 Information collected in the analysis phase

Chunk	Chunk type	Location
the input signal for the system	nomina	0
is the result of the output signal of the signal generator	predicate	27
,	,	83
and	conj	85
it	nomina	89
tries to generate a high value outpu	predicate	91

If the user selects the controlled English comment “the input signal for the system is the result of the output signal of the signal generator” generated by the CE generation phase, the residual information collected by the interaction phase is shown in Table 3.13.

Table 3.13 Residual information collected by the interaction phase after user has made a selection.

Chunk	Chunk type	Location
,	,	83
and	conj	85
it	nomina	89
tries to generate a high value outpu	predicate	91

These residual chunks are used by the CE comment generation phase to form other possible controlled English comments. The controlled English comment “it tries to generate a high value” was formed from the residual information in Table 3.12, collected by the interaction phase after the user had made a selection.

Orphan Chunks

The interaction phase also controls how the system queries the user for informati regarding *orphan chunks.* An orphan chunk is one that was not a part of any controlled English comment selected by the user and cannot be used to form a CE comment by combining it with other residual chunks. It is possible that after the user has selected from the controlled English comments generated by the CEC there are a few chunks in the original complex comment that are not used in any of the selected controlled English comments. The CEC does not know what to do with these chunks and queries the user for supporting information.

Example:

For the ungrammatical comment C9, the user selects the generated controlled English comment "the main idea of this process is to decide which signal to process". The interaction phase then finds an orphan predicate "generate an appropriate output signal" and queries the user for a nominal to support the predicate chunk. Similarly, the system queries the user for a predicate when it finds an orphan nominal. The above queries are equivalent to the interaction phase querying the user, 'W *hat performs a particular function?'* for a residue predicate, and 'W*hat is the function of this object?'* for a residue nominal. The system processes the orphan chunks in the order in which they occur in the UC entered by the user.

The main idea of this process is to decide which signal to process, and, generate an appropriate output signal (C9)

Summary

This chapter was an overview of the approach taken by the CEC system to generate CE comments from an ungrammatical comment and thus aiding the user to enter CE comments. In the next chapter we will look at the actual implementation in terms of classes for an object oriented implementation. The different classes, their purposes and methods included in them are also discussed.

Chapter 4

Implementation

This chapter describes an object oriented implementation of the CEC system in the C++ programming language. The main classes used in this thesis include CChartFormer – to form charts, CChunk – to form chunks, CLocate - to find the location of the various chunk in the comment, CRuleFinder - to find the appropriate CE rule to be used in the generation of CE comments, CInteract – to interact with the user to negotiate CE comments from an ungrammatical comment (UC). The CECsystem class that uses the above classes. These various classes are described in detail below. Code specific implementation details ar omitted and only important methods and attributes of each class are shown in the class diagrams.

4.1 The Unified Modeling Language (UML)

The various classes of the CEC system are described using a notation called the Unified Modeling Language (UML) [Rational96]. An overview of UML class diagrams is presented here.

In UML class diagrams, a box represents a class. Each box has three partitions. The first entry at the top is the name of the class. Class attributes and class member functions ar listed below the name. The "lock" icon is used to distinguish private attributes from th public attributes.

The classes can have two types of relationships between them. These relationships are th dependency relationship and the aggregate relationship. An arrow represents the dependency relationship between two classes. The tail attaches to the client class and th head of the arrow points to the server class. A dependency relationship is used to denot “using” association between two classes. The aggregate relationship is used to show a part/whole relationship between two classes. The class at the diamond end of the aggregat relationship is called the aggregate class, and consists of instances of the other class.

4.2 Class overview

This section describes each of the primary classes used by the CEC system using a top-down approach. The next subsection describes the main class of the CEC system and it relation to the other classes. The other sub-sections describe the other classes in the CEC system in much detail.

4.2.1 Class CECsystem

This is the main class that uses the other major classes. The CECsystem class constructor first instantiates the C*ChartFormer* class that runs the CE parser to generate the chart. I the comment is ungrammatical, it constructs an instance of the *CChunk* class that is responsible for the selection of the chunks from the previously formed chart to be used in the generation of CE comments. The CECsystem class then instantiates CLocate class that finds the location of the various chunks in the actual comment. The *CRuleFinder* class that is instantiated next, finds an appropriate rule to be used to form a CE commen and generates all possible CE comments. The attribute *AllComments* contains all the CE comments that were generated. The CInteract is instantiated after the user selects a CE comment to gather the required information to form other CE comments. The class diagram for the *CECsystem* representing its relationships to the other classes is shown in Figure 4.1.

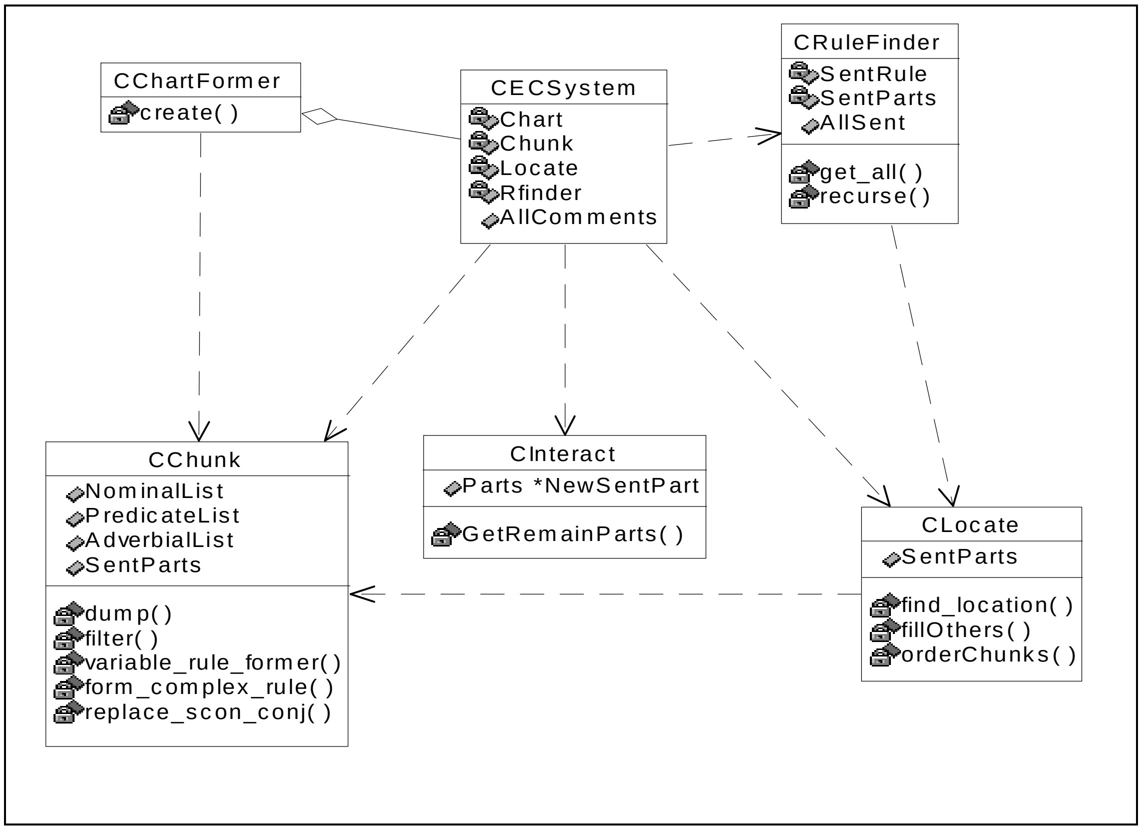


Figure 4.1 UML class diagram representing the relationships between the classes CECsystem, CChunk, CLocate, CRuleFinder and CInteract.

4.2.2 Class CInteract

The *CInteract* class is used in the user interface when the user selects a CE comment generated by the CEC system. This class forms a differen *parts* data structure called *NewSentPar* when a controlled English comment formed earlier by the *CRuleFinder* class has been selected. This is done by the *GetRemainParts* method that removes the chunks that are present in the selected comment from the previously formed *parts* data-structure. The remaining chunks of the *parts* data structure are now used in generating CE comments containing chunks that were not previously used. The class diagram for the *CInteract* class is shown in Figure 4.2.

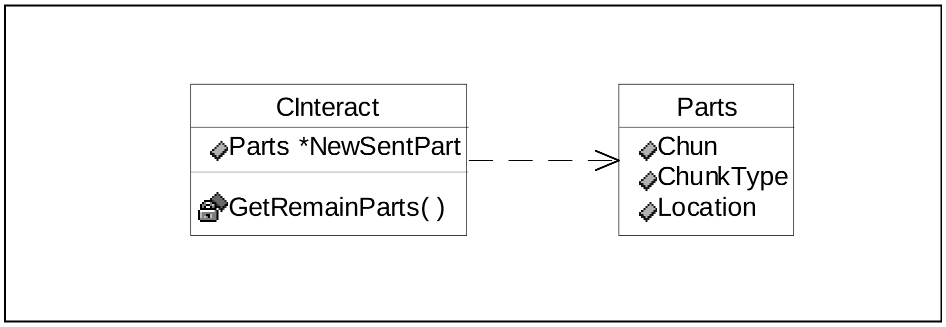


Figure 4.2 UML class diagram representing the relationships between the classes CInteract and Parts.

4.2.3 Class CChartForme

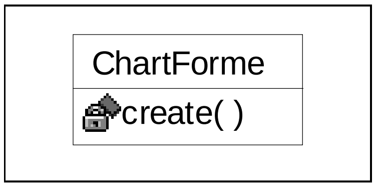


Figure 4.3 UML class diagram of CChartFormer.

This class has a method called *create* and has no attributes. This method is responsible for executing the CE parser that parses the input comment and generates the chart used in the collection of chunks by the chunk class**.**

4.2.4 Class CChunk

This class aids in producing a complex comment structure for an ungrammatical comment (UC). This class has a *dump* method that is used in collecting the high-level chunks from the chart generated by the CE parser. The high-level chunks collected by the *dump* method are then filtered by the *filter* method, which decides whether a chunk should be subsumed by another chunk. For example, a chunk ‘the process’ will be first selected as a nominal. However, if this nominal is a part of the predicate chunk ‘starts the process’, the chunk is treated as part of a predicate and removed from the list o nominals previously collected by the *dump* method. Similarly, nominal and predicate chunks tha are part of an adverbial chunk are removed from the list of nominal and predicate chunk lists previously formed by the *dump* method. The chunks are then segregated into their respective lists of chunk types and the *variable_rule_former* method is called. This method is used when a nominal chunk can also be treated as a predicate chunk and two separate chunk lists have to be formed with one list treating the chunk as a nominal and the other treating the chunk as a predicate (Refer section 3.2.1). The *form_complex_rule* method forms the *parts* data structure called *SentParts*. This data structure contains the various chunks along with the chunk type that constitute the actual comment. The *replace_scon_conj* method is used to collect the phrases that could not be classified and to check if they are a sub-ordinating conjunction or a coordinating conjunction. The selected chunks are stored in the vector of strings 'NominalList', ' PredicateList', or 'AdverbialList' corresponding to their chunk types. This dependency relationship indicates that this class depends on the chart generated by the CChartFormer class. The class diagram for the CC*hunk* class is shown in Figure 4.4.

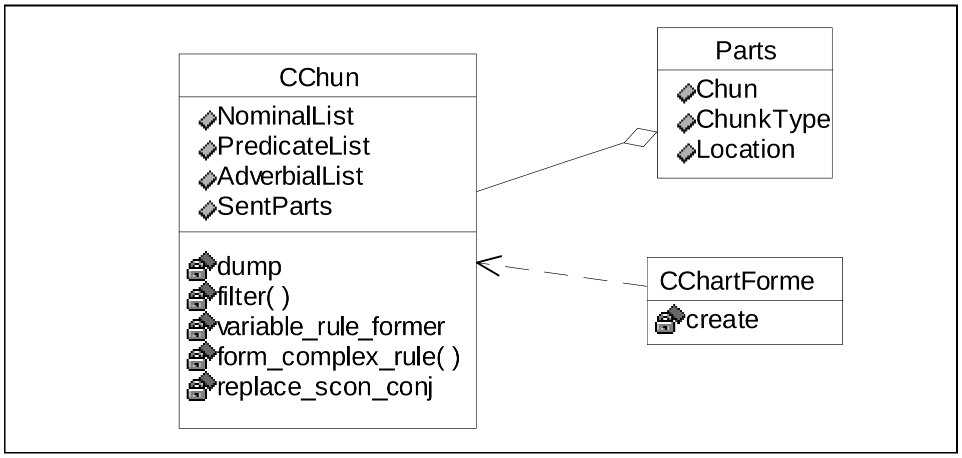


Figure 4.4 UML class diagram representing the relationships between classes CChunk, CChartFormer and Parts.

4.2.5 Class CLocate

The *CLocate* class is responsible for determining the location of the chunks in the ungrammatical comment (UC). This is done by the *find_location* method. The chunks are then placed in the *parts* data-structure called *SentParts* in the order in which they occur in the original complex sentence by the *order_chunks* method. The *FillOthers* method is used to substitute a non-terminal for the chunks that could not be determined by the *CChunk* class. For example, if there is nominal “process” that has not been used in the chunks formed by the *CChunk* class, the *FillOthers* method classifies this as a nominal and records the location of the chunk in the *parts* data-structure. This class depends on the *CChunk* class for the *parts* data structure that is used here. The class diagram for the *CLocate* class is shown in Figure 4.5.

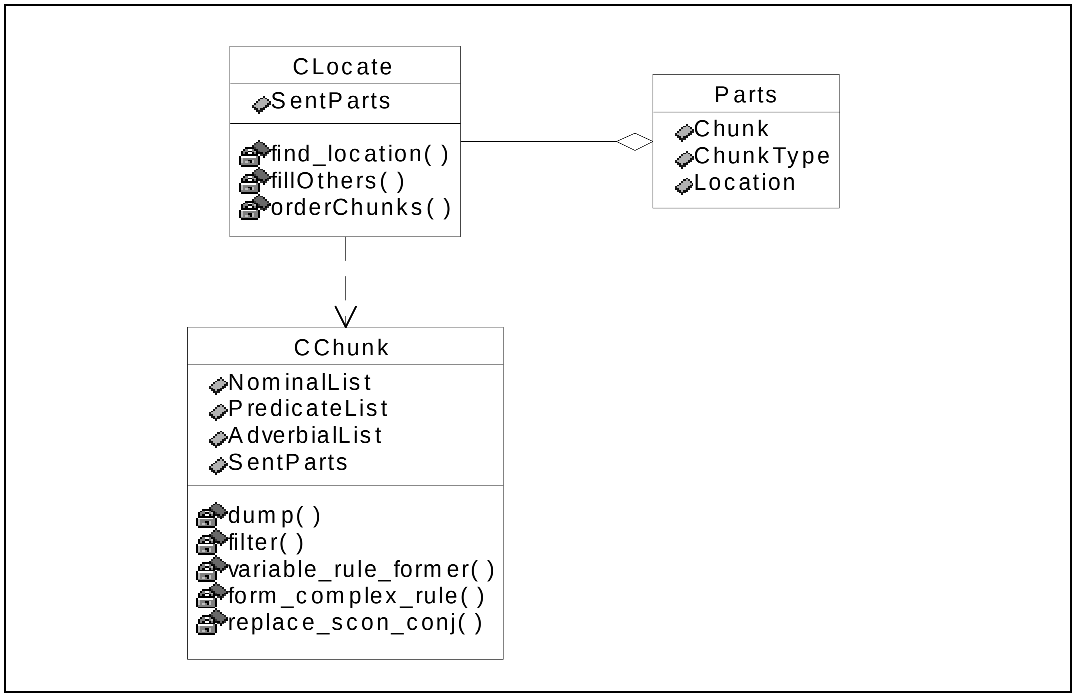


Figure 4.5 UML class diagram representing the relationships between the classes CLocate, CChunk and Parts.

4.2.6 Class CRuleFinde

This class utilizes the *parts* and *rule* data structure to form all possible controlled English comments. The *rule* data structure contains the list of all the CE rules for a simple comment. It implements a recursive method called *recurse* that recursively applies controlled English rules for a simple comment to the *parts* data structure called *SentParts* to arrive at different possible controlled English (CE) comments. The *recurse* method is called by the *get_all* method for each CE rule that is applied for a simple comment. The attribute *all_sent* is a list of all the CE comments that were generated. The filter method removes the CE comments that occur more than once in *all_sent*. The class diagram for the class *CRuleFinder* is shown in Figure 4.6.

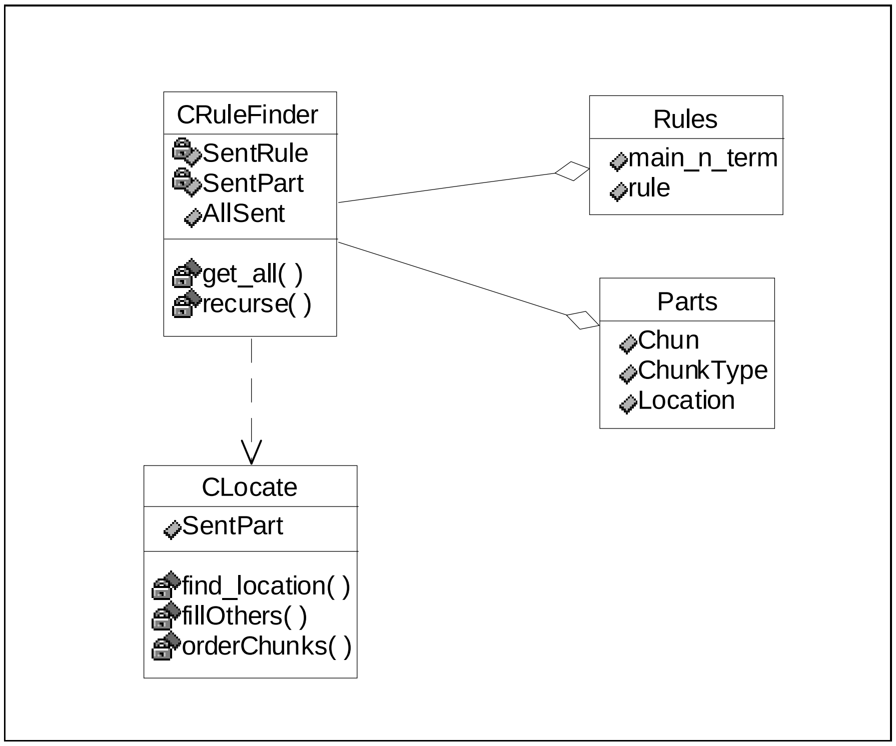


Figure 4.6 UML class diagram representing the relationships between the classes CRuleFinder, CLocate, Parts and Rule.

Chapter 5

User Interface

A graphical user interface has been developed for the CEC system that interacts with the user to select the CE comments generated by the system, help the user to enter CE comments and query the user for information regarding orphan non-terminals. The User interface contains two parts, the Interaction dialog (Figure 5.1) and the Help dialog (Figure 5.3) that helps the user to enter a CE sentence**.**

5.1 Interaction dialog

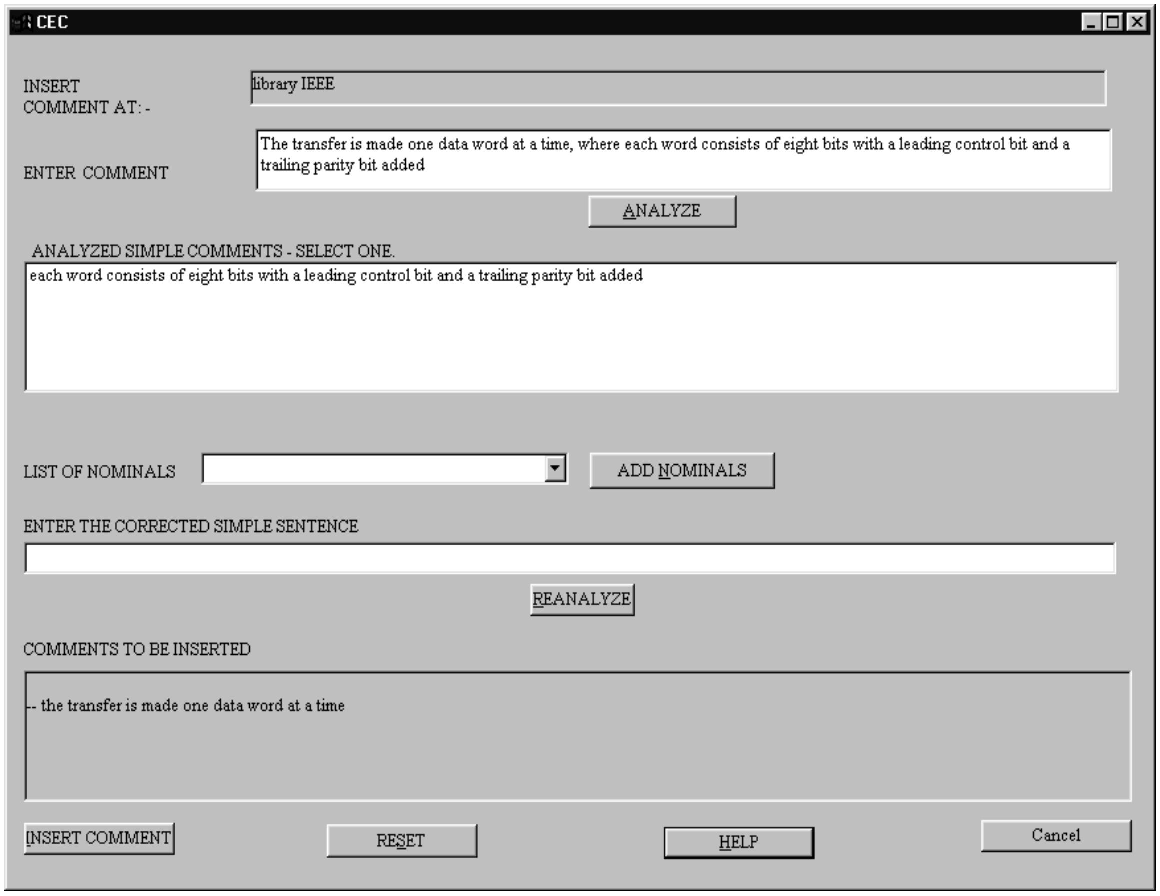
The flow of events that take place while using the CEC user interface are as follows:
The user selects the code region where he/she wants to insert the comment. Next the user enters a comment in a text box of the GUI. The system then analyses this comment. The comment entered by the user is of the controlled English form if the CE parser is able t form a parse tree for the given comment. If the user comment is already in controlled English form, the user is informed that the comment is of controlled English form and the comment is inserted into the code. If the CE parser fails to parse, the comment is classified as an ungrammatical comment and subjected to further analysis. Next, the system sends the ungrammatical comment to the complex structure analysis phase. This phase locates the chunks of the comment and forms the complex structure. The CE comment generator then acts on this complex structure and generates the CE comments that are given to the user. The user now selects one of the various CE comments produced by the system. Once the user selects one of the comments, the system attempts to form other CE comments from the remaining chunks of the actual comment. The user is provided with the new set of CE comments, from which he/she can make another selection of a CE comment. This process continues until the system cannot form further CE comments, or the user decides that the selected CE comments are adequate. If the system is unable to form any more CE comments, and there are still parts of the origina comment that have not been handled yet, the system classifies them as '*orphan non-terminals*'. The *orphan* non-terminals are of three types – predicate, nominal, and

adverbial. If the system finds an '*orphan adverbial'*, it searches for CE comments within the adverbial. In the case of *'orphan predicates'*, the User Interface requests the user for a subject to support the orphan predicate. The system also provides the user with a list of nominals that helps the user to decide which nominal should support the residual predicate. In the case of an orphan nominal, the system requests the user for a predicate to support the orphan nominal.

For example, if the user had entered the ungrammatical comment C10, the system will provide the user with a CE comment "The transfer is made one data word at a time". If the user selects this CE comment, the remaining part of the original comment that has not been handled is the adverbial "where each word consists of eight bits with a leading control bit and a trailing parity bit added". The system then finds the CE comment, "each word consists of eight bits with a leading control bit and a trailing parity bit added", which is again offered to the user.

The transfer is made one data word at a time, where each word consists of eight bits wit a leading control bit and a trailing parity bit added (C10)

Hence the CE comments equivalent to the original ungrammatical comment are "The transfer is made one data word at a time" and "Each word consists of eight bits with a leading control bit and a trailing parity bit added". The interaction dialog box for the above example is shown in Figure 5.1.



INSERT COMMENT AT: - library IEEE

ENTER COMMENT: The transfer is made one data word at a time, where each word consists of eight bits with a leading control bit and a trailing parity bit added

ANALYZED SIMPLE COMMENTS - SELECT ONE.: each word consists of eight bits with a leading control bit and a trailing parity bit added

COMMENTS TO BE INSERTED: -- the transfer is made one data word at a time

Figure 5.1 Interaction dialog box

The sequence of events explained above are represented in the collaboration diagram shown in Figure 5.2. Each event in the sequence is indicated by its step number.

