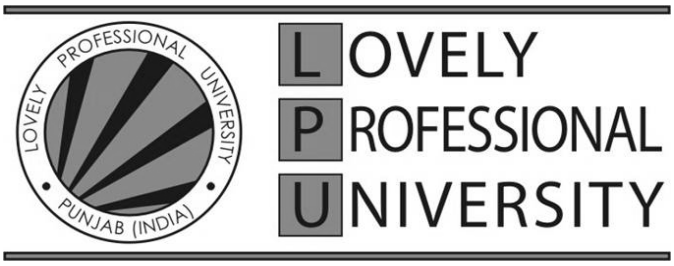

IMPROVEMENT IN AUTOMATED MODEL BASED TESTING BY NATURAL LANGUAGE APPROACHES

A Dissertation

Submitted By

Priyanka Nanda

to

Department of Computer Science and Engineering
In partial fulfilment of the Requirement for the

Award of the Degree of

Master of Technology in Computer Science and Engineering

Under the guidance of

Mr. Makul Mahajan

(May 2015)

ABSTRACT

Models in particular finite state machine models – provide an invaluable source of information for the derivation of effective test cases. However, models usually approximate part of the program semantics and capture only some of the relevant dependencies and constraints. As a consequence, some of the test cases that are derived from models are infeasible. The primary objective is to generate model-based system and acceptance test cases considering Natural Language requirements deliverables. The generation of Executable Test Cases which predicted behaviours that did not exist in the expert's approach.. Model Checking combined with k permutations of n values of variables and specification patterns were used to address this goal. Models in particular finite state machine models – provide an invaluable source of information for the derivation of effective test cases. However, models usually approximate part of the program semantics and capture only some of the relevant dependencies and constraints. As a consequence, some of the test cases that are derived from models are infeasible. We will use NLP-MBT tool in which different test case will execute according to N-gram statistics.

CERTIFICATE

This is to certify that Priyanka Nanda has completed M.Tech dissertation titled **"IMPROVEMENT IN AUTOMATED MODEL BASED TESTING BY NATURAL LANGUAGE APPROACHES"** under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. No part of the dissertation has ever been submitted for any other degree or diploma.

The dissertation is fit for the submission and the partial fulfillment of the conditions for the award of M.Tech Computer Science & Engg

Date:

(Mr. Makul Mahajan)
Assistant Professor in CSE
Department of Computer Science & Engineering
Lovely Professional University,
Phagwara

ACKNOWLEDGEMENT

I would like to take this opportunity to express my deep sense of gratitude to all who helped me directly or indirectly during dissertation-work.

Firstly, I would like to thank my advisor **Mr. Makul Mahajan (Assistant Professor in CSE)** for being great mentor and best adviser I could ever have. His advice, encouragement and critics are sources of innovative ideas, inspiration and cause behind the successful completion of this dissertation. I am highly obliged to all faculty members of computer science and engineering department for their support and encouragement.

I would like to express my sincere appreciation and gratitude towards my friends for their encouragement, consistent support and invaluable suggestions at the time I needed the most.

I am grateful to my family for their love, support and prayers.

PRIYANKA NANDA

Regd no. 11312562

DECLARATION

I hereby declare that the dissertation entitled, "**IMPROVEMENT IN AUTOMATED MODEL BASED TESTING BY NATURAL LANGUAGE APPROACHES**" submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: **(Priyanka Nanda)**
Reg. No. 11312562

Table of Contents

LIST OF FIGURES

\

CHAPTER 1

INTRODUCTION

1.1 Software Engineering

Software engineering is the learning and presentation of engineering to the plan, progress and keep of software. It is an engineering reprimand that is concerned with all facets of software creation. Software engineering is a great, multifaceted, and nonfigurative subject it is problematic to hypothesis vigorous learning trainings that build on the student's basic knowledge of programming and tranquil teach elementary software engineering ideologies. It is also the case that launch students stereotypically know how to build small programs, but they have petite skill with the procedures necessary to harvest consistent and continuing maintainable components. It mainly focus on step-by-step that points students toward the structure of exceedingly reliable trivial components using well known, greatest-applies software engineering performances[21]. Software progress is a speedily varying, knowledge-rigorous business involving many people working in different segments and events. In software development, every person involved regularly makes technical or managerial decisions. Most of the time, team associates make decisions based on personal knowledge and skill or knowledge gained using casual links. This is viable in small administrations, but as establishments grows and knobs a larger volume of information, this process becomes inept. Large organizations cannot rely on casual sharing of employees' personal information. Individual information must be shared and leveraged at project and association levels. Administrations need to define processes for sharing information so that employees throughout the organization can make accurate decisions. [24]

Software engineering is a covered knowledge, referring to figure 1.1. It is the procedure that grasps the knowledge layers together and enables balanced and appropriate development of computer science. Software process defines a basis that must be recognized for actual transmission of software engineering [23].

1.1.1Software Engineering Phases:

Software Development Life Cycle is classified into different stages which determined of better scheduling and organization. SDLC can be divided into ten phases. They are:

- Requirements specification: Analysis, specification, and authentication of requirements for software.
- Software design: The process of defining the building, mechanisms, boundaries, and other features of a system.
- Software coding: The process of coding, verification, unit testing integration testing and corrections is to be done
- Software testing: The process of defining the behaviour of system and check whether the system is to be work properly.
- Software maintenance: The process where how the system is maintained properly so that unauthorized persons cannot corrupt the systems. One of the most important considerations is cost in software maintenance.
- Software configuration management: The documentation of the configuration of a system at separate points in time for the purpose of systematically monitoring changes to the configuration, and maintaining the integrity of the configuration throughout the system life cycle.
- Software engineering management: The process which defines following activities like planning, coordinating, , monitoring, controlling, and reporting.
- Software engineering process: The process where meaning, operation, calculation of the software life cycle process is included.
- Software engineering tools and methods: The process where some tools used like computer aided design such ad Auto-Cad, Corel Draw, etc.
- Software quality management: The process which fulfills the customer needs and giving good quality to them. [25].

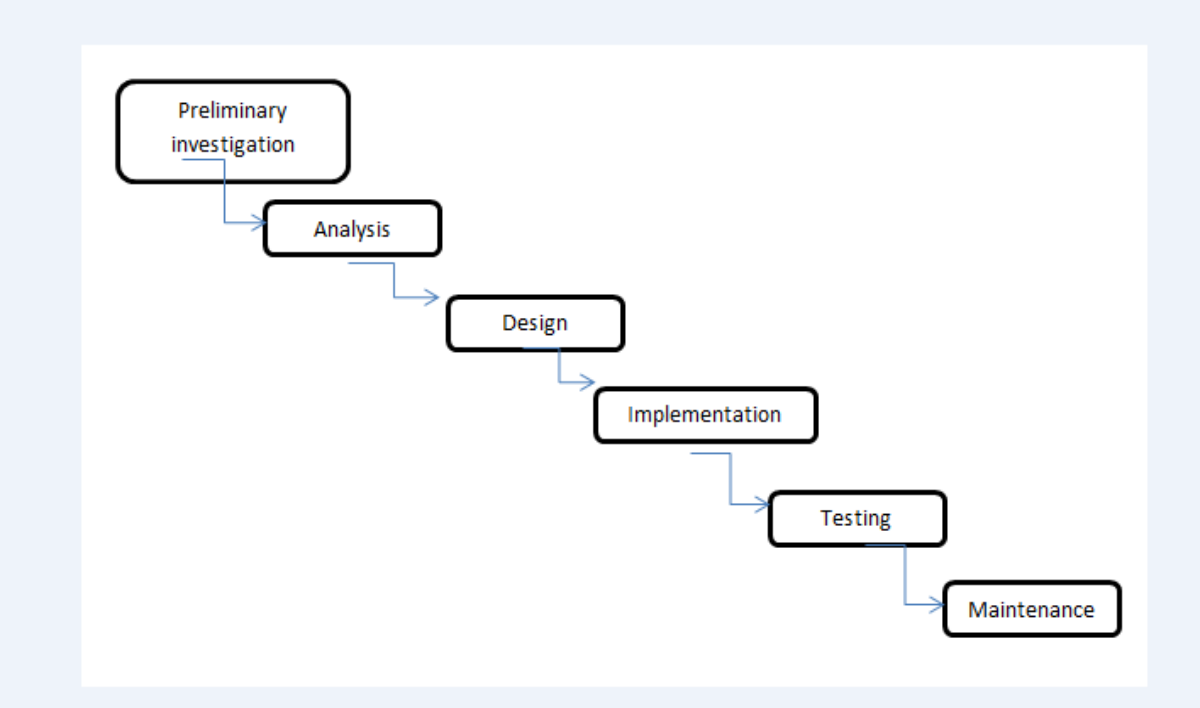

Figure 1.1 SDLC Phases

1.2 Software Testing

Software testing is a learning showed quality of the product. The process of defining the behaviour of system and check whether the system is to be work properly. In software testing, following are the different properties specify the degree to which the component or system under test:

- meets the requirements that focused its design and development,
- replies properly to all kinds of inputs,
- performs its functions within an suitable time [26].

1.2.1 Types of Software testing:

- Black box testing – Internal system design is not measured in this type of testing. Tests are built on requirements.
- White box testing – This testing is built on information of the internal logic of an application's code. It is also known as Glass box Testing. Internal software and code working should be recognized for this type of testing.

- Unit testing – Testing of different software components or modules. It is typically done by the programmer
- Integration testing – In this, all the modules are combined to perform integration testing. This type of testing is particularly applicable to client/server and distributed systems.
- System testing – In this testing, entire system is to be tested as per the requirements [27].

1.2.2 Model Based Testing:

Model-based testing is used for designing model-based and executes artifacts that are to perform software testing or system testing. Models are used to represent testing approaches. In a model based testing, a model concerning a SUT which is typically an abstract that defines the behavior of system under test.[8][11].

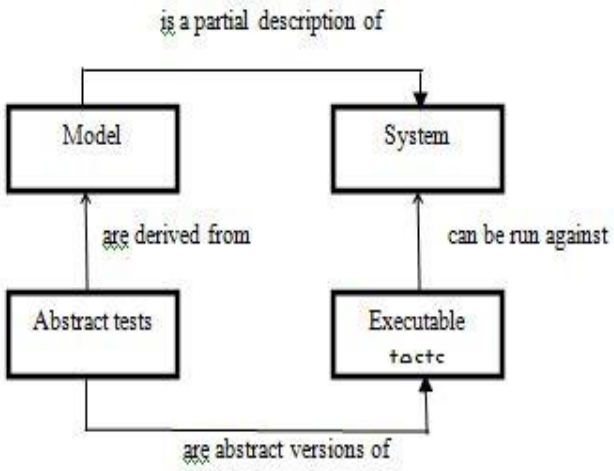

Figure 1.2 Graphical view of Model Based Testing

1.2.3 Model Based testing approaches

Model based testing offers a system for automatic generation of test cases using models mined from software artifacts. MBT approach has three essentials:

- Software comportment

- Measures
- Generate supporting substructure for the tests.

Basically, MBT has two main approaches:

1. Offline MBT
2. Online MBT

Offline MBT:

- It permits automate test execution in third party test execution platform.
- It makes possible to create a tool chain.
- It yields determinate sets of tests and executes [28].

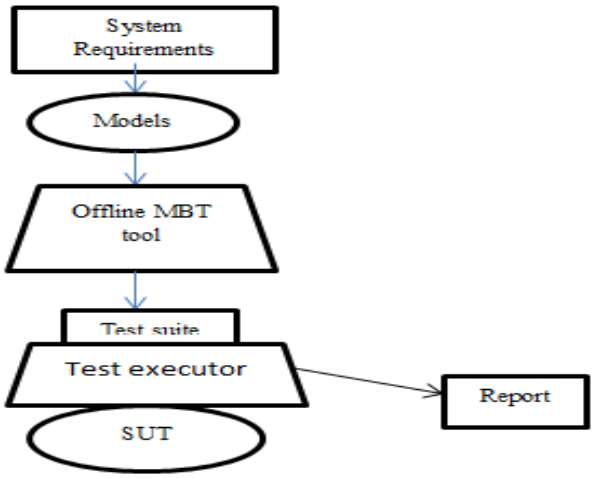

Figure 1.3: Offline Model Based Testing

Online MBT:

- It produce test case which are in performance.
- It stimulating non-deterministic system.
- In it, infinite test suite is repeatedly [28].