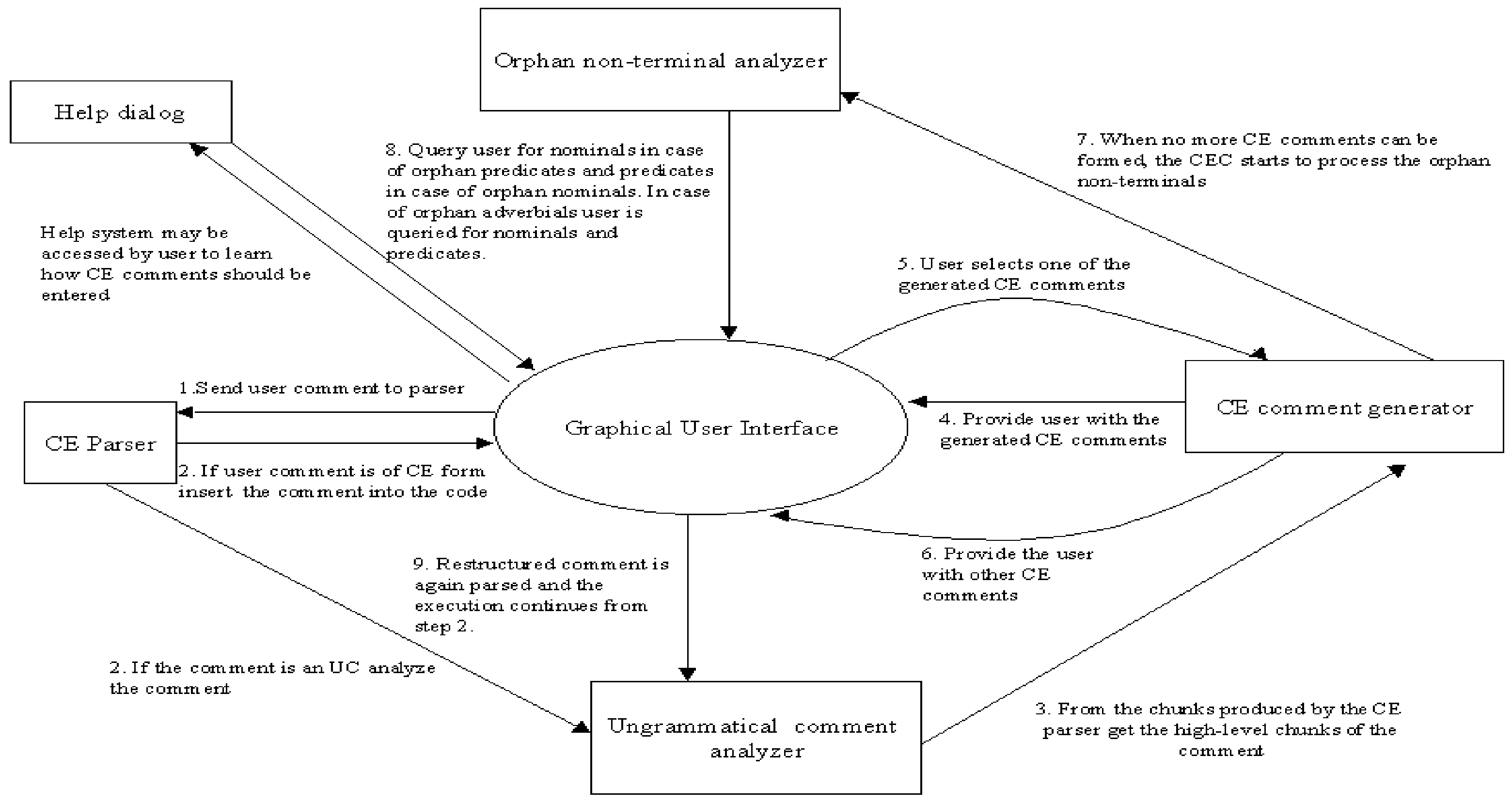
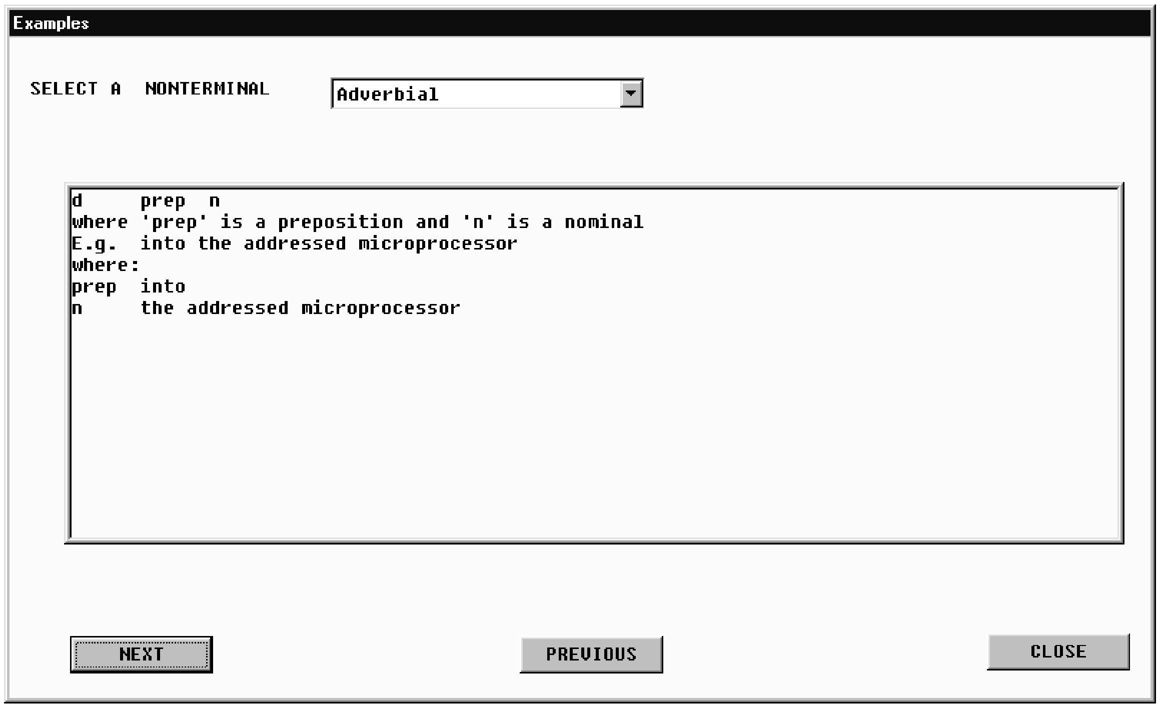

Orphan non-terminal analyzer
Help dialog
8. Query user for nominals in case of orphan predicates and predicates in case of orphan nominals. In case of orphan adverbials user is queried for nominals and predicates.
7. When no more CE comments can be formed, the CEC starts to process the orphan non-terminals
Help system may be accessed by user to learn how CE comments should be entered
5. User selects one of the generated CE comments
1.Send user comment to parser
CE Parser
Graphical User Interface
4. Provide user with the generated CE comments
CE comment generator
2. If user comment is of CE form insert the comment into the code
6. Provide the user with other CE comments
9. Restructured comment is again parsed and the execution continues from step 2.
2. If the comment is an UC analyze the comment
Ungrammatical comment analyzer
3. From the chunks produced by the CE parser get the high-level chunks of the comment

Figure 5.2 Sequence of events in the CEC system

5.2 Help Dialog Box

The help dialog box is used to teach the user how different chunks, such as nominals, predicates, and adverbials are formed and how they should be used to form a CE comment. The user selects a non-terminal from the combo box. The example explanation is then shown in the text window. The example presents the user with one of the CE comment rules, followed by an example CE comment and parts of the comment tha correspond to the parts of the rule. The buttons, “NEXT” and “PREVIOUS” buttons allow the user to view the next or previous example for other CE comment rules.



```
d      prep  n
where 'prep' is a preposition and 'n' is a nominal
E.g.  into the addressed microprocessor
where:
prep  into
n      the addressed microprocessor
```

Figure 5.3 Help Dialog Box

Chapter 6

Experiments

The CEC system was tested to determine how users adapt to writing CE comments with the help of the CEC system. Each subject was given a VHDL program to comment using the CEC system. A total of five subjects were used and the results are shown in Appendix B. Four different VHDL models were used for these experiments. Each of the five subjects was given one of the three different VHDL models: a Histogram generator, a Johnson counter, a Traffic light controller and the ALU model. These four VHDL models are shown in Appendix D along with the comments entered by the five subjects. The subject (subject 5) discussed in this chapter was given a ‘Histogram generator’ model. One set of results for the Histogram generator model is discussed in detail here, and the others are summarized. The subjects selected for these experiments had English as their second language.

6.1 Data collection

The CEC system interaction in each experiment was logged in a file. The log file contains information on the time taken by the user to arrive at a CE comment from an ungrammatical comment, the number of unknown words in the dictionary that were used in the comment, followed by the total number of words in each comment and the number o passes (trials) taken by the user before arriving at a CE comment from an ungrammatica comment. An example of the entry in the log file when the user enters a CE comment is shown in Figure 6.1. Figure 6.2 shows an example entry in the log file when the user enters an ungrammatical comment (UC).

```
**************************************************
Selected code to comment on   : 10/11/2000    at 21:0:56
VHDL model :   Histogram generator
**************************************************


VHDL code selected :
                constant MAXADDR : INTEGER := 2**ADDRLEN - 1;
Comment entered by the user :
           the architecture has a few  constants needed for the logic design
Comment Analyzed at : 21:1:28
Number of words not found in the dictionary : 1
Total number of words : 12
The Comment is of Controlled English form
The Comment has been inserted Comments Inserted into the code at : 21:1:31
```

Figure 6.1 Log file entry when the subject enters a CE comment.

```
**************************************************
Log Started on   : 10/11/2000    at 20:49:25
**************************************************

**************************************************
Selected code to comment on   : 10/11/2000    at 20:49:40
VHDL model :   Johnson counter
**************************************************
VHDL code selected :

            library IEEE;
            use STD.textio.all;
            use IEEE.STD_LOGIC_MISC.all;
            use IEEE.STD_LOGIC_1164.all;
            use IEEE.STD_LOGIC_ARITH.all;
            use IEEE.STD_LOGIC_UNSIGNED.all;

Comment entered by the user :
        The declarations below, are needed for including the various componen
libraries, which contain the various component declarations

Comment Analyzed at : 20:50:35


Number of words not found in the dictionary : 3


Total number of words : 20

Selected the CE comment : " the declarations are needed for including the various
component libraries " at : 20:50:44

Restructured Comment  :
        these libraries contain the various component declarations

Comment restructured at  : 20:51:18


Number of words not found in the dictionary : 2


Selected the CE comment : " these libraries contain the various componen
declarations " at : 20:51:20

Comments Inserted into the code at : 20:51:21
**************************************************
```

Figure 6.2 Log file entry when the subject enters an ungrammatical comment.

The subject's level of adaptation at entering CE comments was evaluated by determining the reduction in time taken by the user to arrive at a set of CE comments equivalent to the original ungrammatical comment, number of unknown words per comment, number of words per comment and number of passes (number of times a comment has to be re-analyzed) a UC undergoes for an equivalent set of CE comments to be generated.

6.2 Experiment results

The results obtained from the log file that is generated as the subject comments the VHD code using the CEC system are shown in Table 6.1. The first column is the comment number. This subject has entered 22 comments. The second column indicates the time taken by the subject from the moment when the UC is entered to the moment an equivalent set of CE comments is arrived at. The third column shows the number of unknown word used in each comment. The fourth column gives the total number of words used in each comment. The fifth column gives the number of passes per UC that were required to arrive at the equivalent set of CE comments. A pass count of 0 indicates that the user's comment was originally in the CE form.

Table 6.1 Results collected from log file

Comment Number	Time (Sec)	Unknow words	Total no. of words	No. of Passes
1	117	3	19	1
2	69	4	27	1
3	148	1	22	2
4	29	2	10	0
5	159	4	14	3
6	38	0	5	0
7	35	1	12	0
8	21	2	7	0
9	69	2	11	2
10	63	1	13	2
11	41	0	12	0
12	13	0	8	0
13	16	0	11	0
14	23	2	12	0
15	24	2	16	0
16	27	0	11	2
17	17	1	11	0
18	37	0	14	0
19	84	2	15	4
20	29	2	10	0
21	21	2	13	0
22	26	0	11	0

It can be observed in the above table that the user requires more passes towards the beginning of the experiment than towards the end, and the number of unknown words decreases towards the end of the experiment. Also, the number of words used per comment starts in the twenties and settles at 11 – 16 words towards the end of the experiment. These observations show that the subject's performance at writing CE comments improves as time spent using the CEC system increases.

6.3 Result Analysis

In this section a systematic study of how one subject performed during the experiment is described in detail. The example discussed in this section is a Histogram generator model. Two types of plots are presented, one set of plots with the values as shown in Table 6.1, and the other set of plots depicting online performance versus the comment number. Online performance is the average of all function evaluations up to the present, including the current trial (current comment).

Pon = (1/g) * sum (average g), (1)

where Pon is the online performance value and 'g' is the trial. The online performance values for the results shown in Table 6.1 are tabulated in Table 6.2. Online performance smoothens the data and helps explore trends.

Table 6.2 Online performance values for the data collected from the log file

Comment No.	Time (Sec)	Unknown words	Total no. of words	No. o Passes
1	117	3	19	1
2	93	4	23	1
3	111	3	23	1
4	91	3	20	1
5	104	3	18	1
6	93	2	16	1
7	84	2	16	1
8	76	2	15	1
9	75	2	14	1
10	74	2	14	1
11	71	2	14	1
12	66	2	13	1
13	62	2	13	1
14	59	2	13	1
15	57	2	13	1
16	55	2	13	1
17	53	1	13	1
18	52	1	13	1
19	54	1	13	1
20	52	1	13	1
21	51	1	13	1
22	50	1	13	1

6.3.1 Normal performance Plots

Figure 6.3 shows that the time spent by the subject in arriving at a CE comment varies significantly from the beginning of the experiment to the end of the experiment. The time taken to arrive at CE comments decreases after the mid-region, i.e., around the 12 th comment. The plot shows a number of spikes near the start of the experiment. These are attributed to the subject's trying to adapt to entering CE comments with the help of the CEC system. The spikes smoothen out with time. There is a spike at the 19 th comment due to a complex VHDL code segment that the subject has tried to comment. This VHDL code segment involves the assertion of the various variables and wait statements involved in the operation of a Histogram generator.

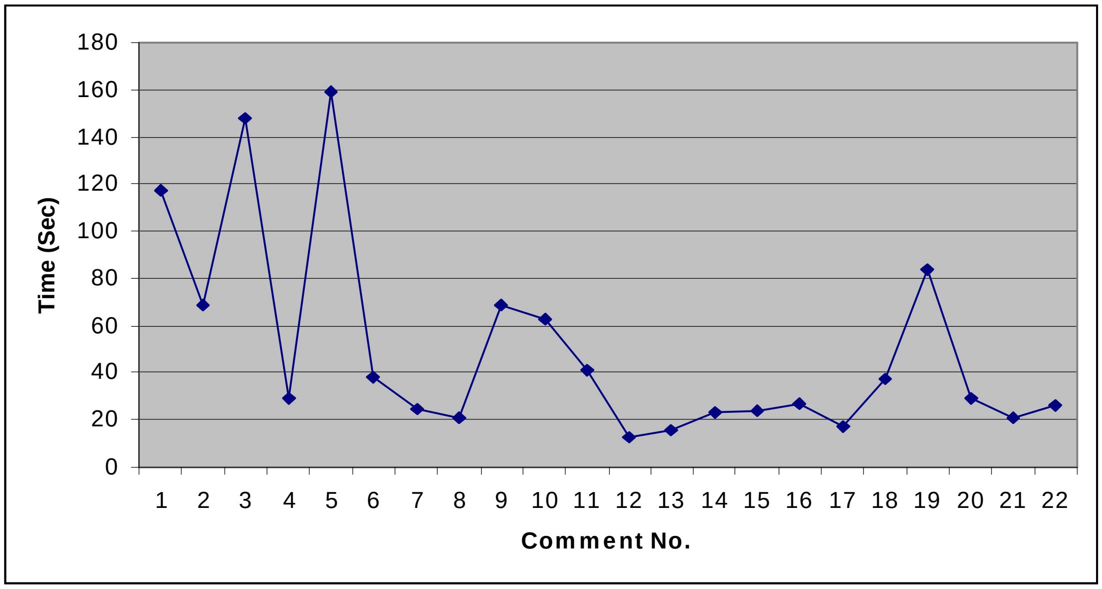


Figure 6.3 Plot showing time taken in seconds versus the comment number

Figure 6.4 shows the number of unknown words versus the comment number. The average of the number of unknown words before the 10 th sentence is more than the number of unknown words after the 10 th sentence, which is almost the mid-region of the plot. The number of unknown words is seen to decrease as the subject's experience in using the CEC system increases. The unknown words used by the subject are identifiers of the form 'P1', 'C10', etc., that are not in the dictionary or words that are not allowed by the CE vocabulary. Examples of unknown words used by this subject for one of the comments are ‘CHIP-SELECT’ and ‘asserted’.

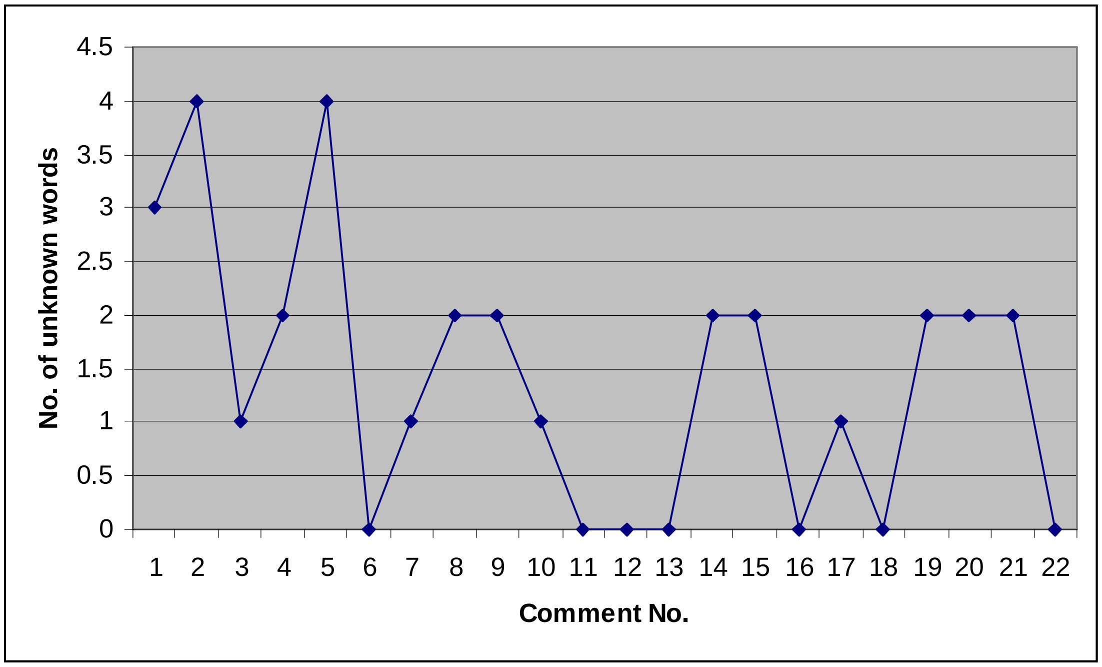


Figure 6.4 Plot showing number of unknown words versus the comment numbe

Figure 6.5 plots the total number of words in each comment entered by the user against th comment number. The total number of words in each comment entered by the user decreases with increase in usage of the CEC system. At the beginning of the experimen , the number of words in the comment are higher than at the end where the number of words in each comment fluctuates around eleven words. The number of words in the comment is higher at the start of the experiment because the subject was trying to explain the VHD model and trying to accustom to writing CE comments. The VHDL model used by this subject was a histogram generator. The average number of words in the first half of all the comments is more than the average number of words in the second half of the comments.

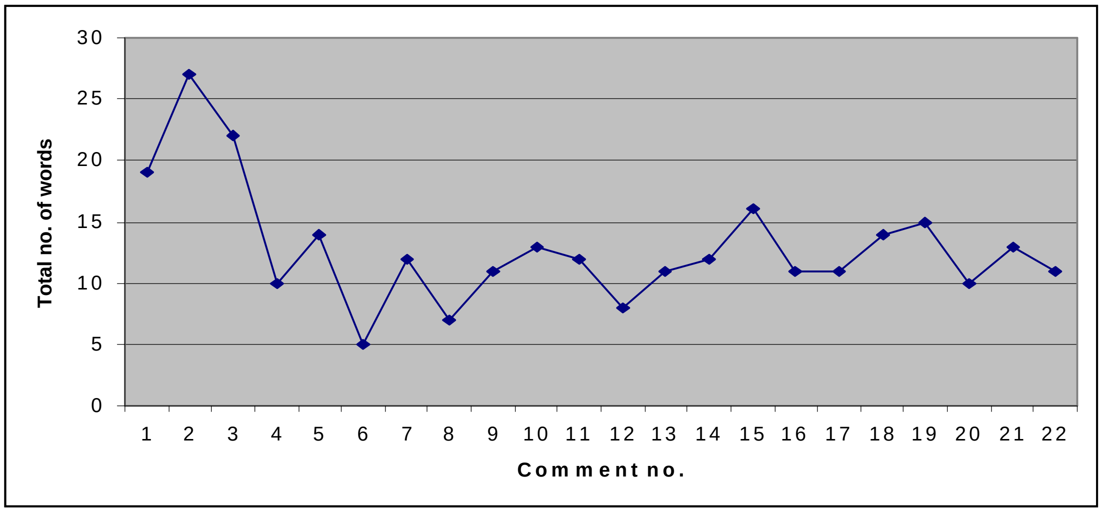


Figure 6.5 Plot showing total number of words versus the comment number

Figure 6.6 gives the number of passes required for each comment entered by the user to arrive at the set of CE comments equivalent to the original ungrammatical commen versus the comment number. The average number of passes before the mid-region of the plot, i.e., around the 11th comment, is more than the average number of passes after the 11th comment. There is one spike in the plot around the 19th comment that indicates the highest number of passes in this experiment. This spike is attributed to the subject trying to comment the complex VHDL code segment that involves the assertion of the various variables and wait statements needed for the operation of a Histogram generator.

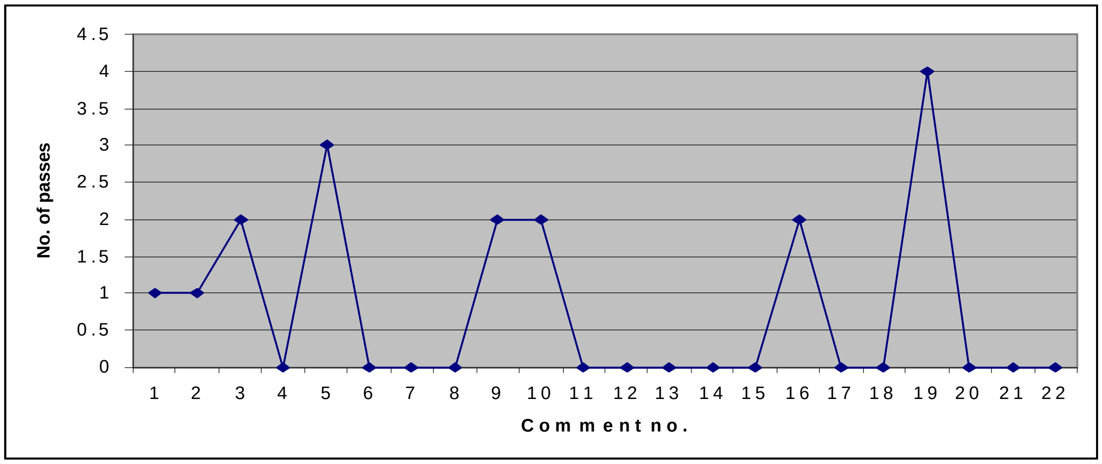


Figure 6.6 Plot showing number of passes versus the comment numbe

6.3.2 Online performance plots

In the plot shown in Figure 6.7, it can be seen that the time taken for the comments to reach their CE form is found to steadily decrease. The average performance towards the end the curve smoothens out to an almost steady value of around 50 seconds per comment. This is a significant improvement over the time of 120 seconds at the start o the experiment. The subject's performance at writing CE comments improves with time spent using the CEC system.

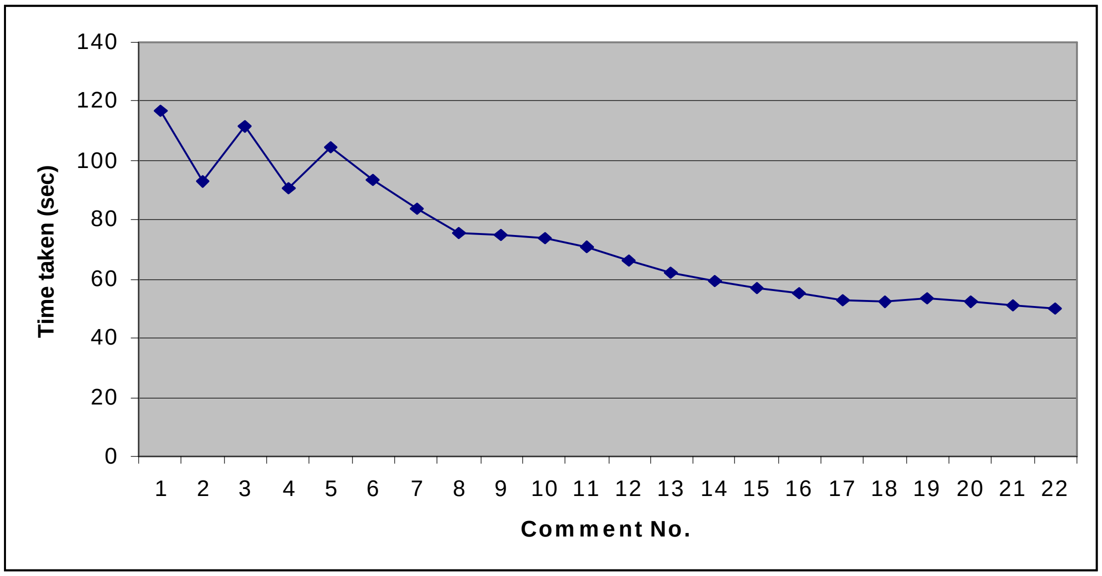


Figure 6.7 Plot showing time taken in seconds versus the comment numbe

Figure 6.8 plots the number unknown words in each comment entered by the subject against the comment number. The curve can be seen to smoothen out to almost a straight line towards the end of the experiment. The number of unknown words in the user comment fluctuates between 3 and 4 at the start and reduces to around 1 or 2 words near the end. This helps us to conclude that the average performance of the subject at writing CE comments increases due to the smaller number of unknown words being used near the end.