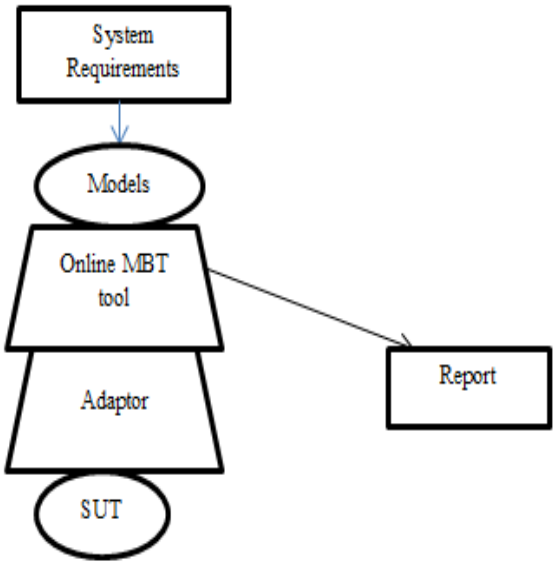

Figure 1.4: Online Model Based Testing

1.3 Natural Language Processing

Natural language is defined to the language spoken by people, e.g. English, Japanese, as different to artificial languages, like C++, Java, etc.

Natural Language Processing is the field of computer science and phonology which is concerned with the communications between computers and human languages. In the term of theory, we can say that natural-language processing is a very attractive method of human-computer interaction. Natural-language is sometimes stated to as an AI-complete problem, because natural-language recognition appears to require wide knowledge about the outside world and the ability to manipulate it [24].

1.3.1 NLP Terminology

The vocabulary of NLP includes tokens, sentence, tokenization, corpus, part-of-Speech (POS) Tag, and parsing, it can be described as follows [25].

- **Token:** The process of breaking input text into linguistic units such as words, punctuation, numbers or alpha numeric, is known as tokenization. These units are known as tokens.
- **Sentence**: The sentence is well-defined as an ordered sequence of tokens.

- **Tokenization:** It is well-defined as the process of breaking a sentence into its constituent tokens.
- **Corpus:** It is well-defined as a body of text that typically contains a large number of sentences.
- **Part-of-speech (POS) Tag**: A word can be categorized into one or more of a set of lexical or part-of-speech categories such as Nouns, Verbs, Adjectives and Articles.

1.3.2 Parse Tree

A parse tree is well-defined as a tree that represents the syntactic structure of the sentence as defined by a formal grammar.

Following are some mutual tasks that are performed by NLP which is as given below:

- **POS Tagging**: The process of categorizing words into their parts of speech and labelling them accordingly is known as part-of-speech tagging[4].
- **Computational Morphology**: Computational morphology is well-defined which is concerned with the discovery and analysis of the internal structure of words using computers [4].
- **Parsing:** Parsing is a process of examining a sentence by taking each word and determining its structure from its module parts [7].

1.4 N-Gram

An *n*-gram model is a type of probabilistic language model for predicting the next item in a sequence (n-1). N-gram models are widely used in probability, computational linguistics During the last fifty years, we have witnessed a significant increase of embedded HW-SW components in critical systems. Clearly, this trend goes along with increased software size and complexity, and strongly impacts critical systems' safety and reliability. Currently, many researchers are focusing on how to achieve the safety and reliability levels required for these systems. Some approaches to deal with such a problem rely on Model Based Testing (MBT) techniques. However, these techniques usually take as input models (e. g., state diagrams) that are usually not yet available in the very beginning of the system development project. In the initial phases, only high-level

and textual requirement descriptions are usually available. Therefore, the use of MBT is postponed.

To enable early MBT, we propose NAT2TESTIM R —an approach to generate test cases from requirements described in Controlled Natural Language (CNL) based on the RT-Tester3 Internal Model Representation (IMR)[12]. The requirements can describe temporal properties besides functional behaviour. We opt for receiving textual requirements as input instead of a graphical notation because the former is usually available first and in some industries it is required to have textual descriptions for certification purposes.

Initially, our approach parses the textual system requirements to evaluate their conformance with the CNL structure. Our CNL (the System Requirement-CNL) is a non-ambiguous and precise subset of the English language. After parsing, our approach provides a semantic interpretation for the requirements, using verb case frames as semantic representation[12]. This idea was first developed by the authors in a previous work [5] , and this paper extends our original ideas. From the case frames, the requirements' semantics are mapped into an internal model representation whose formal semantics is given by means of a transition relation. Based on this model, our approach generates test vectors with the support of the RT-Tester and its SMT solver. This whole process is fully automated by supporting tools.

The tests generated by NAT2TESTIM R provide means for early testing/simulation of models at design level. To evaluate our proposal, we applied it to four examples from different domains: (i) a Vending Machine (a toy example); (ii) a control system for Safety Injection [15] in a Nuclear Power Plant (publicly available); (iii) one example provided by Embraer4 (a Brazilian aircraft manufacturer); and (iv) part of the Turn Indicator System [16] of today's Mercedes vehicles (publicly available5).

The NAT2TESTIM R approach was evaluated from three perspectives:

(i) performance

(ii) automatically generated versus manually written test vectors (by Embracer)

(iii) Mutant-based strength analysis.

Within seconds, our approach generated 94% of the test vectors manually written by Embracer specialists. Moreover, considering a mutant-based strength analysis, our

approach yielded a mutation score between 54% and 98%.Therefore, the main contributions of this work are: [2] an MBT approach for generating tests from textual requirements, [4] a formal representation of case frames by means of a transition relation, and [4] empirical evaluations of our approach considering four examples from different domains.

1.5 Python Language

Python is a new kind of scripting language, and most scripting languages it is constructed about an interpreter. Many outdated scripting and interpreted languages have forwent syntactic lucidity to simplify parser building; consider e.g. the tender grammar needed to compute the value of modest expressions like a+b*c in Lisp, Smalltalk or the Bourne shell. Others, e.g. APL and Perl, provision arithmetic expressions and other conveniences, but have made cryptic one-liners into a sculpture form, turning program keep into a nightmare. Python programs, on the other hand, are neither stiff to neither write nor stiff to read, and its expressive power is comparable to the languages said above. Yet Python is not big: the entire interpreter fits in 200 kilobytes on a Macintosh, and this even comprises a windowing interface Python is used or planned as an application development language and as an extension language for non-expert programmers by several profitable software vendors. It has also been used successfully for numerous large non-commercial software projects [22].

CHAPTER 2

REVIEW OF LITERATURE

A.Pretschner*et al* (2012)proposed that the models of system under test which are depends on model based testing that develop test cases for the system. Here authors discussed the classification of main features that cover the model based testing methods that show that how to classify so that it should be considerate the comparisons and modification of model based testing methods. In this paper, authors have discussed different tools and methods to improve the scalability which increase the performance of test generation [1].

H.Samih*et al* (2014)proposed a Model based testing for PL-usage models has proposed to reinforce model based testing that provides automatic test case generation which furnish with variability information. , it show a supported tool that allow model based testing which generate test cases for not only one product but different developments [111

Bernhard Rumpe(2014) proposed to check what properties a modelling UML sequentially support extreme programming technique well. It uses XP which is an explicit reaction to the complexity of today's modelling methods like the Unified Process, the Open Toolbox of techniques are needs to make UML suitable for an extreme modelling approach [3].

CyrillaArtho*et al* (2013) proposed to display test sequences of application programming interface calls which engross model based testing with different system configuration. It proposed to try SAT solvers techniques which used for verification back-ends that generates sequences of valid API for progressive feature of SAT solvers [5].

Julien Botella *et al* (2013) proposed the procedures of model based testing has proposed that where some application is done in the scene of a qualification testing phase made by at autonomous designers, developers and sponsors of the cryptographic components under test appeal on security cryptographic components. It will work upon the bid of MBT

techniques which use MBT for pure functional testing that is the test generation model and the test selection criteria [12].

Mark Harman*et al* (2013) proposed to use the slant of Oracle automation that is the main key to isolate present constriction which hampers unstinting general automation for tests and Oracle automation includes modeling, specifications, contract driven development and metamorphic testing. This paper also tells the ample report of Oracles in software testing which describe implied attitude that gives some endowment for the lack of Oracle [15].

Briand *et al* (2012) have proposed commonly used FSM model which is use in UML, class and sequence diagram. Here author can represent the values of class attributes and the graphical objects. Here author discussed how to represent abstract and concrete applications where each FSM represents Authors discussed different transitions in FSM which signifies action or event related to an application. Here mainly work depends on how to perform event or action in an application and how to call method so that application state can change during execution. [2].

Dudekula Mohammad Rafi *et al* (2012) have proposed that how space is near between both views by inspecting in respect of the advantage and bound of test automation. This paper builds analysis of some advantages and drawback of software test automation in educational information. This tells how to unify actor view of software test automation [8].

Mohamed Mussa*et al* (2012) have proposed that how to brings some idea based on model based performances that use UML2 Testing Profile for generating integration test cases from unit test models and how to construct integration test model that use for UTP models [16].

Petra Brosch*et al* (2012) have proposed that how to use of overlapping information which innate in multiple views of models for automatic testing has proposed. The authors have proposed to use multi-view modeling languages like UML that offer different diagram types to lower the complexity of re-counting software systems where each

diagram allowing for splitting a complex model into various areas of concern. So, in that way, the diagrams are complemented with one another, that work together to provide a holistic representation of the system. Here we find that how the information can be used as test data [17].

Yoav Bergner*et al* (2012) have proposed that how collaborative filtering is applied to use dichotomously scored student response data and find optimal parameters for each student and item based on cross-validated prediction accuracy. To use CF, it is fast, stretchy and firm [20].

Cristran Cadar*et al* (2011) proposed to use symbolic executions which is a program analysis performance used for solving restraint in technology that increased availability of computational power. In this paper, it become able to all plainly that how modern symbolic execution slant empower organized testing for bug finding and symbolic execution used for handle the expanding number of paths in the code [6].

Gervazi*et al* (2011)proposed a formal framework for identifying, analysing and managing inconsistency in natural language requirements derived from multiple stakeholders. In this article, a particular inconsistency namely logical contradiction (any situation in which some fact α and its negation ¬α can be simultaneously derived from the same specification) was concentrated [10].

Dias Neto*et al (*2011)proposed to define the behaviours which are appropriate for measuring the testing. In this phase is also known as group related to test. Here authors take an example of FSM models which is used as an test case which showing output that involves sow to classify an events [7].

ChristelBaier*et al* (2010)proposed the checking of Model.Gervazi also proposed a methodology for the lightweight validation of natural language requirements. In this paper it tells validation as a decision problem [4].

Kedian Mu*et al* (2008)proposed the priority -based scoring vector, which participates the measure of the degree of inconsistency with the measure of the significance of inconsistency. Here author discussed for checking the inconsistencies in natural language requirements [14].

El-Far *et al* (2007)proposed the model-based testing which is an method that bases common events of the software testing process such as test case generation and test results evaluation [9].

Utting*et al* (2006) proposed a model-based testing which consists of a test strategy in which test cases are derived completely from a model that describes some feature of software. In this paper authors discussed the behaviour or structure of the software which has been formalized by means of models with well-defined rules such as UML diagrams. Here authors also discussed that a model-based testing technique can be applied to any type of testing (functional, structural, etc[19].

Sarma*et al* (2005)In this paper, authors proposed that the technique which are depend on the source based on UML show system state graph where use case models, sequence diagrams, and State chart models represent. In this paper, authors discussed to cover the transition path coverage Here authors discussed that how the work can interact with users during the process.[18].

Jurafsky*et al* (2004)proposed to differentiate six categories of the knowledge of language that is needed to engage in complex language behaviour: Phonetics and Phonology, Morphology, Syntax, Semantics, Pragmatics, and Discourse. Here authors discussed that how to use a novel semantic encoding of the CNL behaviour in the form of a timed transition relation [13].

CHAPTER 3

PRESENT WORK

3.1 PROBLEM FORMULATION

3.1.1 Model base testing

In model-based testing, various types of models can be used in automated models which is to define the appropriate subclass of application behaviours that is to be considered for testing and this stage is termed group related to test.

3.1.2 N-Gram Approaches

• This is prediction model using the n gram for prediction.

• NLP uses in sentence derivation which is use for prediction of sentences, which have many

possibilities.

• If we resemble above give point and model based testing is the same because model based

testing have many test cases which can use any point but some test cases is useful ,so we can use n gram statics for predicting these useful test cases.

• N-gram approach useful for reduce the complexity of model based testing and increasing

the feasibility of model based testing.

3.1.3 Problem statement

The model based testing have many way to select the test cases but some test cases is useful, so we can say model-based testing is non deterministic approach, we can reduce this by N-gram statistics of test cases and reduce the complexity of model based testing. Whatever generation has to come, we have to avoid these all generation of infeasible test sequences and expect the feasible sequences by using N-Gram approach and it is same as sentences. N-gram statistics can also be used in same way as in NLP to achieve such purpose. NLP also generate event sequences that contain N-gram previously observed in

real executions, the probability that such sequences will in turn be executable is increased.