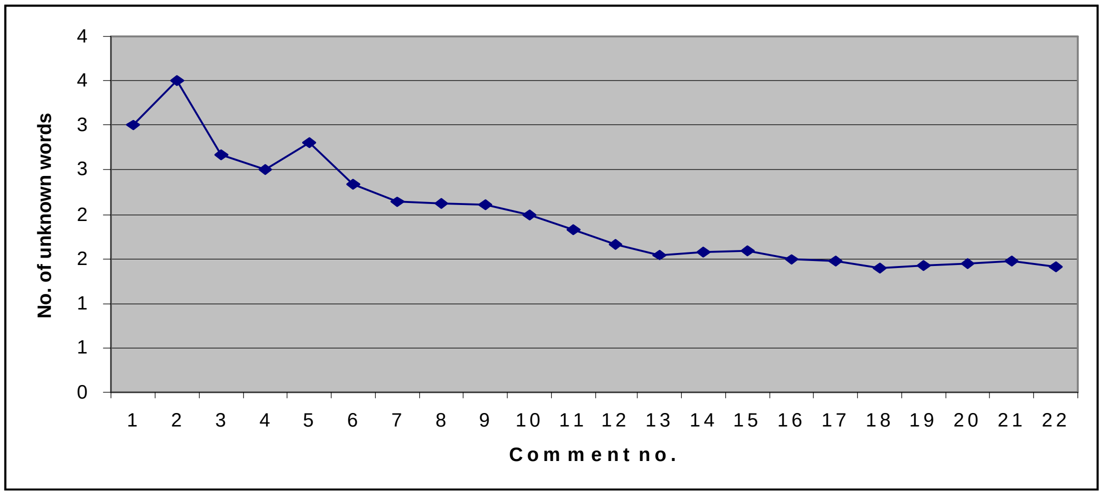


Figure 6.8 Plot showing number of unknown words versus the comment numbe

Figure 6.9 shows the total number of words in each comment entered by the user plotted versus the comment number. At the start of the experiment, the average number of words in the comment is higher than near the end where it smoothens to a low value of around 13 words. The user's performance at entering CE comments is thus seen to increase as a smaller comments tends to have a higher probability of being a CE comment than a larger comment.

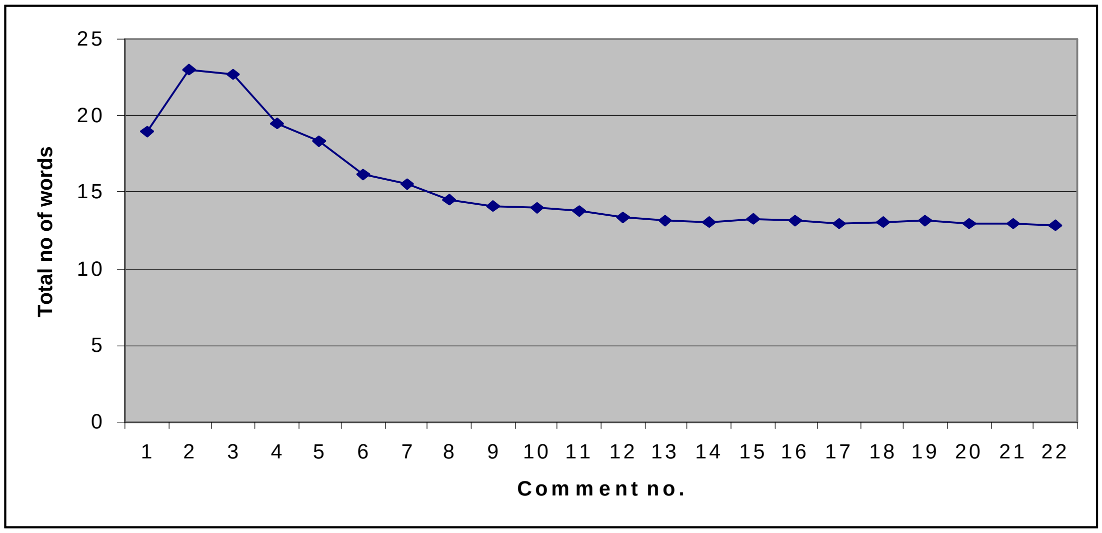


Figure 6.9 Plot showing total number of words versus the comment number

Figure 6.10 plots the number of passes for each comment entered by the user before the set of CE comments equivalent to the ungrammatical comment is arrived against the comment number. The plot shows a number of spikes. These spikes are due to the actual values tha are plotted (not rounded). The values in the Table 6.2 are rounded. There is a stead decreas in the number of passes as the user moves towards the end of the experiment. This shows that the average number of passes to arrive at a set of CE comments from an ungrammatical comment decreases with time.

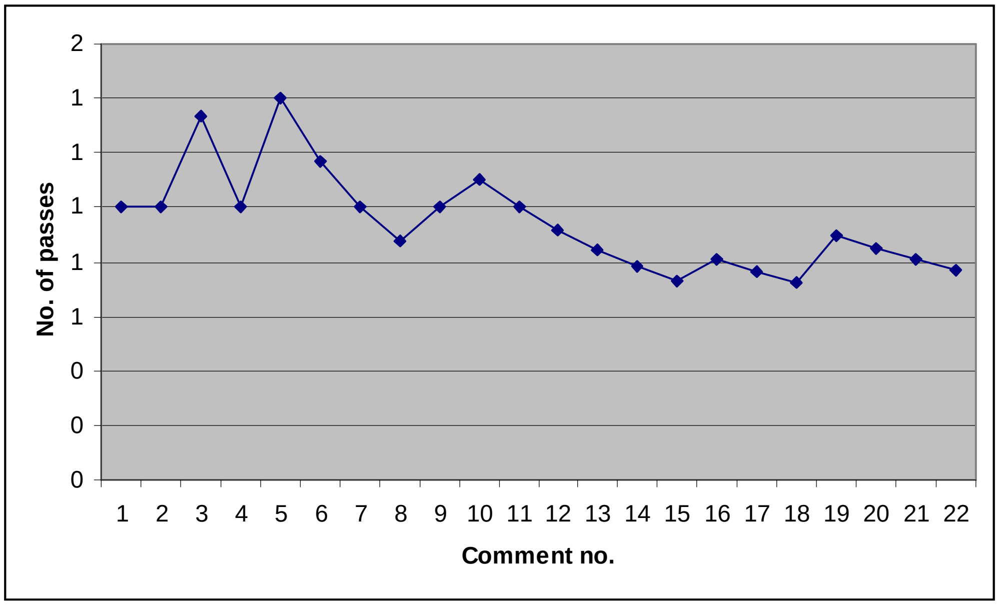


Figure 6.10 Plot showing number of passes versus the comment numbe

6.4 Summary of experiments

The data collected from the subject's experiments and the information obtained from the analysis of the different plots indicate an increase in user performance at writing CE comments as time spent in using the CEC system increases. The plots obtained for the experiments done by the other subjects are available in Appendix B. The various plots of the different subjects are similar and there is a trend of the subject using the CEC syste getting accustomed to use CE comments towards the end of the experiments. An ensemble plot (average) of all the five subjects for the normal performance is discussed below.

6.4.1. Ensemble plots

This section discusses the ensemble plot (average) of all the five subjects for the normal performance. The ensemble plot shows a similar trend of the subject's ability in writing CE comments improving with time using the CEC system.

Figure 6.11 shows that the time spent by the subject in arriving at a CE comment varies significantly from that the beginning of the experiment to the end of the experiment. The time taken to arrive at CE comments gradually decreases with a few spikes in between comment. The plot shows a number of spikes near the start of the experiment. These spikes were as a result of the subjects trying to adapt to entering CE comments with the help of the CEC system. The spikes smoothen out with time. The spike around the 9th comment is due to a complex VHDL code segment that the subjects have tried to comment.

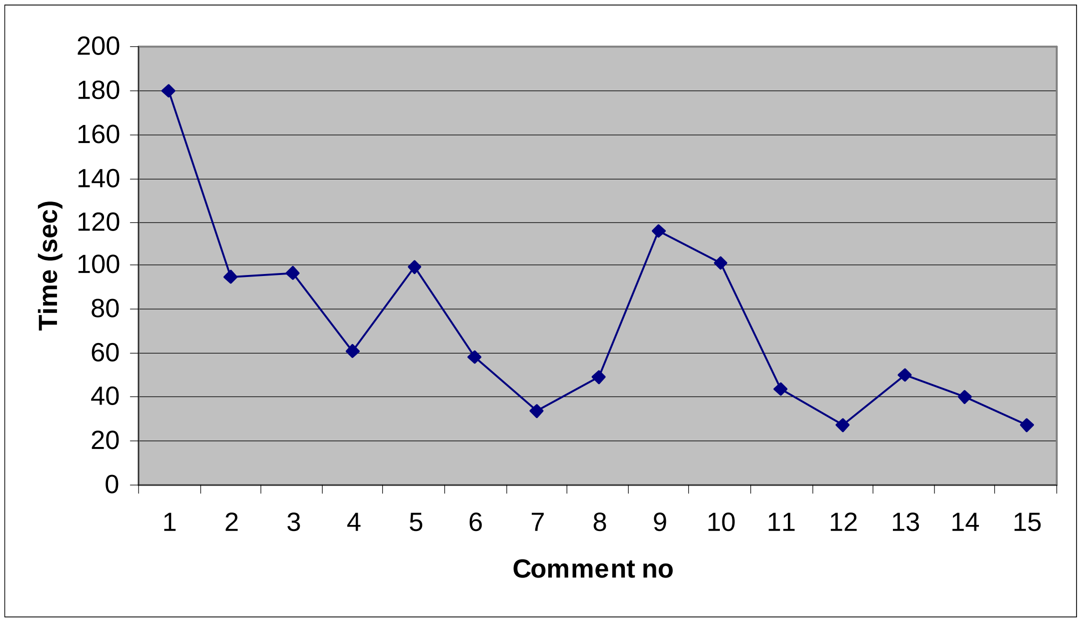


Figure 6.11 Plot showing time taken in seconds versus the comment numbe

Figure 6.12 shows the number of unknown words versus the comment number. The average of the number of unknown words before the 9th sentence is more than the number of unknown words after the 9th comment. The spikes in this plot are due to the identifiers used by the subjects to explain the various variables used in the VHDL models. The number of unknown words is seen to decrease as the subject's experience in using the CEC system increases.

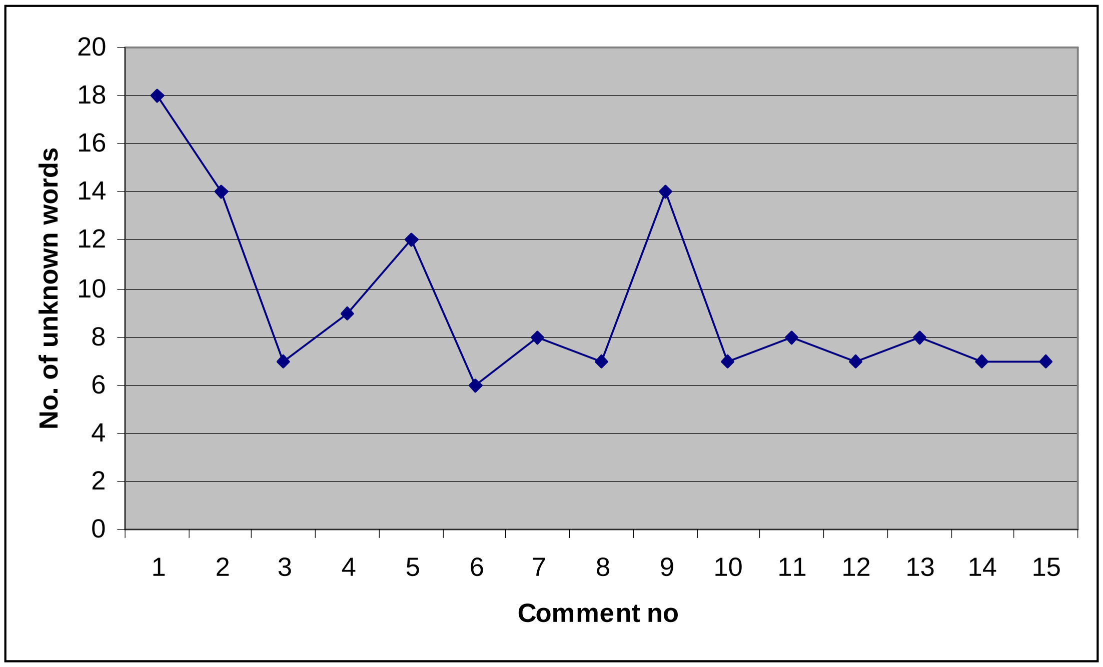


Figure 6.12 Plot showing number of unknown words versus the comment numbe

Figure 6.13 plots the total number of words in each comment entered by the user against the comment number. The total number of words in each comment entered by the user decreases with increase in usage of the CEC system. At the beginning of the experiment, the number of words in the comment are higher than at the end. The number of words in the comment is higher at the start of the experiment because the subjects were trying to explain the VHDL model and trying to accustom to writing CE comments. The spike around the 9th comment is attributed to a complex functionality in the VHDL model tha the subjects were trying to comment. The number of words used by the subjects decreases towards the end, indicating that the users ability to write CE comments improves with time spent in using the CEC system.

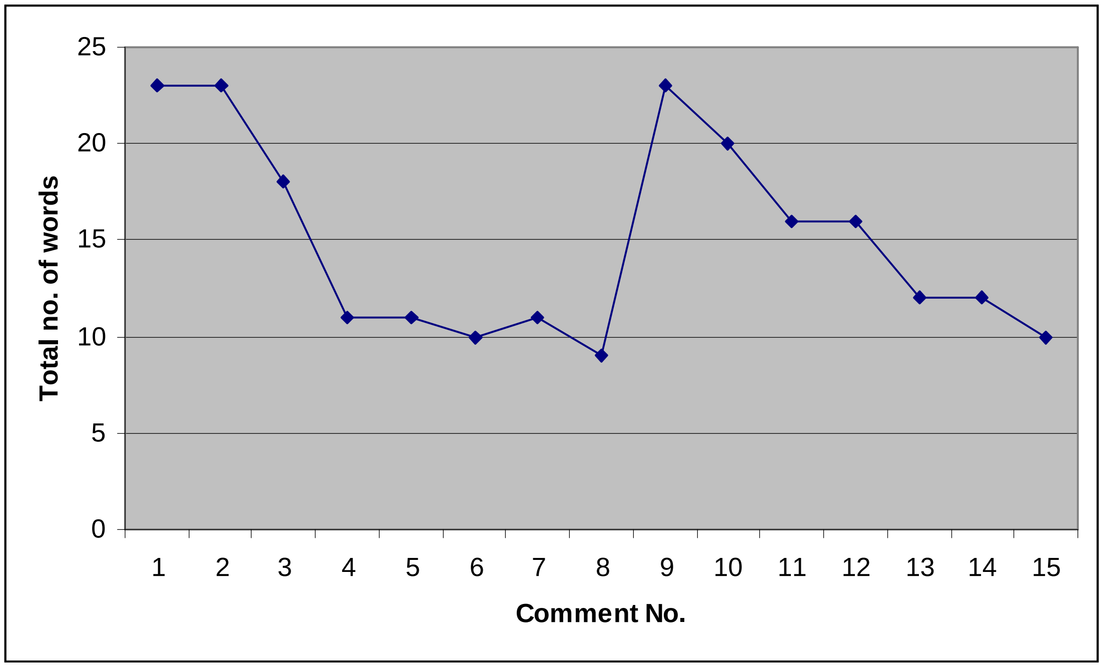


Figure 6.13 Plot showing total number of words versus the comment number.

Figure 6.14 gives the number of passes required for each comment entered by the user to arrive at the set of CE comments equivalent to the original ungrammatical commen versus the comment number. The average number of passes before the mid-region of the plot is more than the average number of passes after the mid-region. The spikes in the plots are due to the complex comments entered by the subjects for complex VHDL code, which in turn required more passes to arrive at equivalent CE comments.

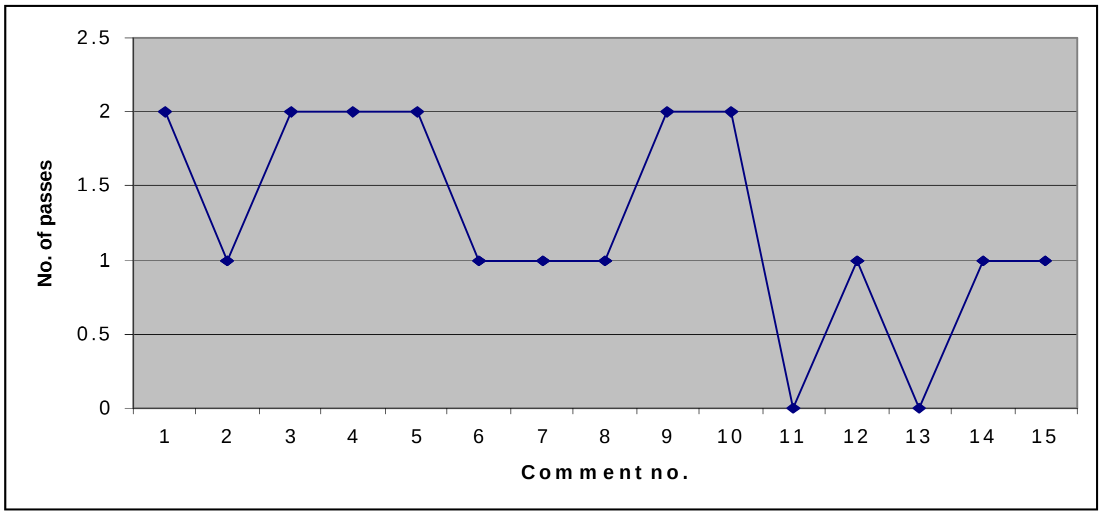


Figure 6.14 Plot showing number of passes versus the comment number

6.4.2 Conclusion

The normal performance analysis shows that the user performance at using the CEC system increases from the first half of the experiments towards the second half. The online performance indicates that the average performance of the user increases graduall with time as the subject’s experience in using the CEC system increases. The methods that might be used to increase the efficiency of the CEC system in aiding the user to enter CE comments are discussed in the next chapter.

Chapter 7

Conclusions and Extension

This chapter summarizes the advantages and disadvantages of the CEC system developed for aiding users in writing CE comments for their VHDL models. Some suggestions to improve the system performance are also recommended.

7.1 System advantages

This section explains the various advantages of using the CEC system. The importance o comments for any programming language is generally accepted. Program developers usually have to follow certain standards for commenting their program. A theoretica standard with respect to structural representation (grammatical) can be laid out to th developers, but the developers will find it difficult to remember these rules and use the efficiently in their code. This difficulty can be overcome by using the CEC system as th standard can be easily encoded as rules to be used by the CEC system in the generation o CE comments.

The GUI provided by the CEC system aids the user in entering CE comments. Some of the positive aspects of the GUI are, its presentation of the set of CE comments generated by the CEC system, querying the user for data to complete CE comments, and the help system aiding the user to construct CE comments

7.2 System limitations

One of the limitations of the system is that the CEC generated CE comments are not al semantically correct. For example, if the user misspells a verb, then the CEC system treat it as a noun because a word not found in the dictionary is treated as an identifier. This can result in the CE parser forming parse trees for meaningless sentences and the CEC syste generating meaningless CE comments.

The CEC system handles only the high-level non-terminals for the generation of CE comments. A better performance might be achieved by using non-terminals at one leve lower in the analysis of the ungrammatical comments and in the generation of CE comments, i.e., in terms of intermediate non-terminals. A wider choice of CE comments can be obtained by taking into consideration the intermediate non-terminals that may be used to form new high-level non-terminals. For example the active verb sequence (avs) ‘sends’ in the predicate chunk ‘sends the signal’ may be combined with some other nominal to form a different predicate chunk and thus forming a different CE comment.

7.3 Future extensions

The CEC system developed thus far does not include semantic analysis. Semantic analysis, if incorporated along with the structural analysis used currently in the CEC system can help in generation of CE comments that are semantically correct. Another improvement in the CEC system will be the addition of a spell checker to dynamically prompt the user when words are misspelled and prevent verbs from being used as nouns as explained in section 7.2.

Development of a knowledge base for the CEC system to aid users in entering CE comments will be very beneficial. The knowledge base has to have a measure of probability in terms of frequency of a particular sentence structure being used in the CEC system. This will help in providing the user with a limited number of generated CE comments and decreasing the time taken to arrive at a CE comment from an ungrammatical comment.

The CEC system can be used as an efficient tool for CE documentation. The CEC system can be modified to read every sentence in a document and with the help of interaction fro the user can generate equivalent CE sentences for every ungrammatical sentence.

The CEC system can be modified to allow the user to first enter all comments into the code and then analyze the comments in the code one by one to arrive at CE comments. This will be helpful for users who would like to have an undisturbed flow of thought while commenting their code. The CE comments that are then generated at the end wil have all the required information that may otherwise be lost due to the users thought stream fluctuating between writing a CE comment and giving all the required information in the comment. This modification may result in considerable delay in the time taken for the user to adapt to writing CE comments as the continuous learning advantage of the current system is lost.

References:

[PelD00] David Pellerin ,"An Introduction to VHDL for Synthesis and Simulation" Accolade Design Automation, Inc. http://www.acc-eda.com/h_intro.htm, 2000

[McCS93] Steve C McConnell, "Code Complete : A Practical Handbook of Software Construction" , Microsoft Press, Redmond1993.

[WojR96] Richard H. Wojick and Heather Holmback, "Getting a Controlled Language o the Ground at Boeing", ", First International Workshop on Controlled Languag Applications (CLAW '96), pp. 22 – 31, 26-27 March 1996.

[FarG96] Cordon Farrington, "AECMA Simplified English: an Overview of th International Aircraft Maintenance Language", ", First International Workshop on Controlled Language Applications (CLAW '96), pp.1 – 21, 26-27 March 1996.

[FucN96] Norbert E. Fuchs and Rol Schwitter, "Attempto Controlled English (ACE)", ", First International Workshop on Controlled Language Applications (CLAW '96), pp.124 – 136, 26-27 March 1996.

[WojR98] Richard H. Wojick, Heather Holmback & James Hoard, "Boeing Technica English", International Workshop on Controlled Language Applications (CLAW '98), pp. 114 - 122, 21-22 May1998.

[HayP96] Phil Hayes, Steve Maxwell and Linda Schmandt, "Controlled Language Support for Translated and Original English Documents", First International Workshop on Controlled Language Applications (CLAW '96), pp.84 – 92, 26-27 March 1996.

[CyrW95]Walling R. Cyre, "A Requirements Sublanguage for Automated Analysis", International Journal of Intelligent Systems, pp. 665 – 689, July 1995.

[WinT83] T. Winograd, "Language as a Cognitive Process", vol. 1 : Syntax, Addison – Wesley, 1983

[LinP90] Peter Linz, "An Introduction to Formal Languages and Automat " ,D.C Heath & CO, Boston,1990

[ManM] Menakshi Manek, "Natural Language Interface to a VHDL Modeling Tool", Ms. Thesis, Virginia Tech, 1993

[Rational96] Rational Software Corp. Rational rose 4.0.3, http://www.rational.com/uml, 1991-1996.