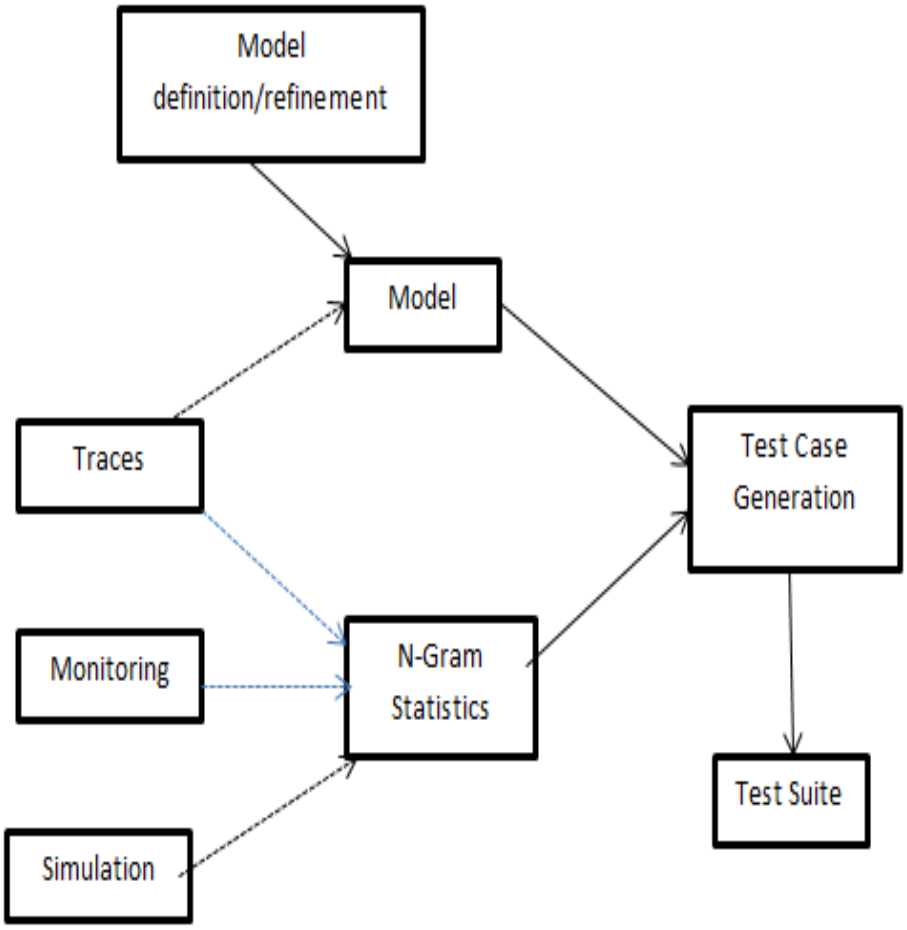

Figure 3.1 Problem Statement of proposed work

3.2 OBJECTIVES

Objectives is to comprehensive study of Natural Language Processing and Model based testing by using the N-gram Approach on Model based testing so that it reduce the complexity of model based testing by predict feasible transition of states event. In this model-based system and acceptance test case generation, and particularly taking into account NL requirements documents, identification of scenarios, their respective models and test case generation are used. There are many benefits related to formal methods, such approaches are not largely adopted for software development in general. On the other hand, NL is still widely used to develop software requirements specifications.

- To implement N-Gram approach on model-based testing.
- To reduce the possibility of occurrence of different transitions.
- To convert non-deterministic approach to deterministic approach

3.3 PROPOSED METHODOLOGY

N -gram statistics play important role in prediction next test cases in test suite. But it is important to using this in model based testing

1. Probabilistic model for prediction next word in sentence same as model based testing predict best test cases in test suite
2. N-gram statistics best possible combination of states in test suite.
3. N-gram Statistics reduce the event sequences by prediction approach. Avoiding the generation of infeasible event sequences is very similar to avoiding the derivation of sentences.
4. Depending on the application the most appropriate among these three data collection methods may different.

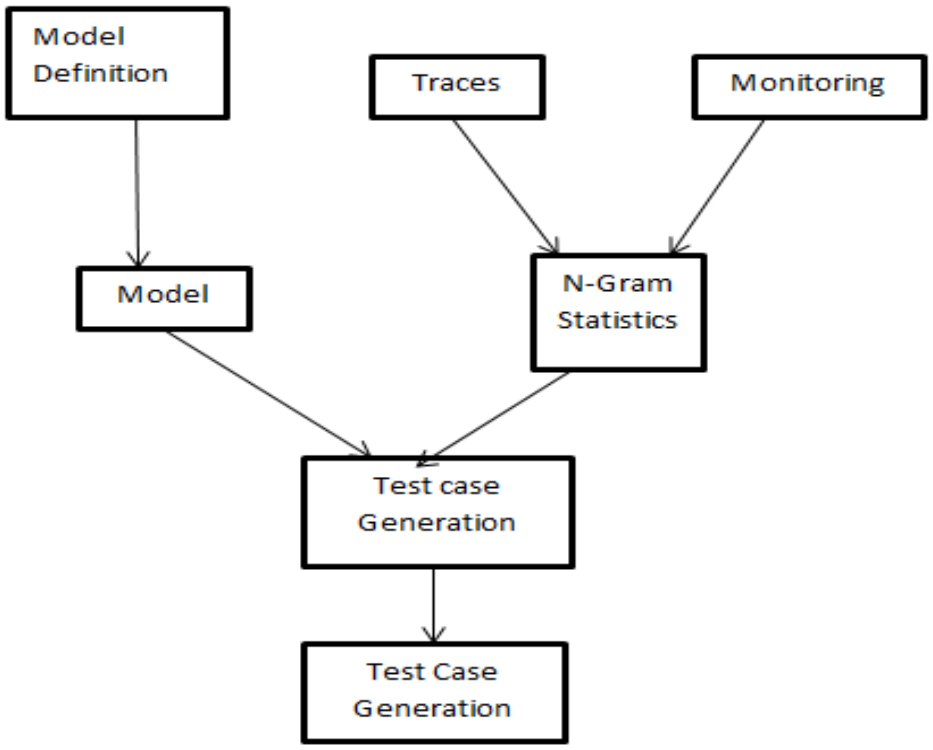

Figure 3.2: Flow Chart of research methodology

Figure 3.2: shows a high level view of the proposed approach. While graph visit test case generation algorithms (Random, Depth first and Breadth first) require just one input (i.e., the model), N-Gram based test case generation needs two inputs: model and N-gram statistics. The model can be defined manually by the user; it can be inferred automatically from execution traces using state abstraction or event sequence abstraction or a mixed approach can be followed, in which the model is first inferred and then it is manually refined by the user.

Tools Used: Model based NLP MBT

Model-based testing (MBT) is application of model-based design for designing and optionally also executing artifacts to perform testing or system testing. Models can be used to represent the desired behaviour of a System Under Test (SUT), or to represent testing strategies and a test environment. A model describing a SUT is usually an abstract, partial presentation of the SUT's desired behaviour. Test cases derived from such a model are functional tests on the same level of abstraction as the model. These test cases are collectively known as an abstract test suite. An abstract test suite cannot be directly executed against an SUT because the suite is on the wrong level of abstraction. An executable test suite needs to be derived from a corresponding abstract test suite. The executable test suite can communicate directly with the system under test. This is achieved by mapping the abstract test cases to concrete test cases suitable for execution. In some model-based testing environments, models contain enough information to generate executable test suites directly. In others, elements in the suite must be mapped to specific statements or method calls in the software to create a concrete test suite. This is called solving the "mapping problem". In the case of online testing (see below), abstract test suites exist only conceptually but not as explicit artifacts.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Parameters for testing results:

- **Feasibility:** This is the Ratio between feasible test sequences and total number of test sequences generated by each test strategy.
- **Coverage:** This is the Ratio between covered transitions and total number of transitions in the model.
- **Size of test cases:** this is the Number of test sequences in the test suite. (Test suite size)
- **Length of test:** This is the Average number of events in the test (Test case Length) sequences added to each test suite.

4.2 Feasibility with all test cases in Flexi- store software

	Feasibility	Coverage (%)	Size of test cases	Length of test case	
Depth first	2%	58	4	25.34	
Breadth first	11%	87	34	3.38	
Random	72%	65	4.5	55.01	
Ngram-2	65%	88	4	47.73	
Ngram-3	4%	73	4.3	49.66	
Ngram-4	27%	88	5	45.48	
Interpolated3	39%	79	14.33	54.15	Flexi-store
Interpolated4	78%	89	13	53.41	
Interpolated5	57%	8	12	51.27	
Interpolated6	81%	89	10	55.95	
Interpolated7	46%	8	9.7	53.63	
Interpolated8	71%	88	9	55.01	
Interpolated9	43%	8	8.39	57.18	
Interpolated10	86%	87	8	56.5	

4.2.1

	Feasibility
Depth first	2%
Breadth first	2%
Random	11%
Ngram-2	72%
Ngram-3	65%
Ngram-4	4%
Interpolated3	27%
Interpolated4	39%
Interpolated5	78%
Interpolated6	57%
Interpolated7	81%
Interpolated8	46%
Interpolated9	71%
Interpolated10	43%

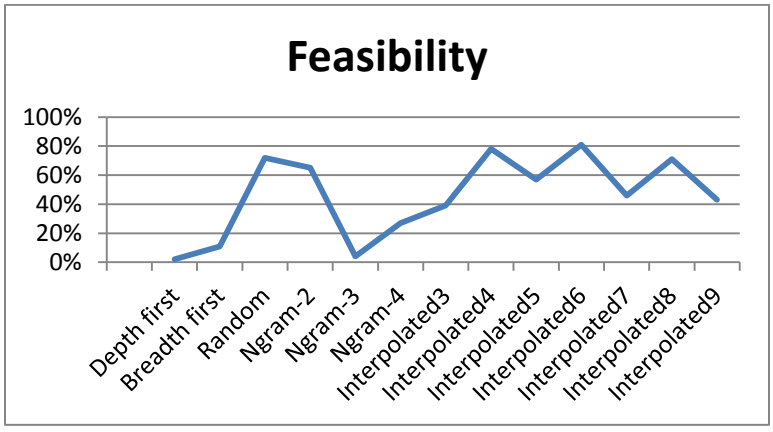

Figure 4.1(Feasibility Vs total no of test sequences in flexi-store)

Feasibility of Breadth first, Depth first and Random Approaches less than N-gram and interpolated N-gram approaches (INTERP), it will increase when increase N-gram value of n. It show prediction increase when N-gram increase.

4.2.2

	Coverage (%)
Depth first	58
Breadth first	87
Random	65
Ngram-2	88
Ngram-3	73
Ngram-4	88
Interpolated3	79
Interpolated4	89
Interpolated5	8
Interpolated6	89
Interpolated7	8
Interpolated8	88
Interpolated9	8
Interpolated10	87

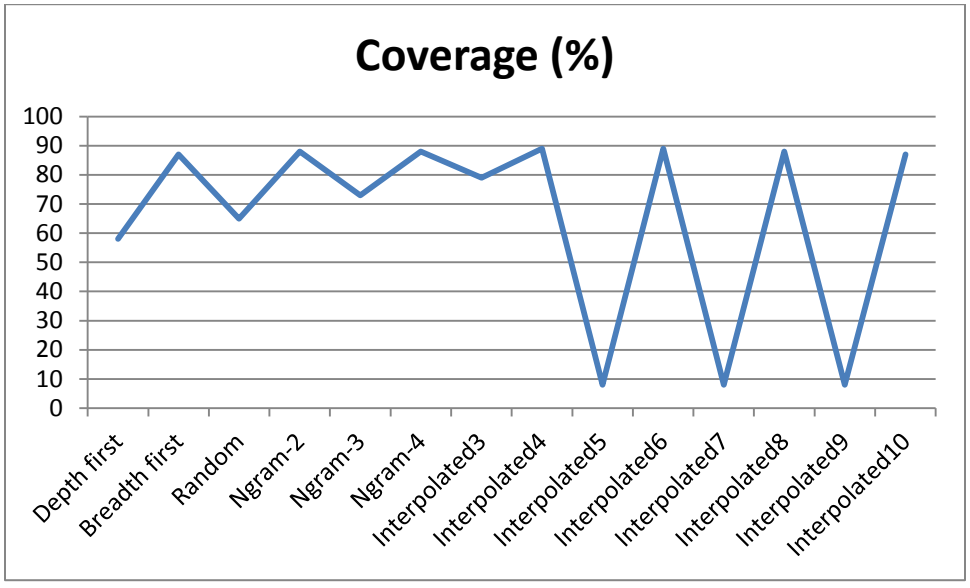

Figure 4.2(Coverage Vs total number of test sequences in flexi-store)

Here step by step feasibility of Breadth first, Depth first and Random approaches increases when N-Gram value increase and simultaneously N-Gram approaches (interpolated) increases.