Appendix A

User’s guide for Controlled English Commenting System (CEC)

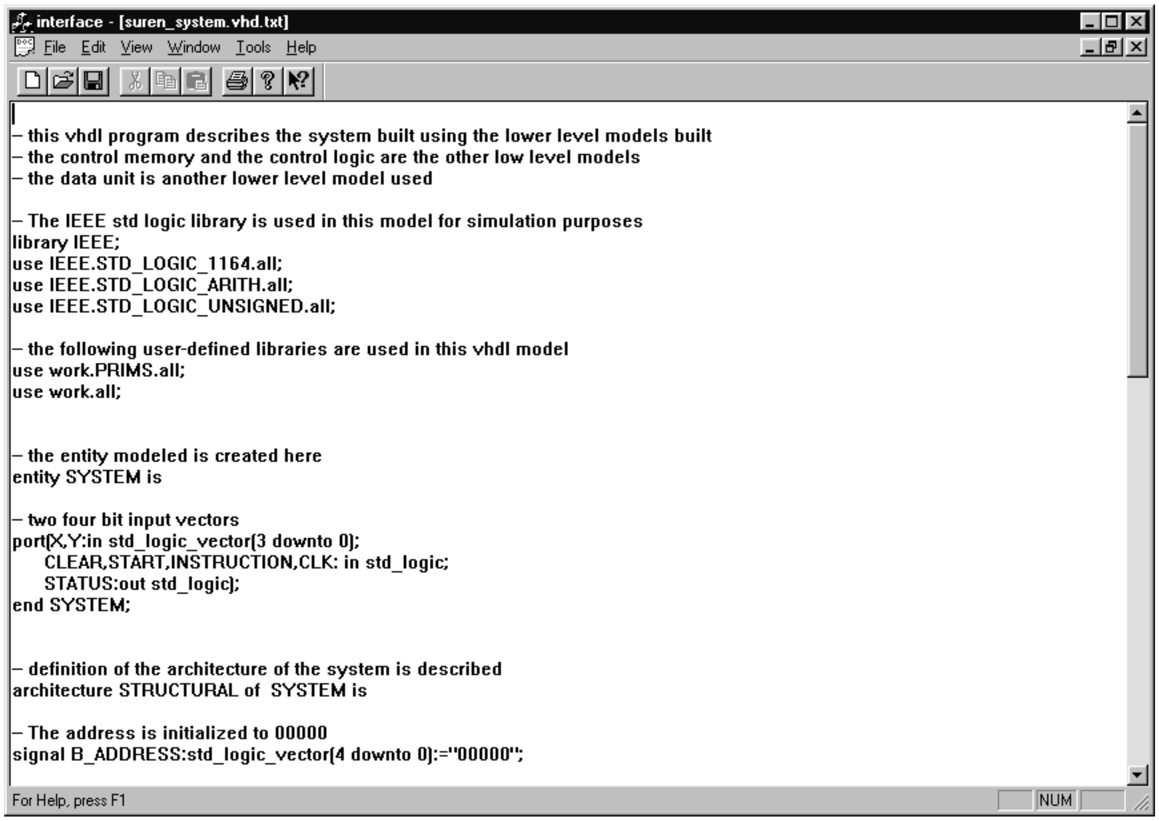
This appendix explains in detail, how the user should use the CEC system as a tool for commenting VHDL programs. A demonstration of how to use the CEC system is provided, step by step. The demonstration starts with the explanation of how to use the “code window” (window showing the VHDL code to be commented), followed by the “interface dialog” which interacts with the user to enter CE comments and then the “help dialog” which helps the user to learn how a CE comment should be constructed.

Demonstration

Double clicking on the CEC.exe icon starts the CEC. The CE parser is incorporated into the CEC and hence the commenter and the parser make one executable file. The files used by the parser are stored in the same directory as the CEC executable file.
The different files used by the parser are:
rules.txt : This contains all the grammar rules used by the CE parser
rdict.txt : This is a single word dictionary used by the CE parser
mdict.txt : This is a multiword dictionary used by the CE parser
The formats of the above mentioned files are shown in Appendix C.

Code Window

An editable document window is presented to the user initially when the CEC is started. The user can open the VHDL document that he/she wishes to comment using the ‘File – Open’ option in the menu. The code to be commented in the opened VHDL document is highlighted by using the mouse. The ‘Select’ option available in the ‘Tools’ menu is then chosen.


interface - [suren_system.vhd.txt]
File Edit View Window Tools Help
For Help, press F1
NUM

```
-- this vhdl program describes the system built using the lower level models built
-- the control memory and the control logic are the other low level models
-- the data unit is another lower level model used

-- The IEEE std logic library is used in this model for simulation purposes
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

-- the following user-defined libraries are used in this vhdl model
use work.PRIMS.all;
use work.all;


-- the entity modeled is created here
entity SYSTEM is

-- two four bit input vectors
port(X,Y:in std_logic_vector(3 downto 0);
     CLEAR,START,INSTRUCTION,CLK: in std_logic;
     STATUS:out std_logic);
end SYSTEM;


-- definition of the architecture of the system is described
architecture STRUCTURAL of  SYSTEM is

-- The address is initialized to 00000
signal B_ADDRESS:std_logic_vector(4 downto 0):="00000";
```

Figure A.1 Code Window

Interface Dialog

Selecting the ‘Select’ tab in the ‘Tools’ menu opens an interface dialog that interacts with the user in arriving at CE comments from the ungrammatical comments entered by the user. The user enters the comment in the text field labeled “Enter Comment” and indicates the end of a comment with a period ‘.’ (The CEC appends the period if the user fails to include it. This feature is made default because the CE parser recognizes the end of a sentence by searching for the period). The interface dialog is shown in Figure A.2.

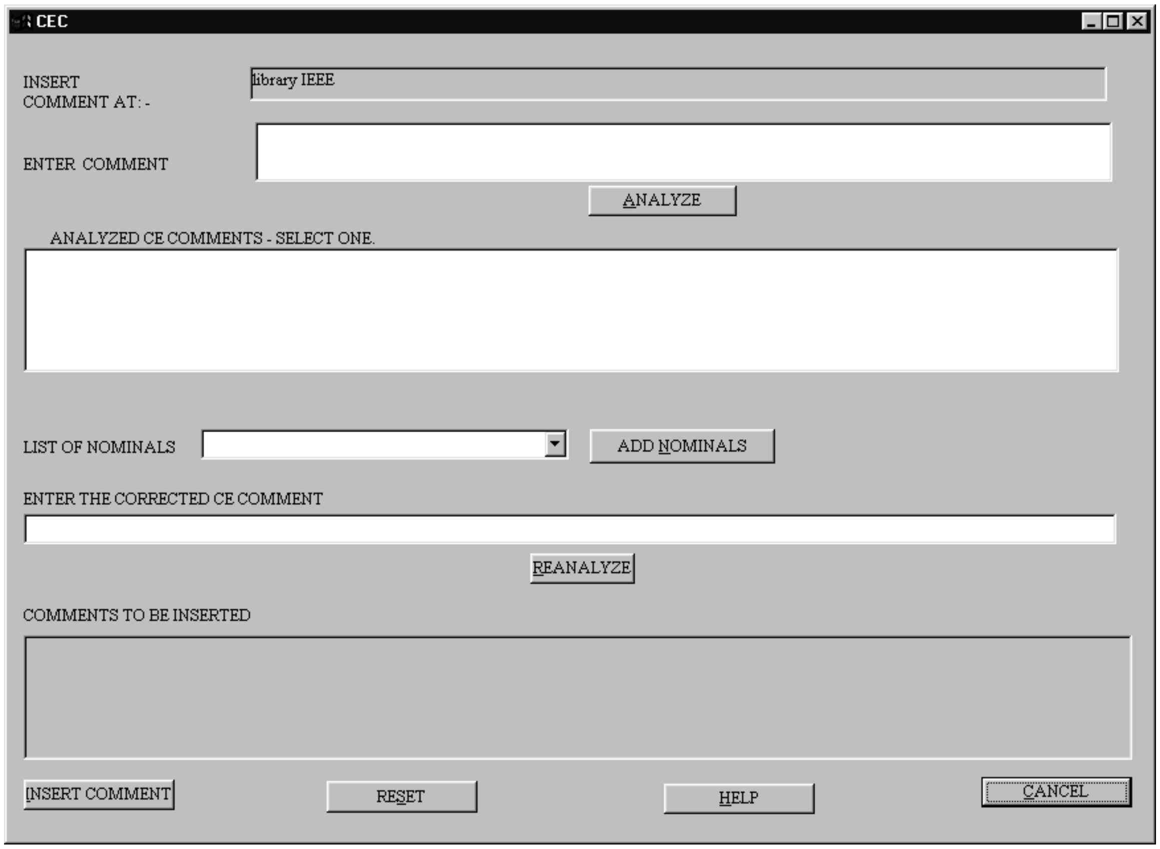
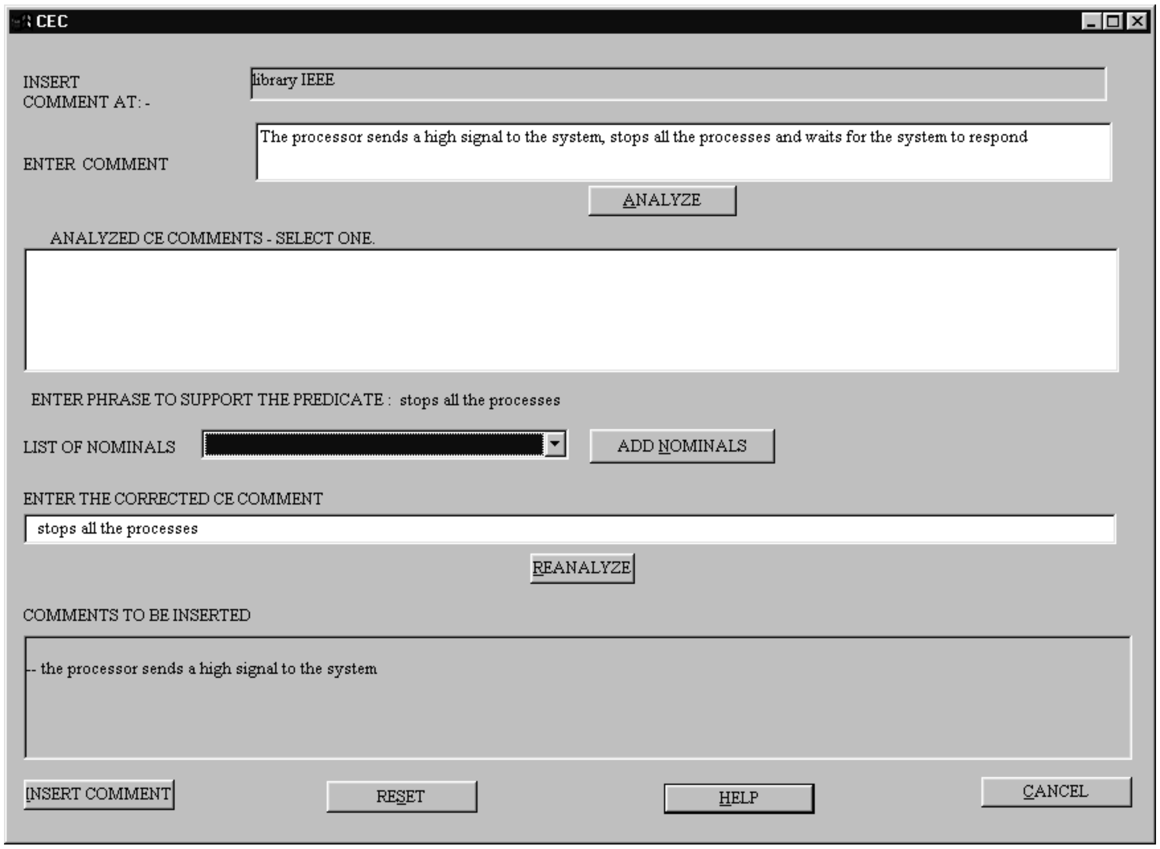


Figure A.2 Interface dialog box

When the user clicks the ‘Analyze’ button below the comment entry field, the CE analyzer starts to analyze the comment the user has entered. The CEC forms all possible CE comments and displays them in the list box below the heading - “Analyzed CE Comments – Select one”. The user can go through the CE comments produced by the CE and select one of them by double clicking on it. Once the user selects a CE comment, the selected comment is added to the field below the heading - “Comments to be inserted”. The CEC system then forms other CE comments and the user is prompted again to select one of them. If the user is not satisfied with the CE comments generated by the CEC, he/she can rephrase the comment in the field below the heading -“Enter the corrected CE comment” and then click on the ‘Reanalyze’ button to have the comment analyzed again.

In Figure A.3, the CEC queries the user for subjects to support residue predicates, and predicates to support residue nominals. This is possible if the CEC is not able to form any more CE comments but still has orphan predicates or nominals that have not been used in the CE comments previously selected.



library IEEE

The processor sends a high signal to the system, stops all the processes and waits for the system to respond

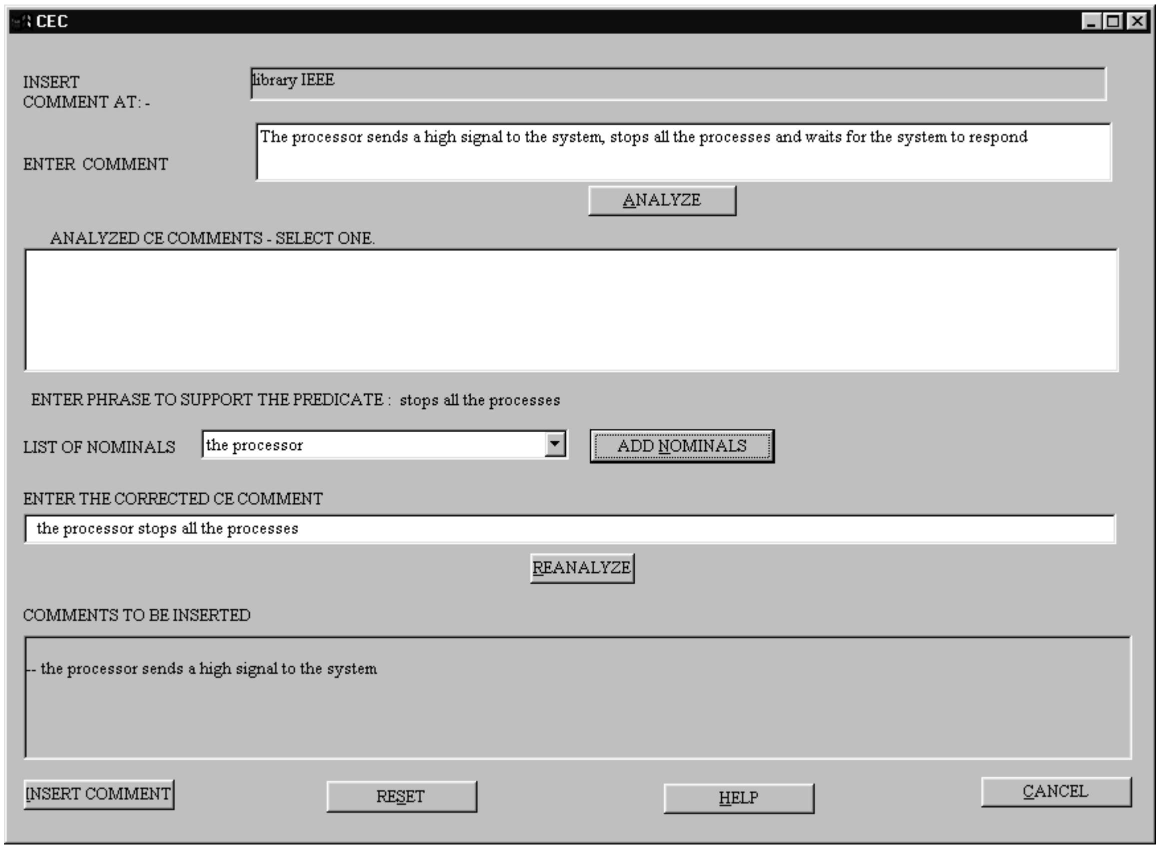
ENTER PHRASE TO SUPPORT THE PREDICATE : stops all the processes

stops all the processes

-- the processor sends a high signal to the system

Figure A.3 CEC querying the user for a nominal to support a orphan predicate

The user can select a nominal from the drop box titled "List of Nominals" that has a lis of all the nominals used in the original ungrammatical comment. Clicking on the "Add Nominal" button appends the selected nominal to the residue predicate in the field below the heading - "Enter the corrected CE comment". The 'Reanalyze' button is then clicked to re-analyze the comment. The re-analyzed comment can now be selected. The interface dialog after a nominal has been selected is shown in Figure A.4.



INSERT COMMENT AT: - library IEEE

ENTER COMMENT: The processor sends a high signal to the system, stops all the processes and waits for the system to respond

ENTER PHRASE TO SUPPORT THE PREDICATE : stops all the processes

LIST OF NOMINALS: the processor

ENTER THE CORRECTED CE COMMENT: the processor stops all the processes

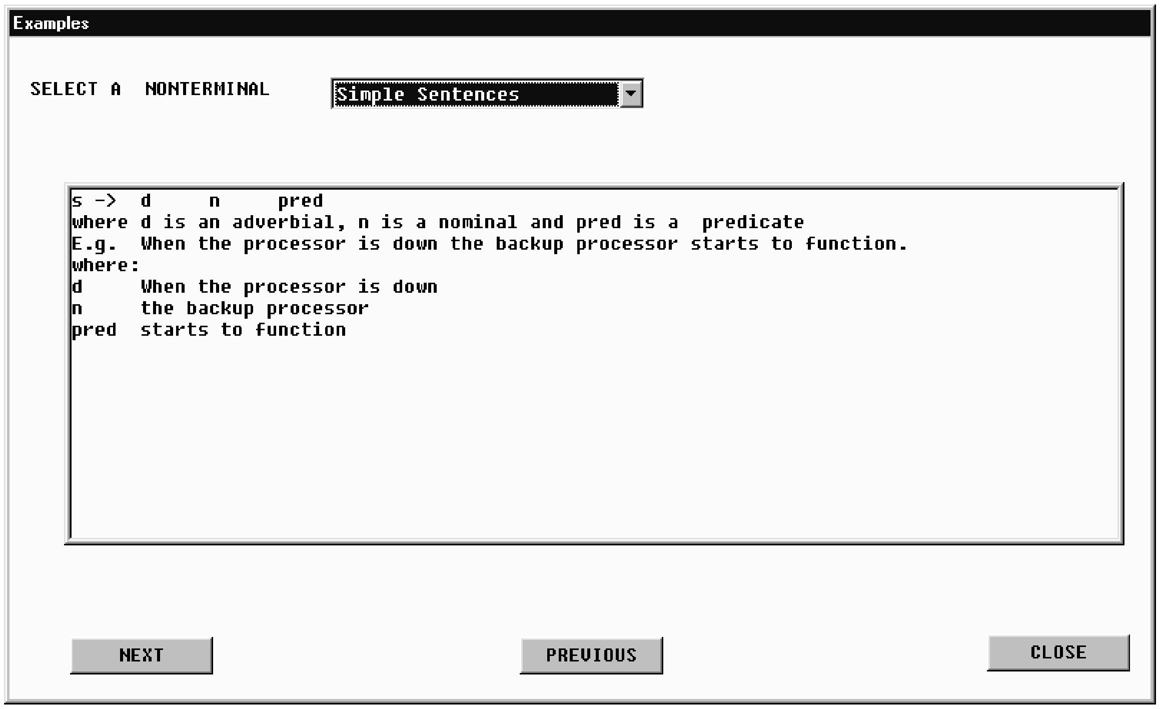
COMMENTS TO BE INSERTED:
-- the processor sends a high signal to the system

Figure A.4 Interface dialog with a nominal being added to a orphan predicate

The user then selects the CE comments and inserts the CE comments into the code b clicking on the 'Insert' button. The "Reset" button clears all the entries in the interface dialog and allows the user to start afresh. The "Help" button opens up the help dialog that helps the user to enter a controlled English comment.

Help Dialog Box

The user can get help in writing a controlled English comment by clicking on the "Help" button on the interface dialog. This dialog helps the user in understanding how a CE comment should be constructed. The help dialog box is show in Figure A.5.



```
s ->  d     n     pred
where d is an adverbial, n is a nominal and pred is a  predicate
E.g.  When the processor is down the backup processor starts to function.
where:
d     When the processor is down
n     the backup processor
pred  starts to function
```

Figure A.5 Help dialog box

The help dialog provides the user with details on how nominals, predicates, and adverbials are constructed with various other lower level non-terminals. It also provides the user with details on constructing the CE comments that make use of nominals, predicates, and adverbials. The help dialog has a “Next” and “Previous” button that help the user to traverse through various examples. The user can select appropriate help details for nominals, predicates, adverbials, or for CE comments by selecting the appropriate entry from the drop box provided in the help dialog and clicking on the “Show Example” button.

Appendix B

Experiment Results

This appendix shows the experiment results for the four subjects not discussed in detail in chapter 6. Both normal performance plots as well as the online performance plots ar shown here. Table B1 shown below indexes the various normal and online performance plots for the respective subjects.

Table B1 Index of plots

Subject	**Normal performance plots**				**Online performance plots**			
	Time	**Unknow words**	**Total no. o words**	**No. o passes**	**Time**	**Unknow words**	**Total no. o words**	**No. o passes**
1	B1	B2	B3	B4	B5	B6	B7	B8
2	B9	B10	B11	B12	B13	B14	B15	B16
3	B17	B18	B19	B20	B21	B22	B23	B24
4	B25	B26	B27	B28	B29	B30	B31	B32

Subject 1.

Normal performance plot for Johnson counter and traffic light controller.

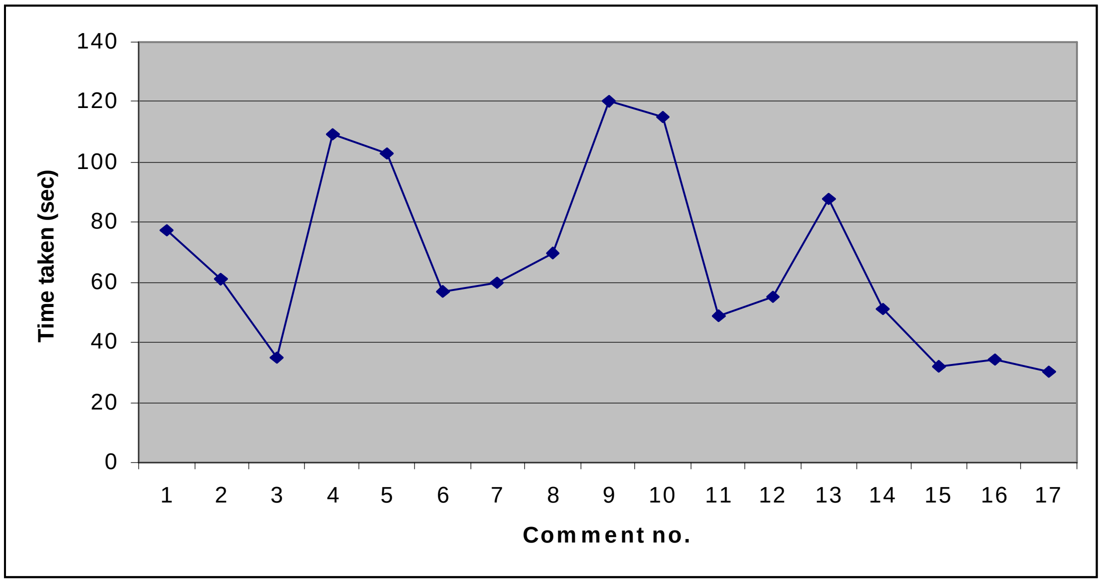


Figure B1 Plot showing time taken in seconds versus the comment numbe

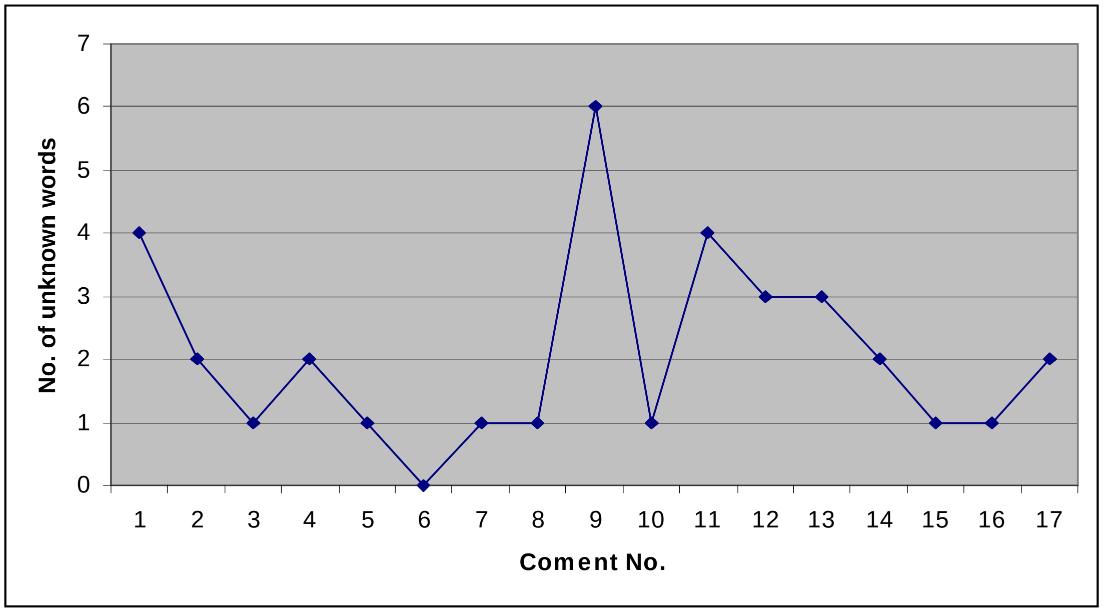


Figure B2 Plot showing number of unknown words versus the comment number

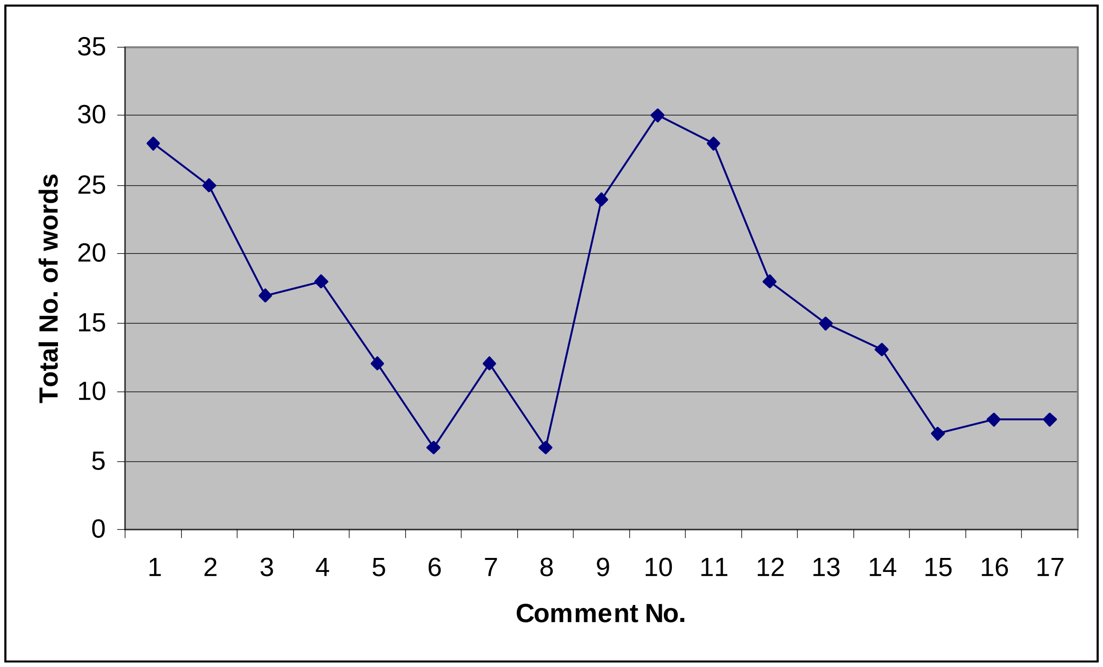


Figure B3 Plot showing total number of words versus the comment number

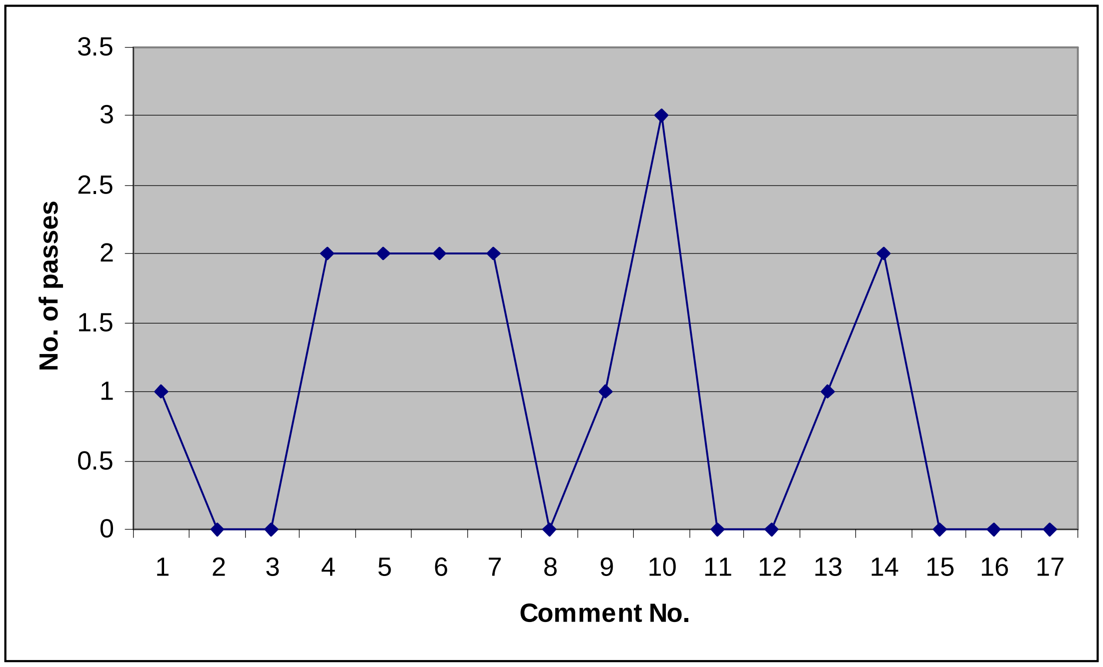


Figure B4 Plot showing number of passes versus the comment number.

Online Performance plots for Johnson counter and traffic light controlle

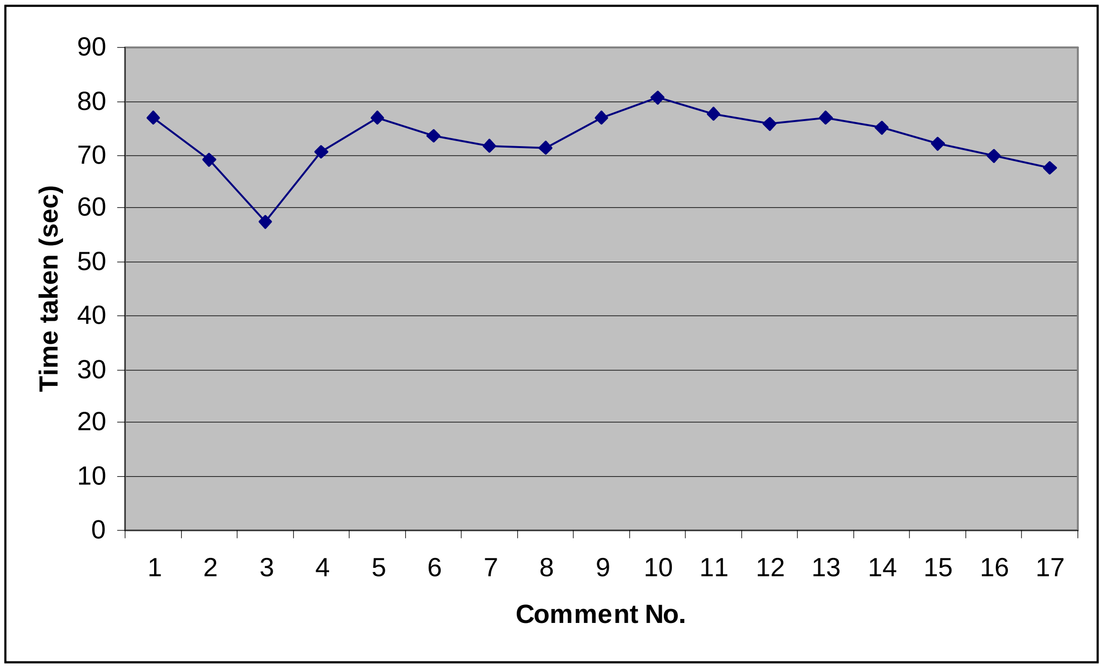


Figure B5 Plot showing time taken in seconds versus the comment numbe

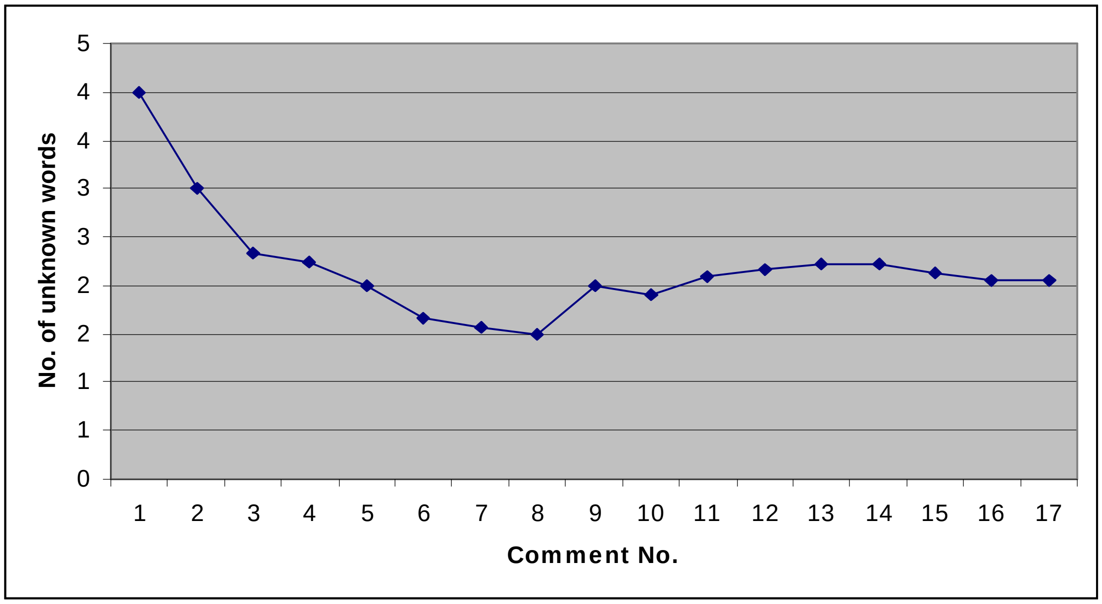


Figure B6 Plot showing number of unknown words versus the comment numbe

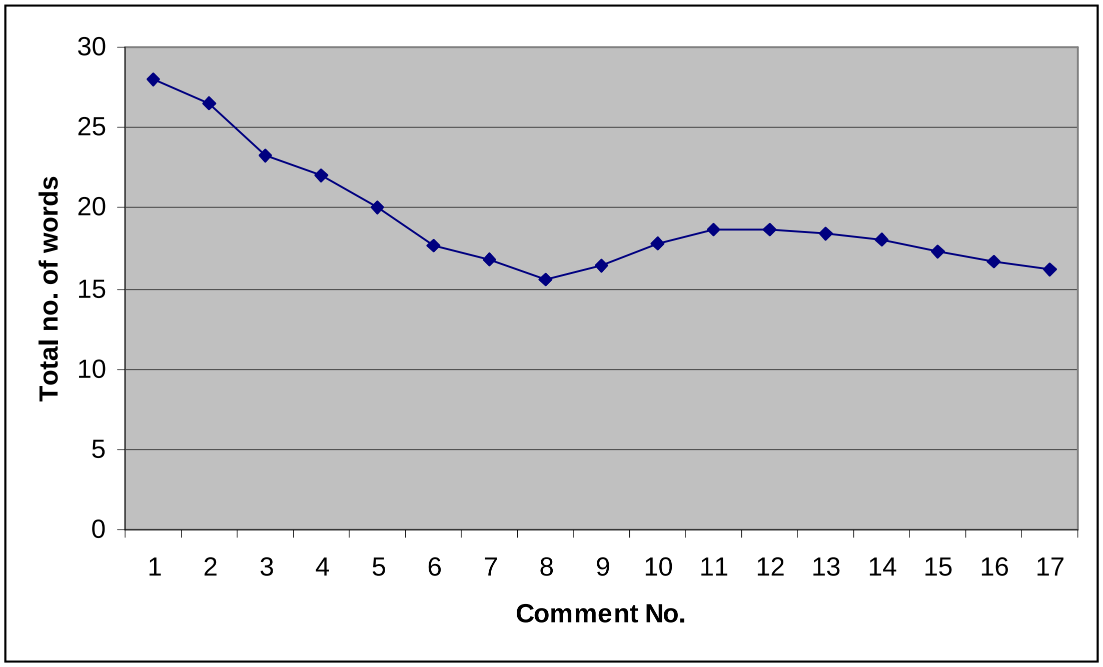


Figure B7 Plot showing total number of words versus the comment number

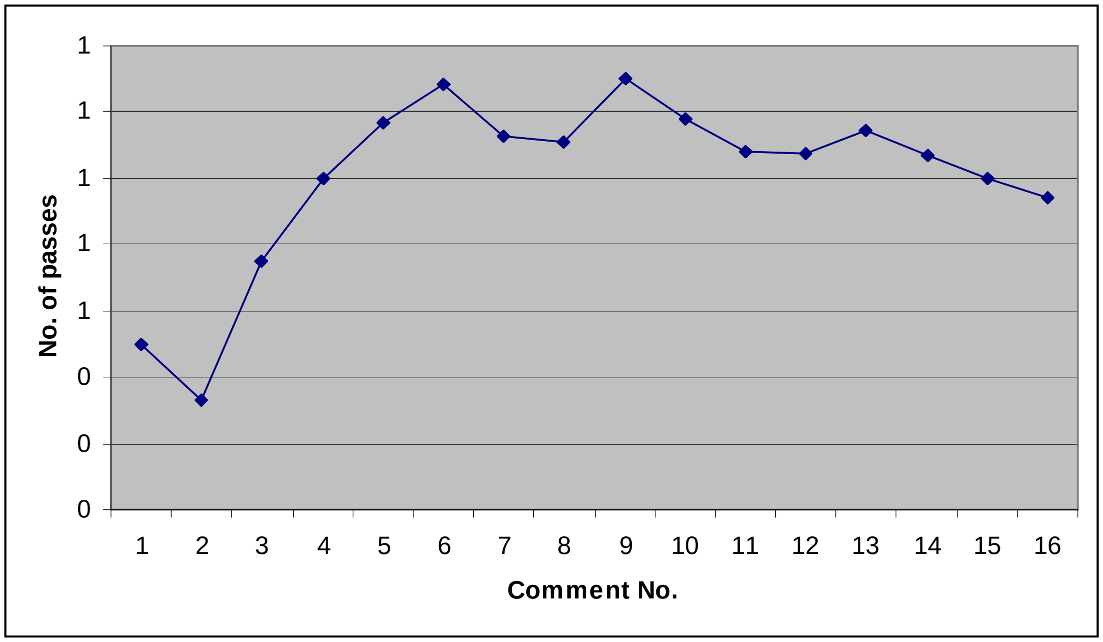


Figure B8 Plot showing number of passes versus the comment number.

Subject 2.

Normal performance plots for Histogram generator

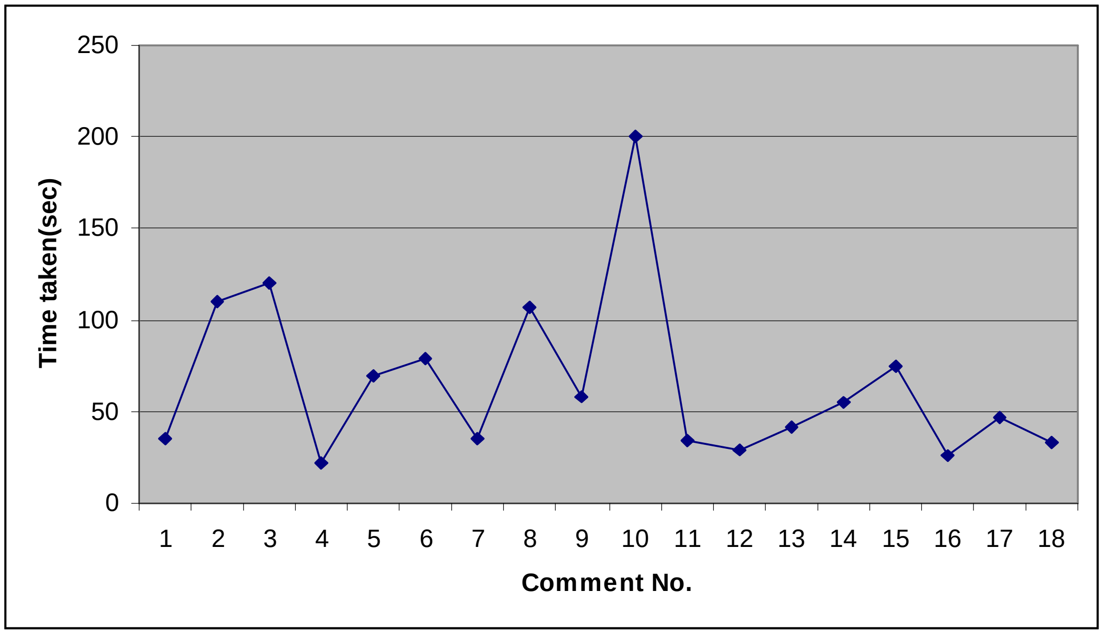


Figure B9 Plot showing time taken in seconds versus the comment numbe

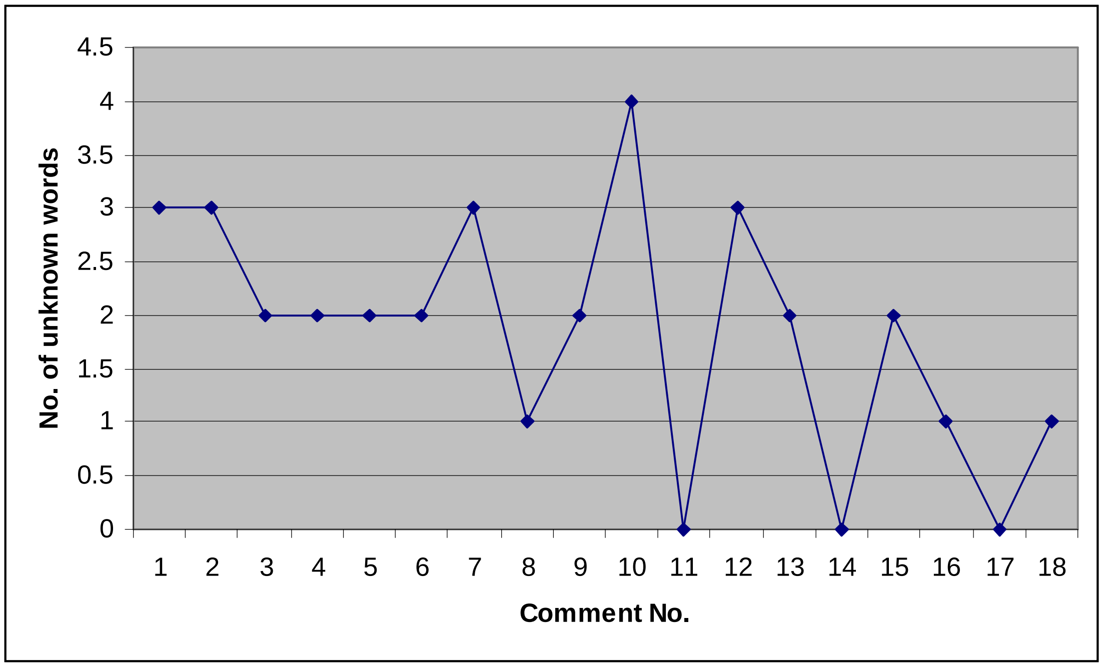


Figure B10 Plot showing number of unknown words versus the comment numbe

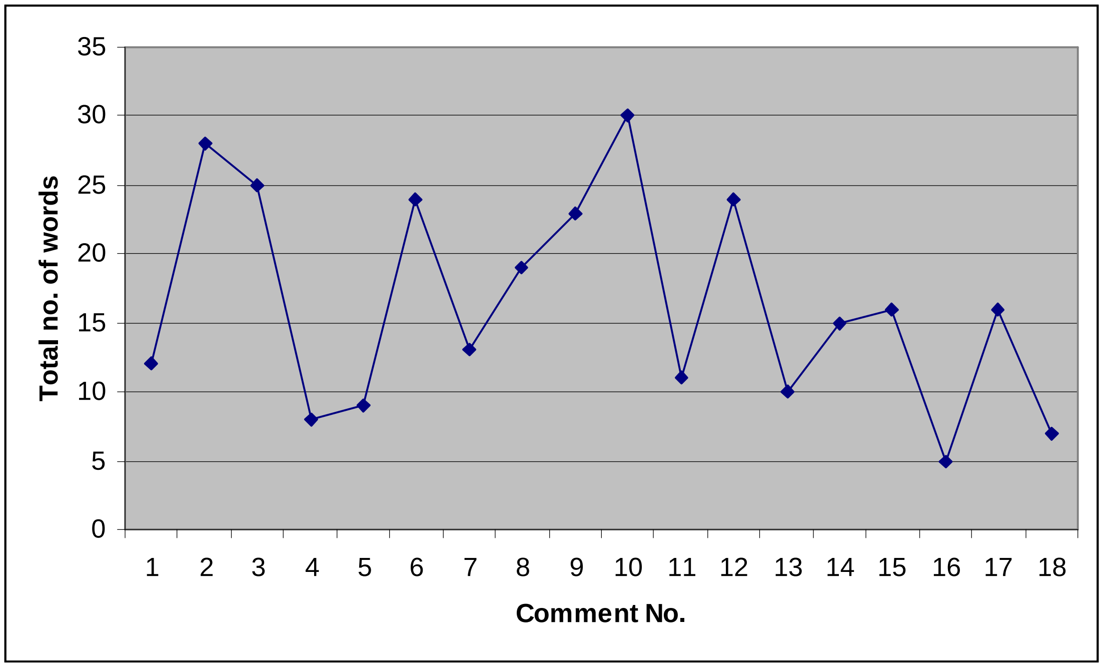


Figure B11 Plot showing total number of words versus the comment number.

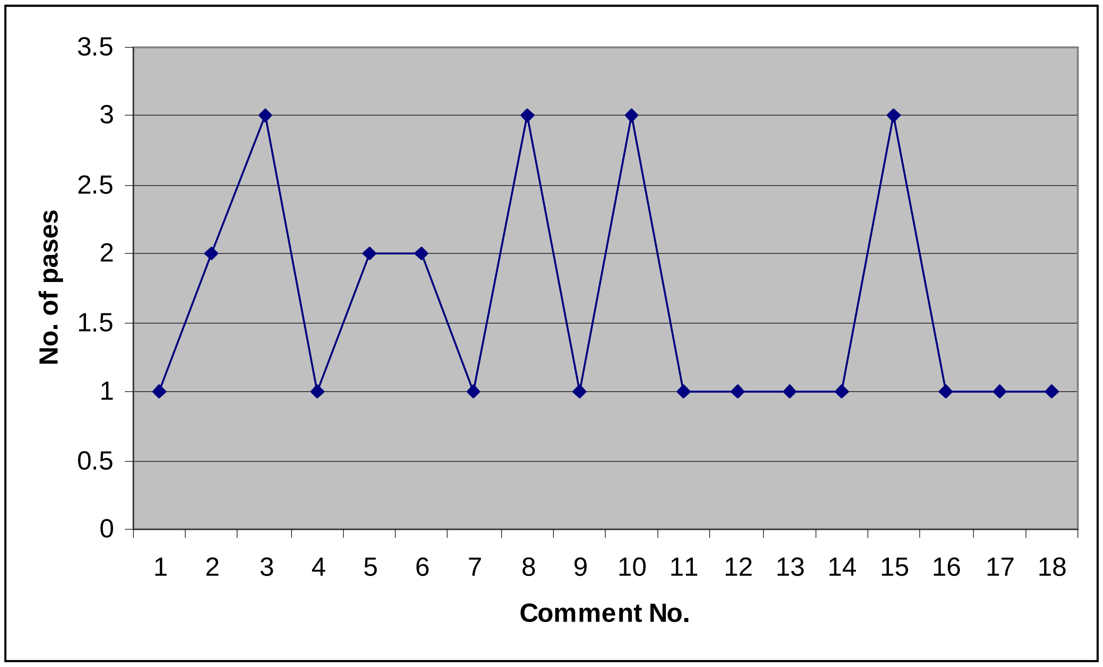


Figure B12 Plot showing number of passes versus the comment number.