4.2.3

	Size of test cases
Depth first	4
Breadth first	34
Random	4.5
Ngram-2	4
Ngram-3	4.3
Ngram-4	5
Interpolated3	14.33
Interpolated4	13
Interpolated5	12
Interpolated6	10
Interpolated7	9.7
Interpolated8	9
Interpolated9	8.39
Interpolated10	8

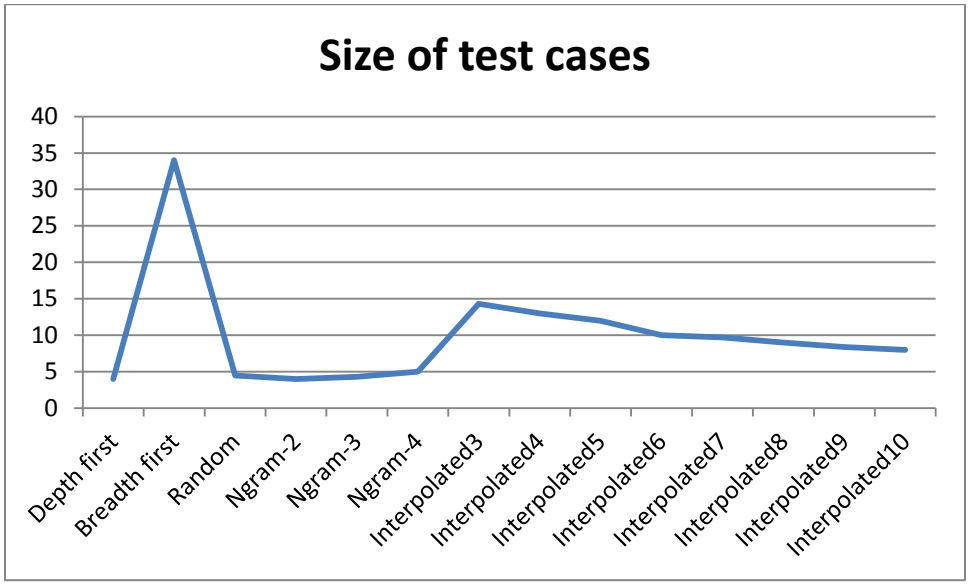

Figure4.3(Size of test cases Vs total number of test sequences in flexi-store)

Feasibility of Breadth first, Depth first and Random Approaches more than N-gram and interpolated N-gram approaches (INTERP).where N-gram decrease when increase interpolated N-gram value of n.

4.2.4

	LENTH
Depth first	25.34
Breadth first	3.38
Random	55.01
Ngram-2	47.73
Ngram-3	49.66
Ngram-4	45.48
Interpolated3	54.15
Interpolated4	53.41
Interpolated5	51.27
Interpolated6	55.95
Interpolated7	53.63
Interpolated8	55.01
Interpolated9	57.18
Interpolated10	56.5

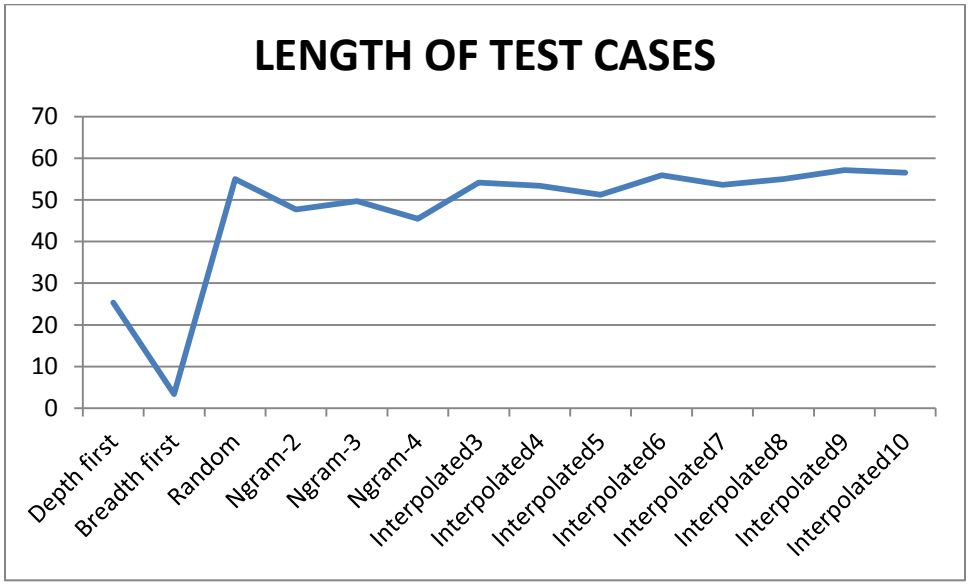

Figure 4.4(Length of test cases Vs total number of test sequences in flexi-store)

Feasibility of Breadth first, Depth first approaches less and Random Approaches increases and simultaneously interpolated N-gram approaches (Interpolated) will increase, it will increase when increase N-gram value of n. It show prediction increase when N-gram increase.

4.1.5

	Feasibility	Coverage (%)	Size of test cases	Length of test case
Depth first	2%	58	4	25.34
Breadth first	11%	87	34	3.38
Random	72%	65	4.5	55.01
Ngram-2	65%	88	4	47.73
Ngram-3	4%	73	4.3	49.66
Ngram-4	27%	88	5	45.48
Interpolated3	39%	79	14.33	54.15
Interpolated4	78%	89	13	53.41
Interpolated5	57%	8	12	51.27
Interpolated6	81%	89	10	55.95
Interpolated7	46%	8	9.7	53.63
Interpolated8	71%	88	9	55.01
Interpolated9	43%	8	8.39	57.18
Interpolated10	86%	87	8	56.5

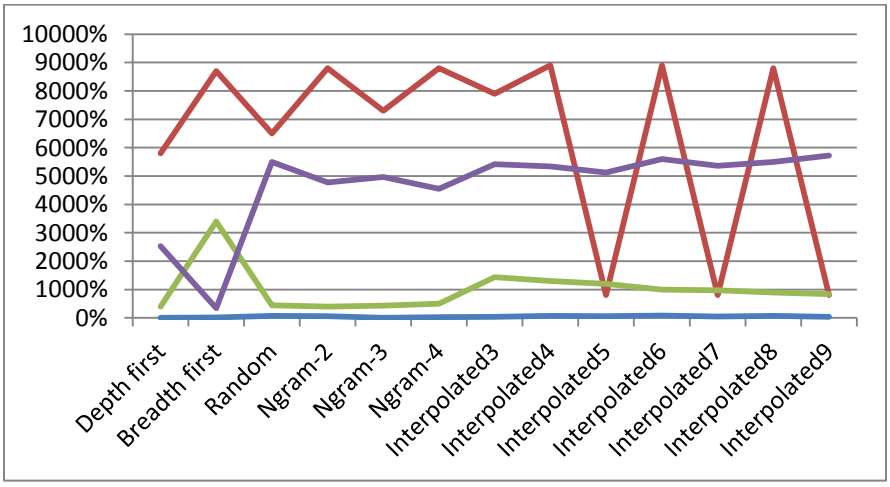

Figure 4.5 (Ratio between all test case Vs total test sequences in flexi-store)

Feasibility of Breadth first, Depth first and Random Approaches less than N-gram and interpolated N-gram approaches (Interpolated), it will increase when increase N-gram value of n. It show prediction increase when N-gram increase.