Online Performance plots for Histogram generator

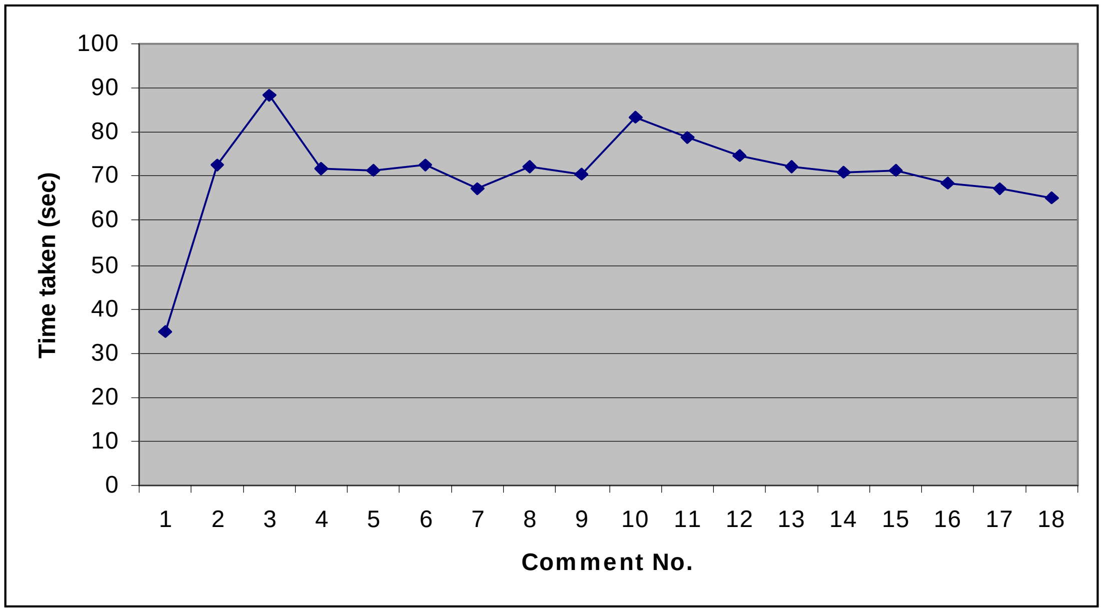


Figure B13 Plot showing time taken in seconds versus the comment numbe

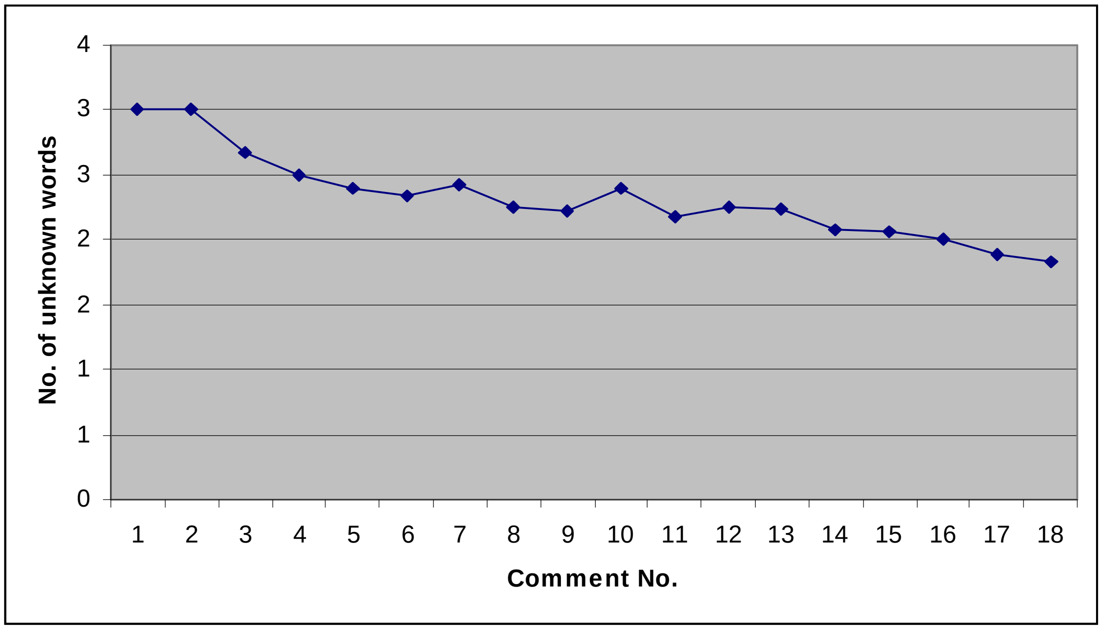


Figure B14 Plot showing number of unknown words versus the comment numbe

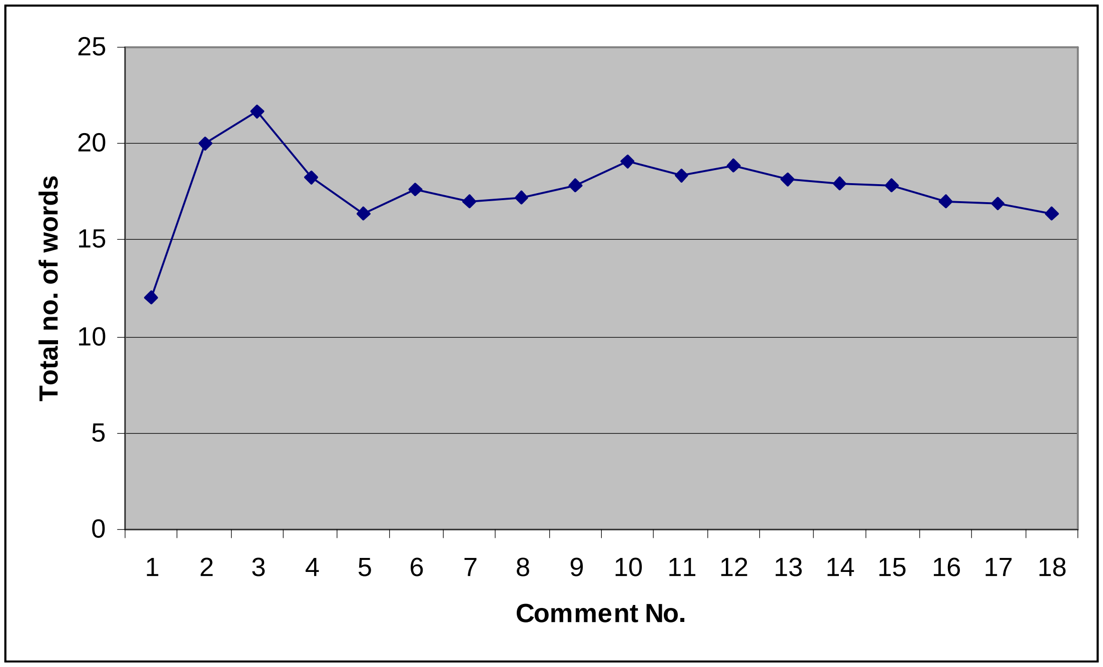


Figure B15 Plot showing total number of words versus the comment number.

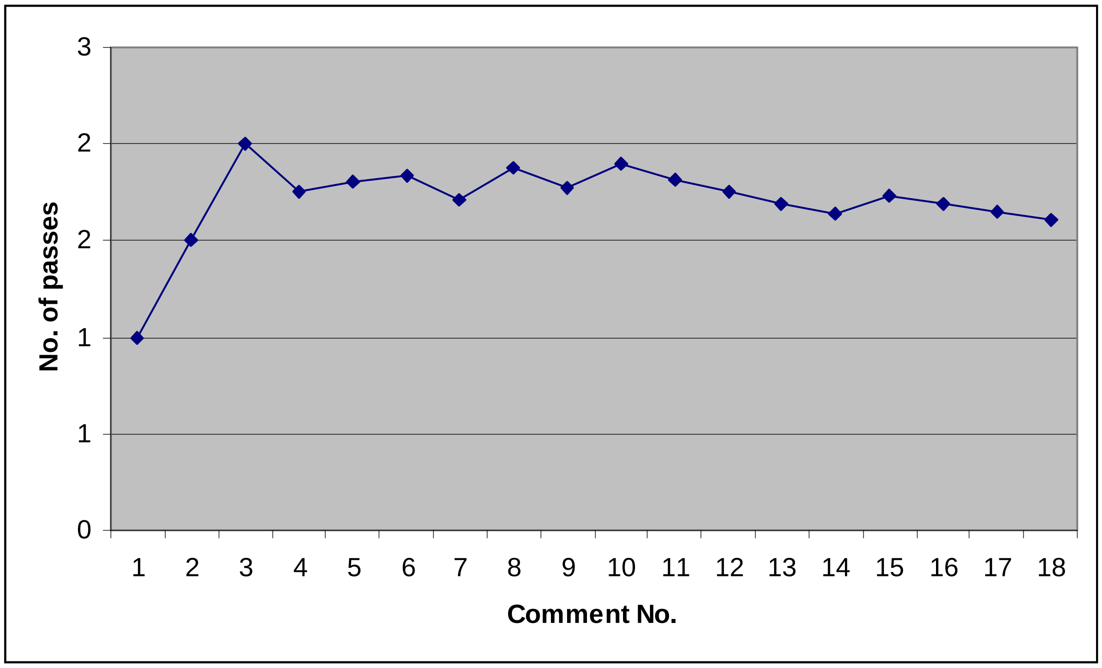


Figure B16 Plot showing number of passes versus the comment numbe

Subject 3

Normal performance plots for ALU model

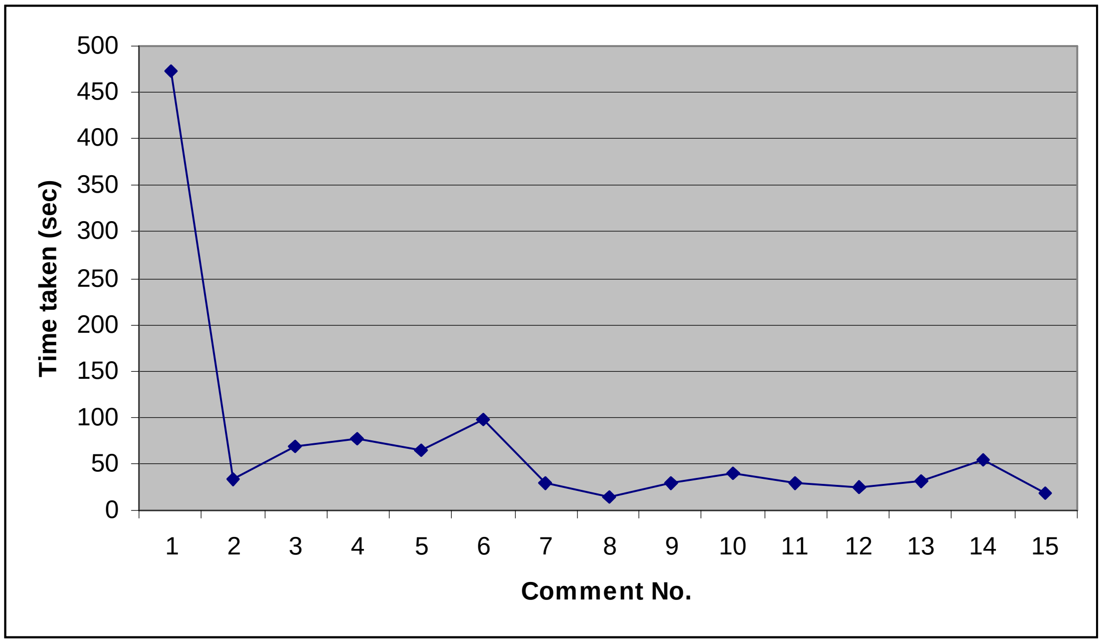


Figure B17 Plot showing time taken in seconds versus the comment numbe

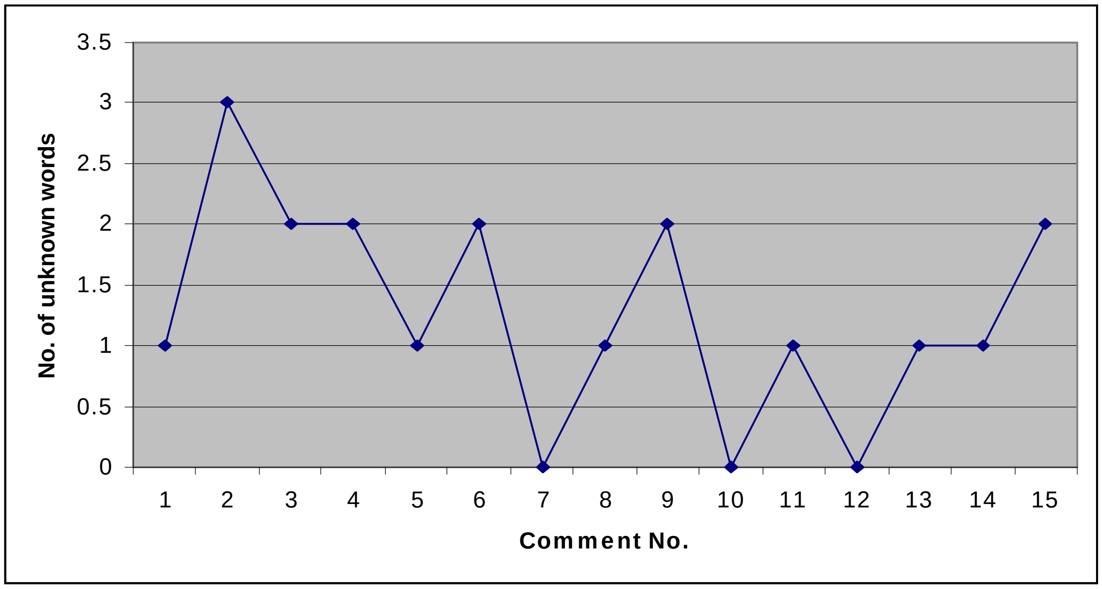


Figure B18 Plot showing number of unknown words versus the comment numbe

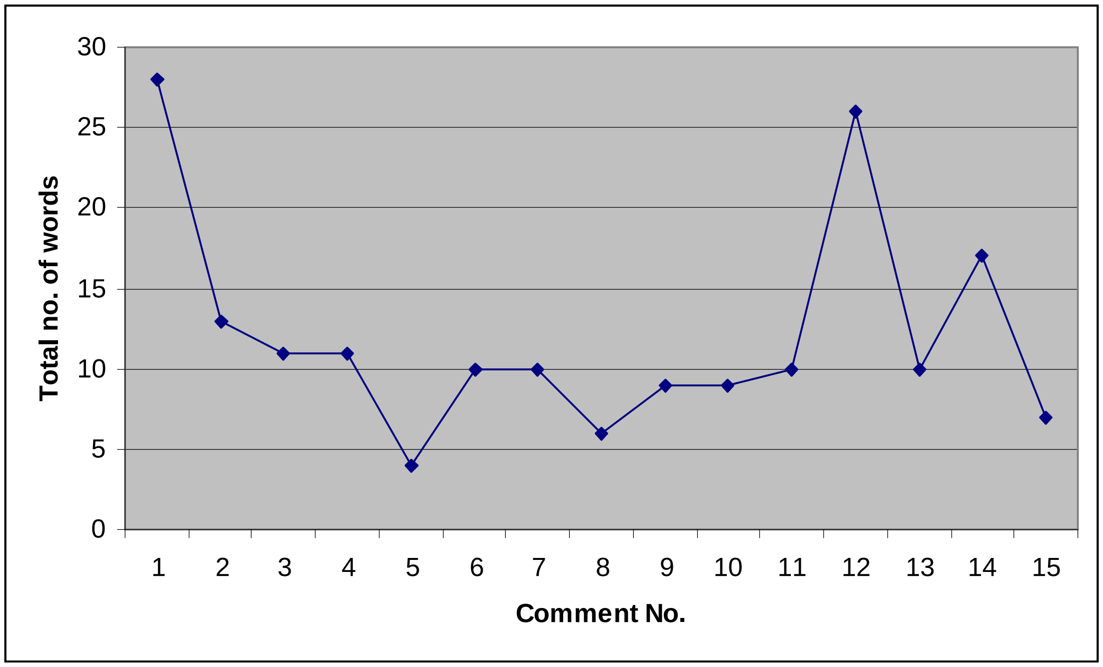


Figure B19 Plot showing total number of words versus the comment number.

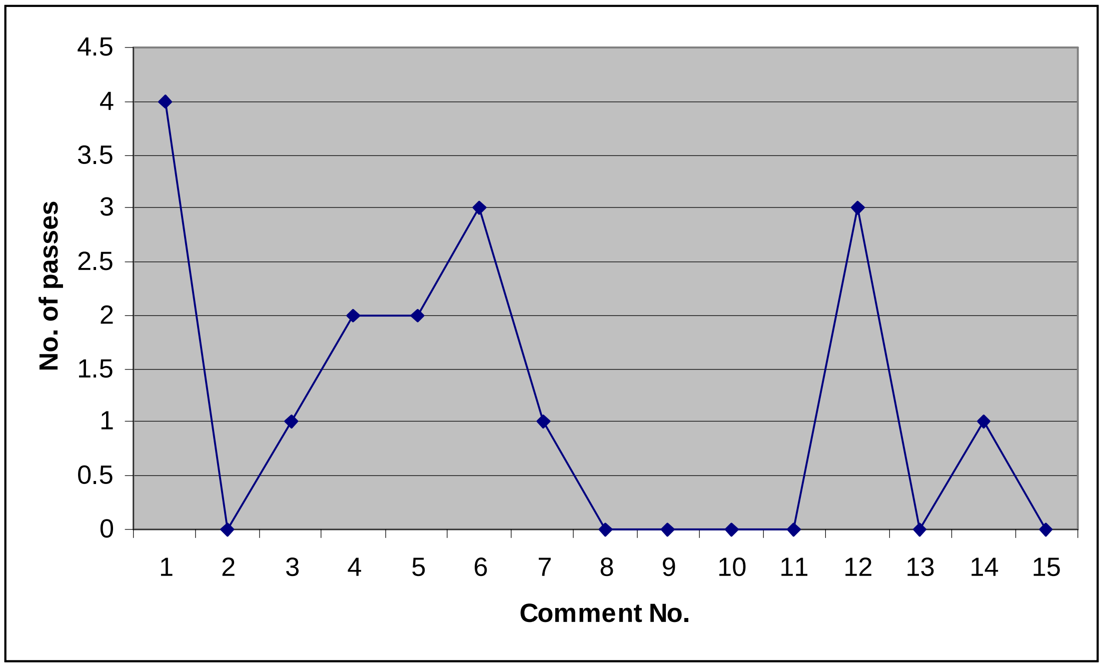


Figure B20 Plot showing number of passes versus the comment numbe

Online performance plots for ALU model

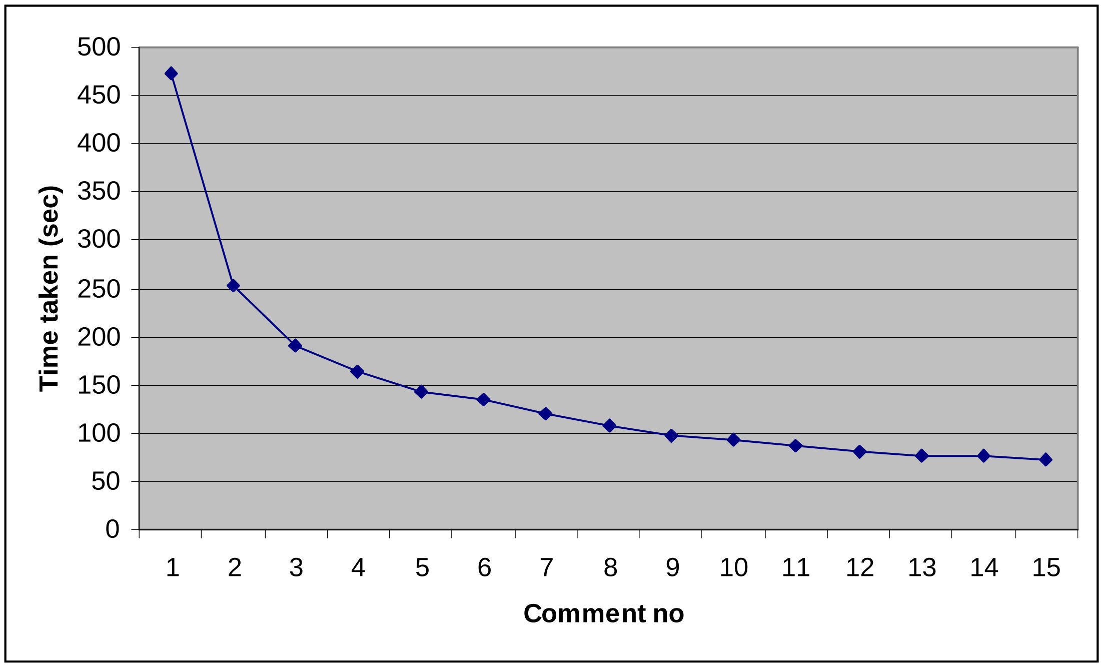


Figure B21 Plot showing time taken in seconds versus the comment numbe

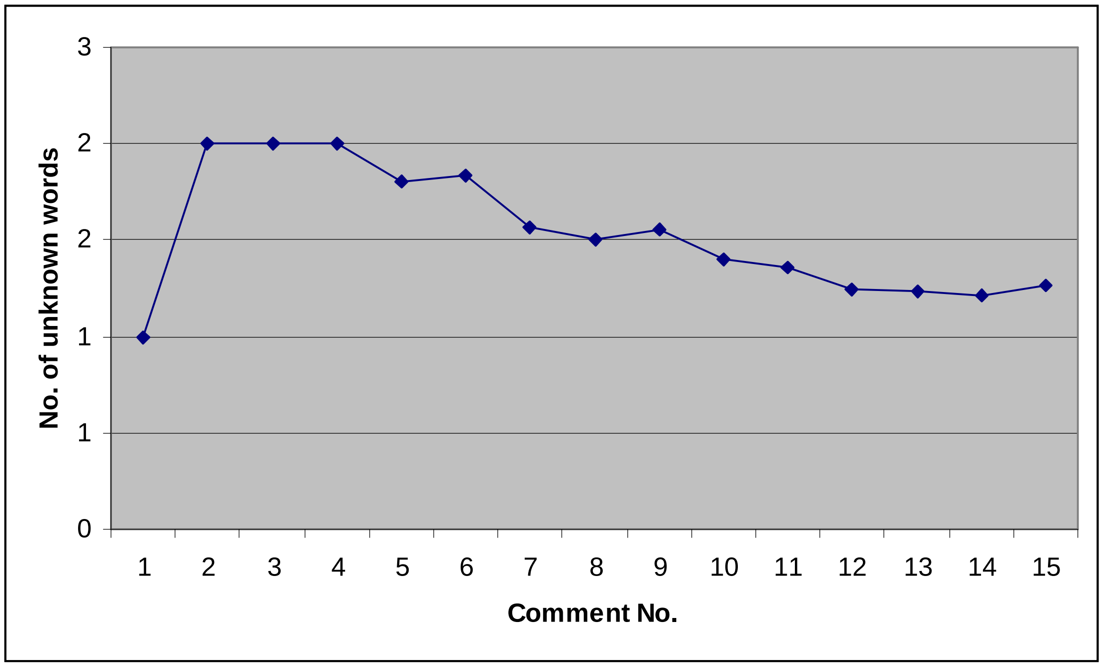


Figure B22 Plot showing number of unknown words versus the comment numbe

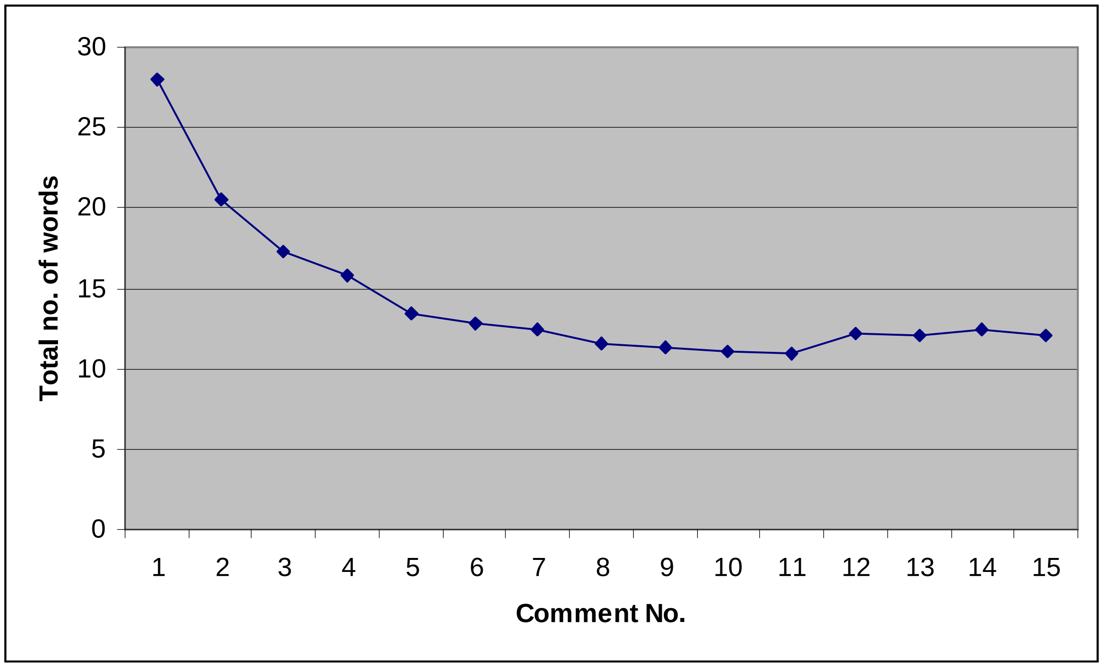


Figure B23 Plot showing total number of words versus the comment number.

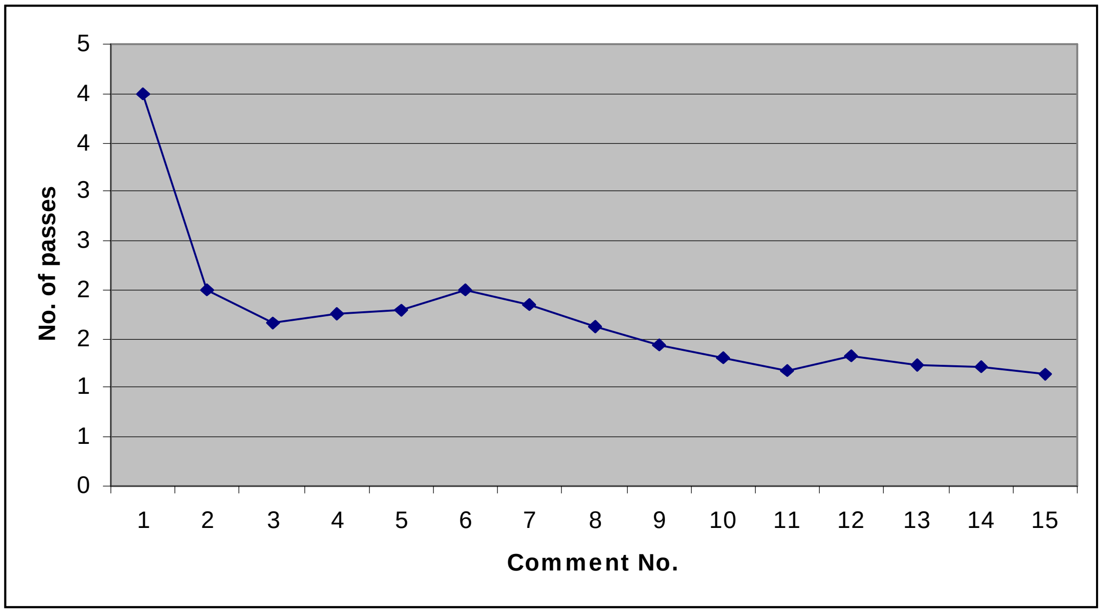


Figure B24 Plot showing number of passes versus the comment numbe

Subject 4

Normal performance plots for Johnson counte

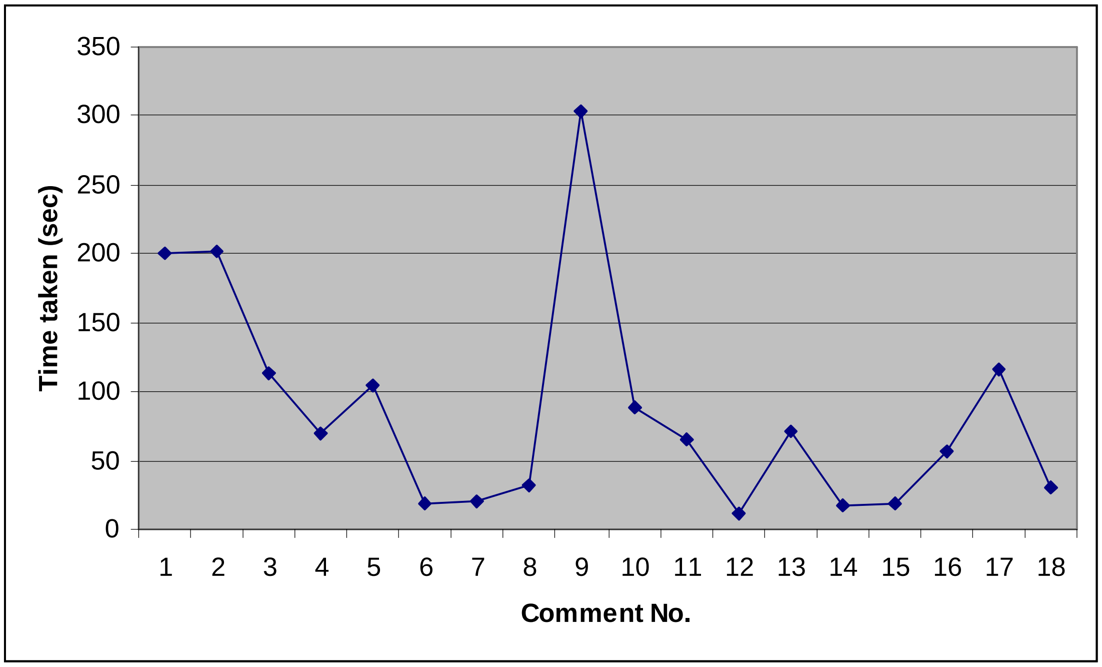


Figure B25 Plot showing time taken in seconds versus the comment numbe

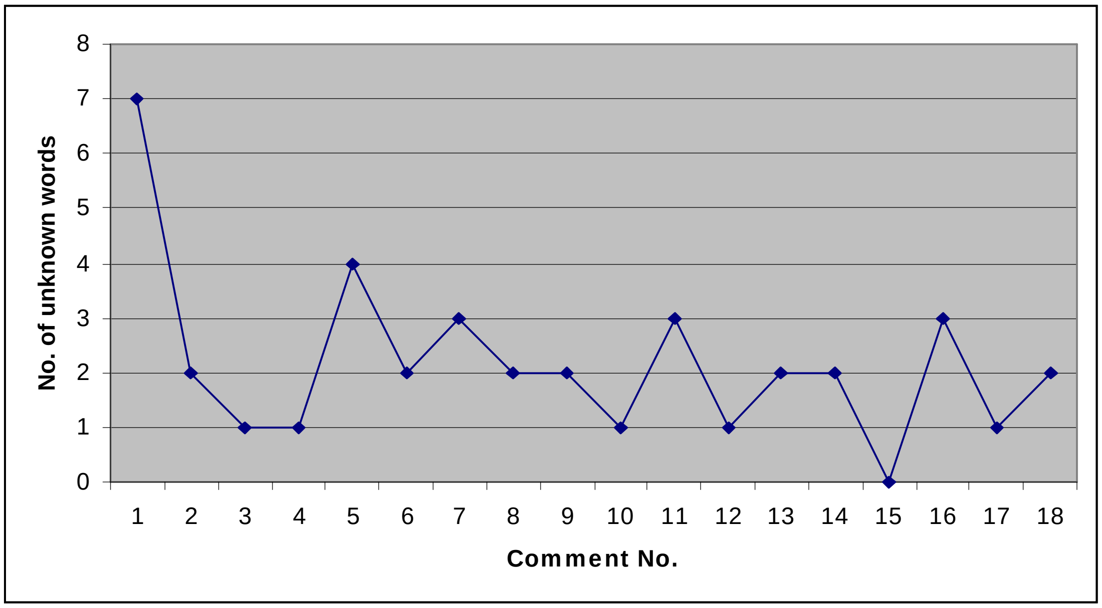


Figure B26 Plot showing number of unknown words versus the comment numbe

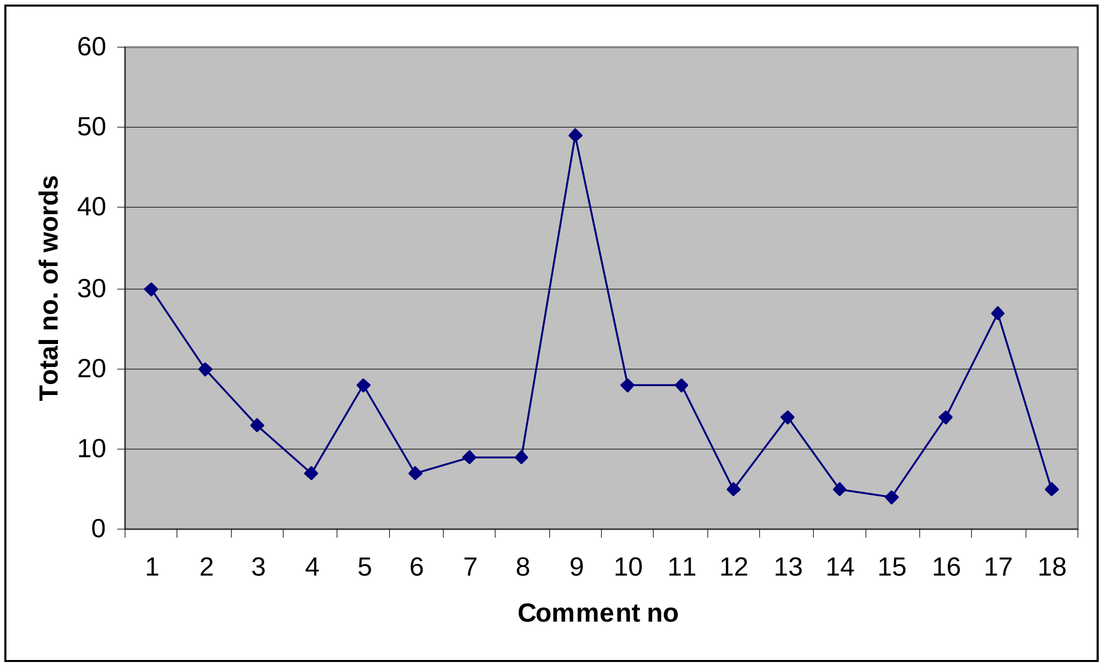


Figure B27 Plot showing total number of words versus the comment number.

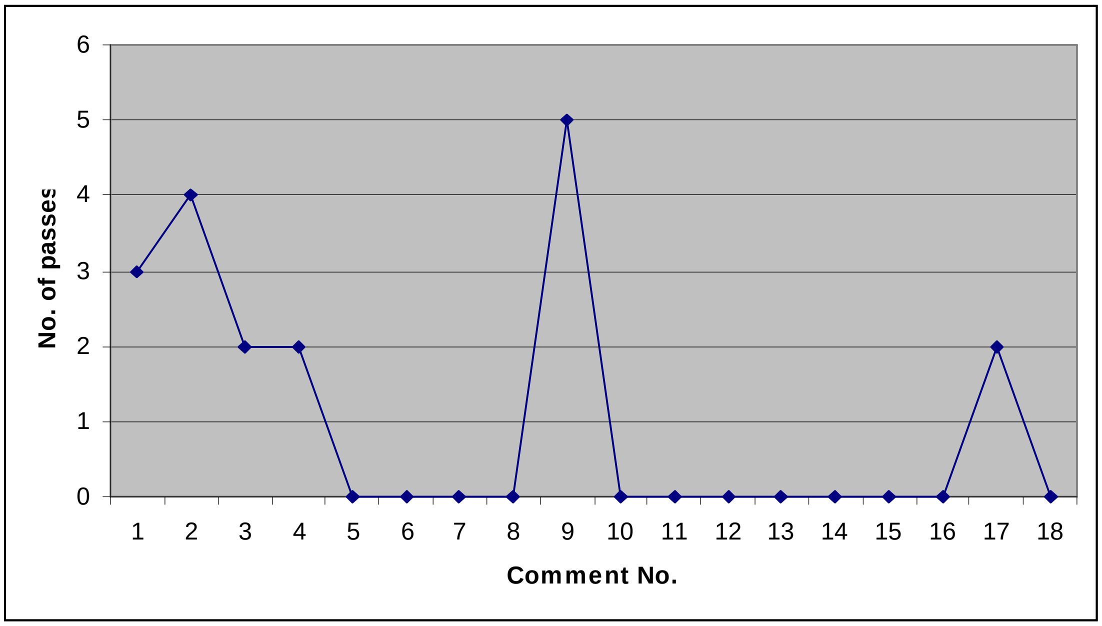


Figure B28 Plot showing number of passes versus the comment numbe

Online Performance plots for Johnson counte

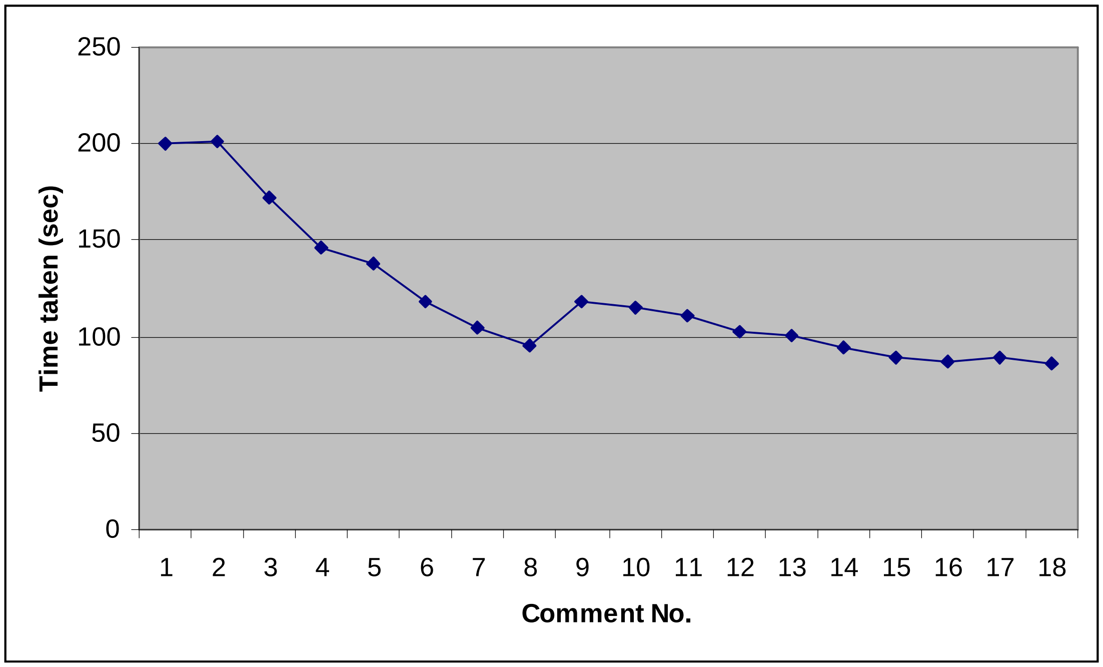


Figure B29 Plot showing time taken in seconds versus the comment numbe

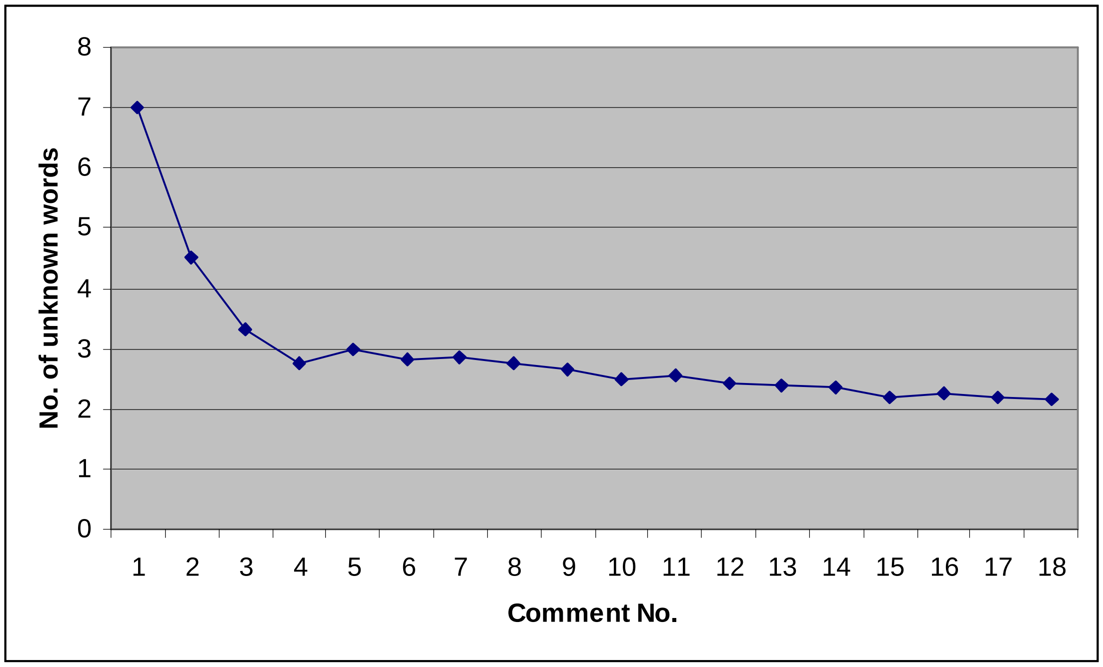


Figure B30 Plot showing number of unknown words versus the comment numbe

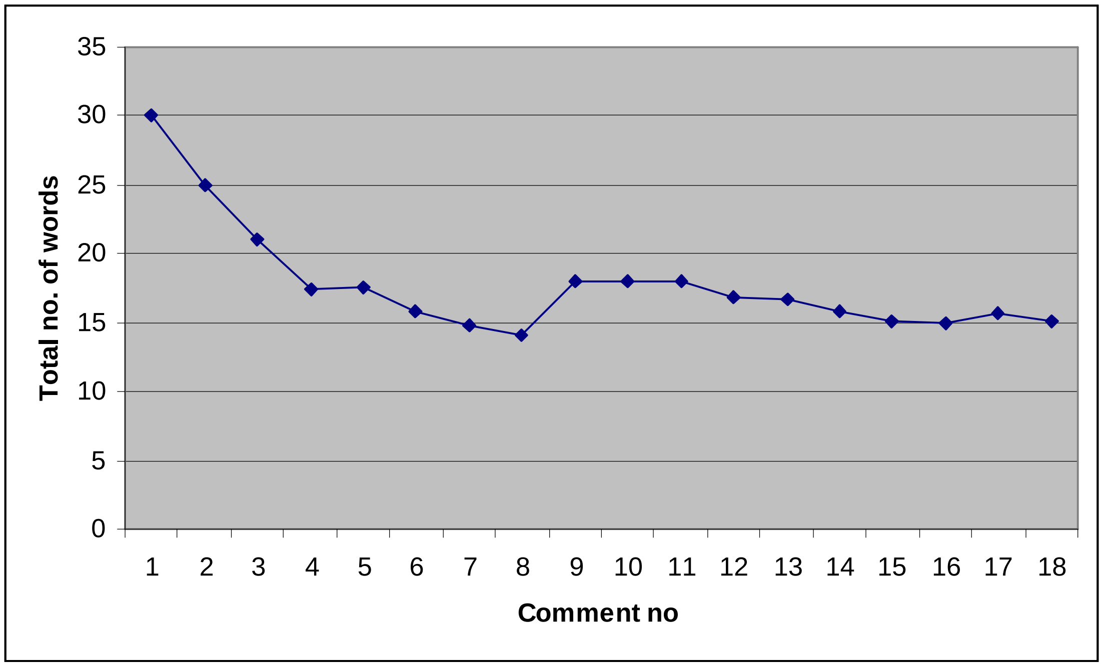


Figure B31 Plot showing total number of words versus the comment number.

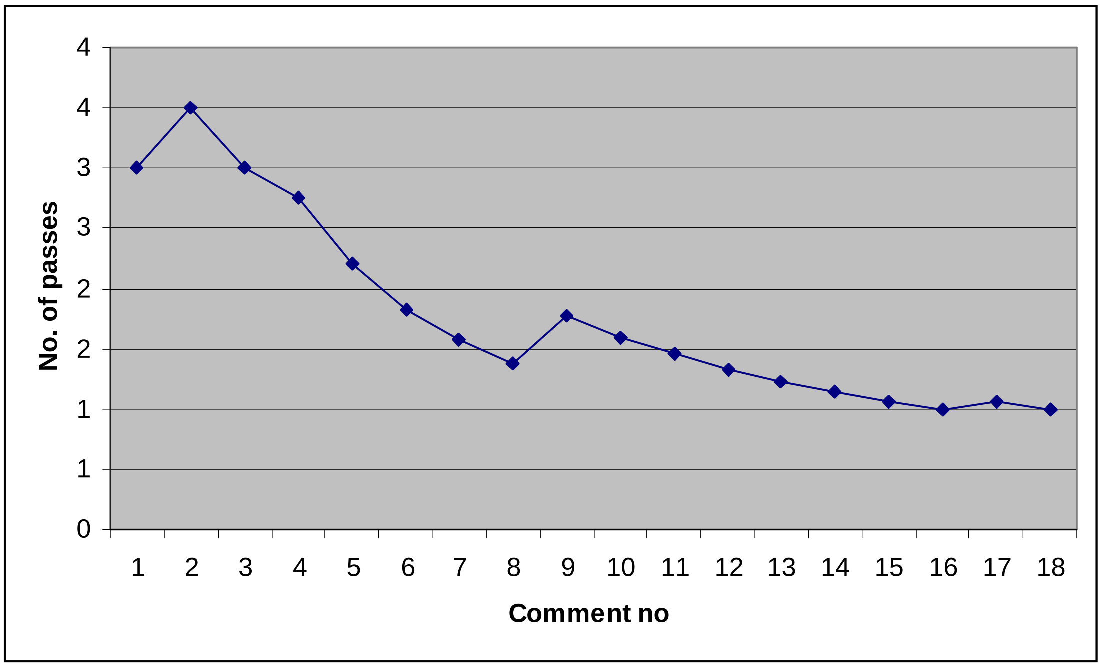


Figure B32 Plot showing number of passes versus the comment numbe

Appendix C

Files used by the CEC system

This appendix describes the various files used by the CEC system in the generation of CE comments for VHDL models. Three input files are used by the CEC system. They are th "rule.txt" which contains the list of all CE rules used by the CE parser, the "rdict.txt" which is a dictionary of all words allowed by the controlled English and the "mdict.txt" which is a multiword dictionary, containing the list of multiwords allowed by the controlled English.

C.1 Rule.txt

This file contains the list of grammar rules used by the CE parser. Each rule consists of a non-terminal symbol on the left-hand side, and a sequence of constituents (terminals and non-terminals) on the right-hand side. The contents of the 'rule.txt' is listed below followed by Table C1 listing the abbreviations used in the rules.

```
s        →    ss   .
s    → sp   .
s    → sc   .
ss   → d    n    pred
ss   → d    ,    n    pred
ss   → n    pred
sp   → d    n    pred  conj  pred
sp   → n    pred  conj  pred
sp   → d    n    pred ,    conj  pred
sp   → d    ,    n    pred  conj  pred
sc   → ss   conj  ss
pred →      avs
pred →      avs
pred →      avs   n    d
pred →      avs   n    d    d
pred →      avs   d
pred →      avs   d    d
pred →      pvs
pred →      pvs
pred →      pvs   d
pred →      pvs   d    d
pred →      pvs   d    d    d
pred    →     evs
pred  →       evs   d
pred  →       evs   n    d
pred  →       evs   adjs
pred  →       evs   adjs  d
pred  →       evs   det   d
avs   →       adv   verb
avs   →       verb
avs   →       mod   vinf
avs   →       mod   adv   vinf
avs   →       mod   not   vinf
avs   →       mod   not   adv   vinf
avs   →       have
pvs   →       be    ven
pvs   →       be    not   ven
pvs   →       be    adv   ven
pvs   →       be    not   adv   ven
pvs   →       mod   bei   ven
pvs   →       mod   adv   bei   ven
pvs   →       mod   not   bei   ven
pvs   →       mod   bei   adv   ven
pvs   →       mod   not   bei   adv   ven
evs   →       be
evs   →       be    no
evs   →       mod   not   bei
evs   →       mod   adv   bei
d     → adv
d     → prep
d     → scon  n
d     → scon  ss
d     → scon  ss    conj  ss
n     → np
n     → np    pp
n     → np    rest
n     → np    of    n
n     → np    o     np    pp
n     → nc
n     → cl
n     → cl    conj  cl
n     → cl    ,     cl    conj  cl
nc    →np    conj  np
np    →pdet  det   adjs  head
np    →pdet  det   head
np    →pdet  adjs  head
np    →pdet  head
np    →det   ord   #    adjs  head
np    →det   ord   #    head
np    →det   ord   adjs  head
np    →det   ord   head
np    →det   #    adjs  head
np    →det   #    head
np    →det   adjs  head
np    →det   head
np    →ord   #    adjs  head
np    →ord   #    head
np    →ord   adjs  head
np    →ord   head
np    →#    adjs  head
np    →#    head
np    →adjs  head
np    →head
np    →head  range
np    →pron
np    →np   #
np    →(    np   )
np    →np   (    np   )
np    →         [    np   ]
np    →det   ving  head
np    →np   ving  head
np    →det   ven   head
np    →det   adv   ven   head
np    →pdet  head  adjs  head
head  →         noun
head  →         id
head  →         noun  head
head  →         id   head
head  →         #   head
adjs  →         adj
adjs  →         adj   adjs
adjs  →         adj   conj  adjs
adjs  →         adj   ,    adjs
adjs  →         (    adj   )
pp    →prep
pp    →prep  conj  prep  n
rest  →rcon  pred
rest  →cl
cl    → nvs
cl    → nvs   d
cl    → n    nvs
cl    → n    nvs   d
cl    → n    xvs
cl    → xvs
cl    → n    xvs   d
cl    → xvs   d
cl    → n    ivs
cl    → n    ivs
cl    → ivs
cl    → ivs
cl    → n     ivs   n     d
cl    → n     ivs   d
cl    → ivs   n     d
cl    → ivs   d
cl    → n     gvs
cl    → gvs
cl    → gvs
cl    → gvs   n     d
cl    → gvs   d
nvs   →             ven
xvs   →             to    bei   ven
ivs   → to    vinf
gvs   →             ving
noun  →             nounp
verb  →             verbp
```

Table C1 List of abbreviations used in “rules.txt”

Abbreviation	**Non-terminal**
adj	adjective
adjs	adjective sequence
adv	adverb
avs	active verb sequence
bei	‘be’ (infinite ‘to be’)
cl	clause
conj	conjuncti
d	adverbial
evs	equative verb sequence
gvs	ving verb sequence (gerund)
head	head
ivs	infinitive verb sequence
mod	modifier
n	nomina
np	noun phrase
nvs	past participle verb sequence
ord	ordinal
pdet	pre-determiner
pp	prepositional phrase
prep	preposition
pred	predicate
pvs	passive verb sequence
rcon	restricted clause
rest	relative conjunction
s	sentence
scon	sub-ordinate conjunction
ss	simple sentence
verbp	verb phrase
ven	past participle
vinf	verb infinitive
ving	present participle
xvs	passive verb sequence

C.2 Rdict.txt.

This file contains the dictionary used by the CE parser. The entire “rdict.txt” file is very large and is not listed here. Table C2 shows sample entries in the “rdict.txt” file. Each word in the vocabulary is followed by the non-terminal it can be represented with, such as noun and verb, followed by other information not used by the CEC.

Table C2 Sample entries in the “rdict.txt” file

```
respond | verb _    vinf action `
responding | ving action `
responds | verb _ `
response | noun behavior `
responses | noun _ `
responsibility | noun attribute `
responsive | adj _ `
rest | verb _    vinf _ | noun _ `
restart | verb _    vinf _ `
restarted | ven action `
restorage | noun _ `
restore | verb _    vinf action `
restored | ven action `
restricted | ven _ `
restriction | noun _ `
rests | verb _ `
result | noun value | verb _    vinf state `
resultant | noun _ `
resulting | ving _ `
results | noun _ `
resume | verb _    vinf action `
resumed | ven _ `
resumes | verb _ `
resuming | ving _ `
resynchronize | verb _    vinf _ `
resynchronized | ven _ `
resynchronizes | verb _ `
retain | verb _    vinf action `
```

C.3 Mdict.txt

This file contains the list o multiwords allowed by the controlled English used in the CE system. The “mdict.txt” file is listed below. . Each multiword in this file is followed by the non-terminal it can be represented with.

```
result in | verb behavior | vinf behavior `
results in | verb _ `
consist of | verb _ |vinf _ `
consists of | verb _ `
each of | pdet _ `
some of | pdet _ `
all of | pdet _ `
many of | pdet _ `
a few of | pdet _ `
a number of   pdet _ `
in spite of | prep _ `
instead of | scon _ `
pair of | # _ `
regardless of | prep _ `
along with | prep _ `
depend on | verb event  vinf event `
depending on | prep _ | ving _ `
depends on | verb behavior `
synthetic aperture radar | noun _ `
control logic | adj _ `
farther than | adj _ `
faster than | adj _ `
fewer than | adj _ `
greater than | pdet _ `
less than | pdet _ `
according to | prep _ `
greater than or equal to | pdet _ `
less than or equal to | pdet _ `
by means of | prep _ `
and then | conj _ `
such as | prep _ `
according to | prep _ `
in accordance with | prep _ `
logic one | noun value `
logic zero | noun value _ `
multiples of | adj _ `
with respect to | prep _ `
in response to | prep _ `
because of | prep _ `
central processing unit | noun _ `
arithmetic logic unit | noun _ `
```

for the purpose of | prep _ `
half of | adj _ `
in order to | prep _ `
in the same way as | prep _ `
in accordance with | prep _ `
as long as | prep _ `
and so | conj _ `
and then | conj _ `
only when | scon _ `
shift register | noun _ `
program counter | noun _ `
index register | noun _ `
shift registers | nouns _ `
program counters | nouns _ `
index registers | nouns _ `
in lieu of | prep _ `
in spite of | prep _ `

Appendix D

VHDL models

This appendix shows the VHDL models commented by the various subjects using the CEC system. Four different models are shown here: a Histogram generator, a Johnson counter, a Traffic light controller and an ALU model. The comment numbers are shown in parentheses. Comments having the same number in a model represent an ungrammatical comment (UC) broken down into more than one controlled English comment.

D.1 Johnson counter and Traffic light controller models commented by subject 1

D.1.1 Johnson counte

```
-- this is a structural implementation of a  johnsons counter where behavioral components are defined as components in the work library and been structurally instantiated from the library (1)
use work.all;
architecture STRUCTURAL of JOHNSONS_COUNTER is

--  these are internal wires that connect to the flip-flop outputs  and are further used to analyze th internal waveforms in the simulator (2)
signal INT1, INT2, INT3, INT0: BIT ;

-- These wires connect to the complemented outputs of the flip-flop and are used for simulation (3)
signal INT5, INT6, INT7, INT8: BIT ;

-- the signals int9 to int14 are used for parallel load implementation and self correcting logic (4)
signal INT9, INT10, INT11, INT12: BIT ;


--INT13 comes from self correcting logic (5)
signal INT13, INT14: BIT;

-- the various components required for the behavioral components are explained here (6)
component MY_AND2 is
```

```
port(I1, I2:in BIT; O:out BIT);
end component;

component MY_OR2 is
port(I1, I2:in BIT; O:out BIT);
end component;

component FF is
port(D, CLK, RESET:in BIT; Q, QC:out BIT);
end component;

component MY_MUX2 is
port(A, B, SEL:in BIT; O:out BIT);
end component;

-- the component associations are done in the following lines (7)
for all: FF use entity D_FLIP_FLOP(BEHAVIORAL);
for all: MY_AND2 use entity AND2(BEHAVIORAL);
for all: MY_OR2 use entity OR2(BEHAVIORAL);
for all: MY_MUX2 use entity MUX2(STRUCTURAL);
begin
C1: FF
        port map(INT9, CLK, RESET, INT0, INT8);
C2: FF
        port map(INT10, CLK, RESET, INT1, INT7);
C3: FF
        port map(INT11, CLK, RESET, INT2, INT6);
C4: FF
        port map(INT12, CLK, RESET, INT3, INT5);

-- the following code implements th   combinational logic required for self correction (8)
C5: MY_OR2
        port map(INT3, INT1, INT14);
C6: MY_AND2
        port map(INT14, INT2, INT13);
C7: MY_MUX2
```

```
        port map( DATA(0), INT1, LOAD, INT9);
C8: MY_MUX2
        port map( DATA(1), INT13, LOAD, INT10);
C9: MY_MUX2
        port map( DATA(2), INT3, LOAD, INT11);
C10: MY_MUX2
        port map( DATA(3), INT8, LOAD, INT12);
OUTPUT(0) <= INT0;
OUTPUT(1) <= INT1;
OUTPUT(2) <= INT2;
OUTPUT(3) <= INT3;
end STRUCTURAL ;
```

D.1.2 Traffic Light Controller

```
-- this is an algorithmic implementation of the classic highway intersection traffic controller  which
has a user interaction facility (9)
use work.all;
use std.textio.all;
entity TEST_BENCH is
end TEST_BENCH ;

-- The test bench architecture has two processes running concurrently with the first process waiting
for the user input and the second process displaying the signal status (10)
architecture STRUCTURAL of TEST_BENCH is

        -- The internal wires representing the highway andfarmway are initialized (11)
        signal C, ST, TRIG: BIT;
        signal DN: BIT := '1';
        signal HW: BIT_VECTOR(0 to 2) := "100";
        signal FW: BIT_VECTOR(0 to 2) := "001";
        signal INIT: BIT;
        component FSM is
                port(CLOCK, CART:in BIT;
                HIGHWAY: out BIT_VECTOR(0 to 2);
                FARMWAY: out BIT_VECTOR(0 to 2));
        end component ;

        component CLK is
                generic(HOT, CT, FOT:time);
                port(START:in BIT; DONE:out BIT; CLK:out BIT);
        end component ;

        for all:FSM use entity TRAFFIC_CONTROLLER(ALGORITHMIC);
        for all:CLK use entity TIMER(BEHAVIORAL);
begin
        C1: CLK
                generic map(20 ns, 2 ns, 1   ns)
                port map(ST, DN, TRIG);

        C2: FSM
                port map(TRIG, C, HW, FW)
```

-- *This is the user interface process which is responsible for giving the status prompt to the user and for receiving the car signal depending on the traffic (12)*

```
USER_INPUT : process
        variable TMP:character;
        variable P_STR:string(1 to 15) := "ARRIVAL STATUS>";
        variable UI, PROMPT: LINE;
begin
```

-- *the user is prompted at the beginning for an input (13)*

-- *after subsequent clock cycles the user is given the prompt (13)*

```
if(DN = '1' or INIT = '1') then
                while(true) loop
                        write(PROMPT, P_STR);
                        writeline(output, PROMPT);
                        readline(input, UI);
                        read(UI, TMP);
```

-- *this piece of code is for verification and execution (14)*

```
                        if (TMP = 'q' or TMP = 'Q') then
                                wait;
```

```
                    elsif (TMP = 'c' or TMP = 'C') the
                            C <= '1' after 1 ns,
                            '0' after 2 ns;
                            ST <= '1' after 1 ns,
                            '0' after 2 ns;
                            exit;
                    end if ;
            end loop ;
end if;
wait on DN, INIT;
end process USER_INPUT;

SIGNAL_DISPLAY: process(HW, FW)
variable FW_DISP, HW_DISP: string(1 to 25);
variable DISPLAY:LINE;
begin
-- This code prints the status of the highway signal (15)
        case HW is
                when "100" => HW_DISP := "HIGHWAY SIGNAL -> GREEN  ";
                when "010" => HW_DISP := "HIGHWAY SIGNAL -> YELLOW ";
                when "001" => HW_DISP := "HIGHWAY SIGNAL -> RED    ";
                when others => HW_DISP := "OUT OF ORDER             ";
        end case ;

        -- This code prints the signal status for th  farmway (16)
        case FW is
                when "100" => FW_DISP := "FARMWAY SIGNAL -> GREEN  ";
                when "010" => FW_DISP := "FARMWAY SIGNAL -> YELLOW ";
                when "001" => FW_DISP := "FARMWAY SIGNAL -> RED    ";
                when others => FW_DISP := "OUT OF ORDER             ";
        end case ;

                write(DISPLAY, HW_DISP & FW_DISP);
                writeline(output, DISPLAY);
        end process SIGNAL_DISPLAY;
```

```
-- This process initiates the user prompt and quits (17)
INIT_PROC : process
        begin
                INIT <= '1' after 1 ns,
                '0' after 2 ns;
                wait;
end process INIT_PROC ;

end STRUCTURAL ;
```

D.2 Histogram generator model commented by subject 2

```
-- these are standard IEEE libraries that contain the corresponding library functions (1)
library IEEE;
use STD.textio.all;
use IEEE.STD_LOGIC_MISC.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

-- this defines the entity of histogram (2)
-- plots the number of instances of a particular value retrieved from a binary file stores it in the ra
(2)
entity HISTGRAM is
  generic( ADDRLEN, WRDLEN : INTEGER;
          RD_DEL, DIS_DEL : TIME;
          INFILENAME, OUTFILENAME : STRING);

-- the variables with the directions of electrical signals are declared here (3)
port( DATA : inout STD_LOGIC_VECTOR (WRDLEN-1 downto 0) := (others => 'Z');
        ADDR : out STD_LOGIC_VECTOR (ADDRLEN-1 downto 0);
        START : in STD_LOGIC;
        CS, RD, WR : out STD_LOGIC);
end HISTGRAM;
```

```
-- This is the algorithmic architecture of the histogram (4)
architecture KEEP_TRACK of HISTGRAM is

-- the constants required for this program is declared here (5)
constant MAXADDR : INTEGER := 2**ADDRLEN - 1;
constant TRISTATE : STD_LOGIC_VECTOR(WRDLEN-1 downto 0) := (others => 'Z');
signal DATAREG : STD_LOGIC_VECTOR(WRDLEN-1 downto 0);

begin
-- this is an example of a process without a sensitivity list (6)
-- wait-statements prevent infinte loops (6)
HGM : process

-- All variables and file declarations are sandwiched between the process and begin statement (7)
variable L : LINE;
        variable VAL : INTEGER;
        file INFILE : TEXT is in INFILENAME;
        file OUTFILE : TEXT is out OUTFILENAME;
        variable BITDATA : BIT_VECTOR(WRDLEN-1 downto 0);
        variable BITADDR : BIT_VECTOR(ADDRLEN-   downto 0);
begin

-- wait-statement checks for a rising edge (8)
-- the process waits for this signal (9)
wait on START until (START = '1' or START = 'H');

-- The contents of the file are read till the end-of-file is sensed and it stores the number of
occurrences of a particular value (10)
while not(ENDFILE(INFILE)) loop
                READLINE(INFILE, L);
                READ(L, VAL);

-- The contents of the file are read till the end-of-file is sensed and it stores the number of
occurrences of a particular value (11)
ADDR <= CONV_STD_LOGIC_VECTOR(VAL,ADDRLEN);
```

```
-- This enables the READ and CHIP-SELECT signals and waits for a definite amount of time befor
reading the data from the DATA signal (12)
RD <= '1'; CS <= '1'; WR <= '0';
wait for RD_DEL;

-- the variable is converted to an integer (13)
VAL := CONV_INTEGER(DATA);

-- Read signal is disabled with a logic zero and also chip select (14)
RD <= '0'; CS <= '0';
wait for DIS_DEL;
DATA <= CONV_STD_LOGIC_VECTOR(VAL+1,WRDLEN);
WR <= '1'; CS <= '1';
wait for DIS_DEL;

-- write and chip select signals and waits until dis_del turns logic high (15)
-- the process waits until dis_del turns logic high (15)
WR <= '0'; CS <= '0';
wait for DIS_DEL;

-- Data signal is tristated to prevent bus contention (16)
DATA <= TRISTATE;
end loop;

-- This loop reads the memory contents and writes it to the output file (17)
for I in 0 to MAXADDR loop
ADDR <= CONV_STD_LOGIC_VECTOR(I,ADDRLEN);
RD <= '1'; CS <= '1'; WR <= '0';
wait for RD_DEL;
VAL := CONV_INTEGER(DATA);
RD <= '0'; CS <= '0';
wait for DIS_DEL;
DATA <= TRISTATE;
                WRITE(L, I);
                WRITE(L,   ');
                WRITE(L, VAL);
```

```
                        WRITELINE(OUTFILE, L);
end loop;
-- This is the end of histogram process (18)
end process HGM;
end KEEP_TRACK;
```

D.3 ALU model commented by subject 3

```
-- this vhdl program describes the system built using the lower level models built (1)
-- the control memory and the control logic are the other low level models (1)
-- the data unit is another lower level model used (1)

-- The IEE   std logic library is used in this model for simulation purposes (2)
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

-- the following user-defined libraries are used in this vhdl model (3)
use work.PRIMS.all;
use work.all;

-- the entity modeled is created here (4)
entity SYSTEM is
  port(X,Y:i   std_logic_vector(3 downto 0);
      CLEAR,START,INSTRUCTION,CLK: in std_logic;
      STATUS:out std_logic);
end SYSTEM;

-- Definition of the architecture of the system is described below (5)
architecture STRUCTURAL of  SYSTEM is
-- The address is initialized to 00000 (6)
signal B_ADDRESS:std_logic_vector(4 downto 0):="00000";

-- The CONTROL_DATA signal is a nine bit logic vector (7)
```

```
signal CONTROL_DATA:std_logic_vector(9 downto 0);

-- The component is mapped to a lower level model (8)
component CONTROL_LOGIC is
  port(CLEAR,START,INSTRUCTION,CLK: in std_logic;
       STATUS:out std_logic; OUT_ADDRESS:inout std_logic_vector(4 downto 0));
end component;

-- The component DATA_UNIT is mapped to a lower level model (9)
component DATA_UNIT is

-- the component input ports are mapped appropriately to the data bus (10)
-- the component output ports are mapped to the control bus (10)
port(DATA_X,DATA_Y:in std_logic_vector(3 downto 0);CONTROL_BUS: in std_logic_vector(9
downto 0);
       CLK:i   std_logic);
end component;

-- The component CONTROL_MEMORY is mapped to a lower level model (11)
component CONTROL_MEMORY is
  port(ADDRESS:in std_logic_vector(4 downto 0);
       OUT_VALUE:out std_logic_vector(9 downto 0));
end component;

-- Th   usage of the library components are defined here (12)
for CM:CONTROL_MEMORY use entity work.CONTROL_MEMORY(TABLE);
for CL:CONTROL_LOGIC use entity work.CONTROL_LOGIC(CONTROL);
for DU:DATA_UNIT use entity work.DATA_UNIT(BEHAVIORAL);
begin

-- The port mapping for the CONTROL-LOGIC module is done here (13)
CL:CONTROL_LOGIC
            port map(CLEAR,START,INSTRUCTION,CLK,STATUS,B_ADDRESS);

-- the control memory port mappings are connected as required (14)
CM:CONTROL_MEMORY
```

```
        port map(B_ADDRESS,CONTROL_DATA);
-- The DATA_UNIT port mappings are made here (15)
DU:DATA_UNIT
        port map(X,Y,CONTROL_DATA,CLK);
end STRUCTURAL;
```

D.4 Johnson counter model commented by subject 4

```
use work.all;
-- this entity generates the clock waveform to be used by the johnson counter (1)
-- the on- and off-period are controlled by the parameters hi_time and lo_time specified in the
testbench (1)
entity CLOCKGEN is
generic(HI_TIME, LO_TIME : time);
port(RUN : in bit; CLK : out bit := '0');
end CLOCKGEN;

-- this is the architecture for the entit   clockgen (2)
architecture ALG of CLOCKGEN is
begin

-- this process is inside the architecture (3)
-- working for the architectur   alg is defined here (3)
process
begin
-- this process waits for a logic high (4)
wait until RUN = '1';

-- when run-signal goes high the signal  clk is made high for hi_time (5)
while RUN = '1' loop
CLK <= '1';
wait for HI_TIME;
CLK <= '0';
wait for LO_TIME;
end loop;
end process;
end ALG;


use work.all;
-- this entity defines the interface of Johnson counter (6)
entity JC is
generic(RDEL, CLKDEL, EXTDEL : time);
port(QOUT: inout bit_vector(3 downto 0); CLK : in bit; RESET: in bit; LOAD: in bit; EXTIN : i
bit_vector(3 downto 0));
end JC;


-- this is the behavioral model for the entit   jc (7)
architecture BEHAVE of JC is
begin


-- the process contains the functionality for the architecture behavioral (8)
process(CLK, RESET, LOAD)
begin


-- the counter is reset when the reset-signal goes high (9)
if (RESET = '1') then
QOUT <= "0000" after RDEL;


-- the Johnson counter output is changed from one valid state to the next valid output state after
some delay. (10)
elsif (CLK'event and CLK='1' and LOAD='0') then
        case QOUT is
        when "0000" => QOUT <= "0001" after CLKDEL;
        when "0001" => QOUT <= "0011" after CLKDEL;
        when "0011" => QOUT <= "0111" after CLKDEL;
        when "0111" => QOUT <= "1111" after CLKDEL;
        when "1111" => QOUT <= "1110" after CLKDEL;
        when "1110" => QOUT <= "1100" after CLKDEL;
        when "1100" => QOUT <= "1000" after CLKDEL;
        when "1000" => QOUT <= "0000" after CLKDEL;
        when others => QOUT <= "0000" after CLKDEL;
        end case;

-- if load-signal is high and the clock goes high, external input istransfered to the jc  output (11)
elsif (LOAD = '1' and CLK'event and CLK='1') then
QOUT <= EXTIN after EXTDEL;
end if;
end process;
end BEHAVE;

use work.all;
architecture STRUCT of JC is
use work.all;

-- this defines a d flip-flop (12)
component D_FLIPFLOP
generic(CLKDEL, RDEL : time);
port(D, CLK, RST : in bit; Q, QC : out bit);
end component;

-- it gives the ports and delays associated with the 3-input and- gate used (13)
component GATE_AND3
generic(GDEL : time);
port(I1, I2, I3 : in bit; O : out bit);
end component;

-- this defines the two-input and-gate (14)
component GATE_AND2
generic(GDEL : time);
port(I1, I2 : in bit; O : out bit);
end component;

-- a two-input or-gate is defined (15)
component GATE_OR2
generic(GDEL : time);
port(I1, I2 : in bit; O : out bit);
```

```
end component;

component GATE_IN
generic(IDEL : time);
port(I1 : in bit; O : out bit);
end component;

-- these statements bind the components declared to the library model of the corresponding
component. (16)
for all : D_FLIPFLOP use entity FLIPFLOP(BEHAVE_FLIPFLOP);
for all : GATE_AND3 use entity EAND3(BEHAVE_AND3);
for all : GATE_AND2 use entity EAND2(BEHAVE_AND2);
for all : GATE_INV use entity EINV(BEHAVE_INV);
for all : GATE_OR2 use entity EOR2(BEHAVE_OR2);

signal DA1, DA2, DB1, DB2, DC1, DC2, DE1, DE2, LOADP, BCEZABEP, APCEZABC,
APBPEZBCE, APBPCPZAPCE, DA, DB, DC, DE, AP, BP, CP, EP, BCE, ABEP, APCE, ABC,
APBPE, APBPCP : bit;

begin

ffa : D_FLIPFLOP
generic map(CLKDEL => 11 ns, RDEL => 8 ns)
port map(D=>DA, CLK=>CLK, RST=>RESET, Q=>QOUT(3), QC=>AP);

ffb : D_FLIPFLOP
generic map(CLKDEL => 11 ns, RDEL => 8 ns)
port map(D=>DB, CLK=>CLK, RST=>RESET, Q=>QOUT(2), QC=>BP);

ffc : D_FLIPFLOP
generic map(CLKDEL => 11 ns, RDEL => 8 ns)
port map(D=>DC, CLK=>CLK, RST=>RESET, Q=>QOUT(1), QC=>CP);

ffe : D_FLIPFLOP
generic map(CLKDEL => 11 ns, RDEL => 8 ns)
port map(D=>DE, CLK=>CLK, RST=>RESET, Q=>QOUT(0), QC=>EP);
```

```
INV1 : GATE_INV
generic map(IDEL => 1 ns)
port map(LOAD, LOADP);


AND1 : GATE_AND3
generic map(GDEL => 3 ns)
port map(QOUT(2), QOUT(1), QOUT(0), BCE);

AND2 : GATE_AND3
generic map(GDEL => 3 ns)
port map(QOUT(3), QOUT(2), EP, ABEP);

AND3 : GATE_AND3
generic map(GDEL => 3 ns)
port map(ap, QOUT(1), QOUT(0), APCE);

AND4 : GATE_AND3
generic map(GDEL => 3 ns)
port map(QOUT(3), QOUT(2), QOUT(1), ABC);

AND5 : GATE_AND3
generic map(GDEL => 3 ns)
port map(AP, BP, QOUT(0), APBPE);

AND6 : GATE_AND3
generic map(GDEL => 3 ns)
port map(AP, BP, CP, APBPCP);

-- following gates select input signals and feeds them to the flip-flops used (17)
-- two and-gates are inputs to flipflop-a (17)
AND7 : GATE_AND2
generic map(GDEL => 3 ns)
port map(LOAD, EXTIN(3), DA1);
```

```
AND8 : GATE_AND2
generic map(GDEL => 3 ns)
port map(LOADP, BCEZABEP, DA2);

-- following two and-gates feed flipflop-b (18)
AND9 : GATE_AND2
generic map(GDEL => 3 ns)
port map(LOAD, EXTIN(2), DB1);

AND10 : GATE_AND2
generic map(GDEL => 3 ns)
port map(LOADP, APCEZABC, DB2);

AND11 : GATE_AND2
generic map(GDEL => 3 ns)
port map(LOAD, EXTIN(1), DC1);

AND12 : GATE_AND2
generic map(GDEL => 3 ns)
port map(LOADP, APBPEZBCE, DC2);
```

D.5 Histogram model commented by subject 5

```
-- the declartions are needed for including the various component libraries (1)
-- these libraries contain the various component declarations (1)
library IEEE;
use STD.textio.all;
use IEEE.STD_LOGIC_MISC.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

-- this is the begining of the entity declaration for the hitogram (2)
-- this describes the various input signals and the corresponding output signals (2)
entity HISTGRAM is
```

```
  generic( ADDRLEN, WRDLEN : INTEGER;
             RD_DEL, DIS_DEL : TIME;
             INFILENAME, OUTFILENAME : STRING);

-- the data is an inout type (3)
-- the addr signal contains the address location of the data( 4)
-- chip select represented b   cs (5)
-- the read signal is represented by rd and write represented b   wr (5)
port( DATA : inout STD_LOGIC_VECTOR (WRDLEN-1 downto 0) := (others => 'Z');
         ADDR : out STD_LOGIC_VECTOR (ADDRLEN-1 downto 0);
         START : in STD_LOGIC;
         CS, RD, WR : out STD_LOGIC);
end HISTGRAM;

-- the architecture description starts here (6)
architecture KEEP_TRACK of HISTGRAM is

-- the architecture has a few constant  needed for the logic design (7)
constant MAXADDR : INTEGER := 2**ADDRLEN - 1;
constant TRISTATE : STD_LOGIC_VECTOR(WRDLEN-1 downto 0) := (others => 'Z');
signal DATAREG : STD_LOGIC_VECTOR(WRDLEN-1 downto 0);
begin
  -- The process HGM represents the histogram generator (8)
 HGM : process
-- various variables required for the hgm are declared (9)
-- they are also initialized (9)
variable L : LINE;
variable VAL : INTEGER;
file INFILE : TEXT is in INFILENAME;
file OUTFILE : TEXT is out OUTFILENAME;
variable BITDATA : BIT_VECTOR(WRDLEN-1 downto 0);
variable BITADDR : BIT_VECTOR(ADDRLEN-   downto 0);

begin

-- the process waits for the rising edge signal (10)
```

```
wait on START until (START = '1' or START = 'H');

-- this is a loop to read the contents of the input file (11)
--  it processes each input when it is read (12)
-- this process continues  till the end of the file is reached (13)
while not(ENDFILE(INFILE)) loop
                READLINE(INFILE, L);
                READ(L, VAL);

-- the following statement converts the incoming address to a vector of typ   std_logic (14)

ADDR <= CONV_STD_LOGIC_VECTOR(VAL,ADDRLEN);

--  This enables the READ and CHIP-SELECT signals and waits for a definite amount of tim
before reading the data from the DATA signal (15)
RD <= '1'; CS <= '1'; WR <= '0';
                         wait for RD_DEL;
 -- the data that is read is now converted to an integer (16)
-- it is then stored (16)
VAL := CONV_INTEGER(DATA);

-- The READ and CHIP-SELECT signals are now disabled and the system waits for some time (17)
RD <= '0'; CS <= '0';
wait for DIS_DEL;

-- the incremented value of the data is now placed back on the data signal (18)
DATA <= CONV_STD_LOGIC_VECTOR(VAL+1,WRDLEN);

-- the write and chip-select signals are asserted (19)
-- the memory stores the incremented data (19)
WR <= '1'; CS <= '1';
wait for DIS_DEL;

-- after waiting for some time the asserted signals are de-asserted (20)
-- the data signal is tri-stated  so that other entities can write to it (21)
WR <= '0'; CS <= '0';
```

```
wait for DIS_DEL;
DATA <= TRISTATE;
end loop;
-- the procedure described reads the data from the memory in serial order (22)
for I in 0 to MAXADDR loop
        ADDR <= CONV_STD_LOGIC_VECTOR(I,ADDRLEN);
        RD <= '1'; CS <= '1'; WR <= '0';
        wait for RD_DEL;
        VAL := CONV_INTEGER(DATA);
        RD <= '0'; CS <= '0';
        wait for DIS_DEL;
          DATA <= TRISTATE;
          WRITE(L, I);
          WRITE(L,   ');
          WRITE(L, VAL);
           WRITELINE(OUTFILE, L);
end loop;
end process HGM;
end KEEP_TRACK;
```

VITA

Pradeep Victor is a graduate student, doing his MS in Computer Engineering at Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, Virginia.

He will be taking up a position as a software engineer at Lockheed Martin Global Telecommunications in Maryland.