4.3Feasibility with all test cases in Cyclos software:

	Feasibility (%)	Coverage (%)	Size of test cases	Length of test case	
Depth first	53	36	5.61	6.22	
Breadth first	74	78	14	2.86	
Random	64	34	7.36	5	
Ngram-2	72	94	5.61	6	
Ngram-3	32	99	7.11	5.40	Cyclos
Ngram-4	73	94	7.22	5.60	
Interpolated3	34	56	5.73	7.02	
Interpolated4	67	94	5.79	6.92	
Interpolated5	77	57	5.63	6.90	
Interpolated6	80	94	5.71	6.94	

4.3.1

	Feasibility (%)
Depth first	53
Breadth first	74
Random	64
Ngram-2	72
Ngram-3	32
Ngram-4	73
Interpolated3	34
Interpolated4	67
Interpolated5	77
Interpolated6	80

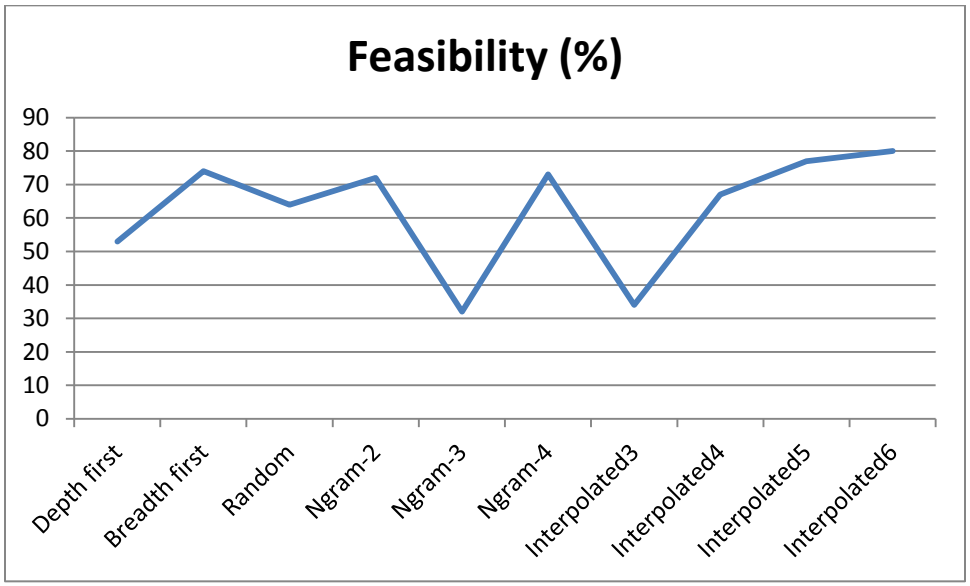

Figure 4.6(Feasibility Vs total test sequences in cyclos)

Feasibility of Breadth first, Depth first and Random Approaches less than N-gram and interpolated N-gram approaches (Interpolated), it will increase when increase N-gram value of n. It show prediction increase when N-gram increase.

4.3.2

	Coverage (%)
Depth first	36
Breadth first	78
Random	34
Ngram-2	94
Ngram-3	99
Ngram-4	94
Interpolated3	56
Interpolated4	94
Interpolated5	57
Interpolated6	94

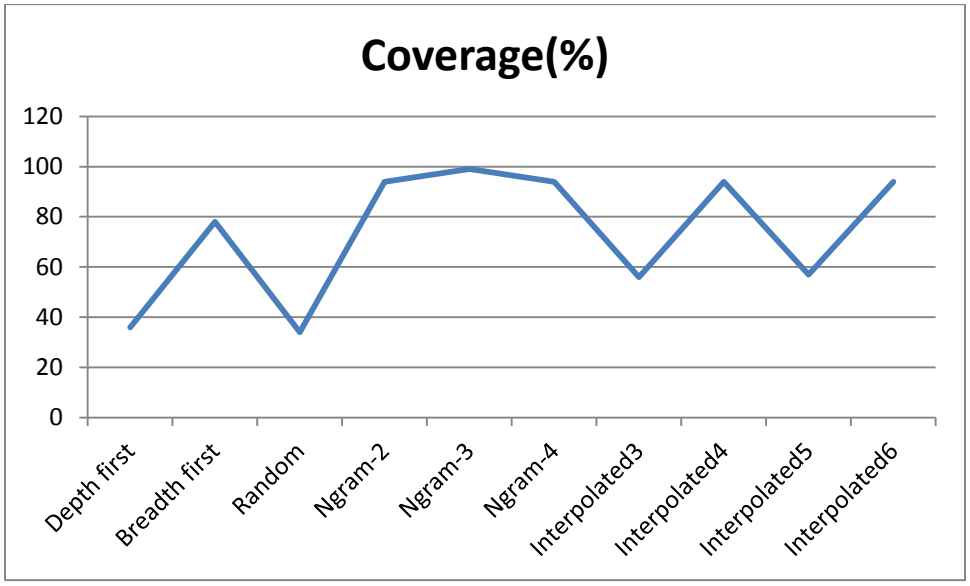

Figure 4.7 (Coverage vs total test sequences in cyclos

Feasibility of Breadth first, Depth first and Random Approaches increases step by step and to meet the N-gram and interpolated N-gram approaches (Interpolated).

4.3.3

	Size of test cases
Depth first	5.61
Breadth first	14
Random	7.36
Ngram-2	5.61
Ngram-3	7.11
Ngram-4	7.22
Interpolated3	5.73
Interpolated4	5.79
Interpolated5	5.63
Interpolated6	5.71

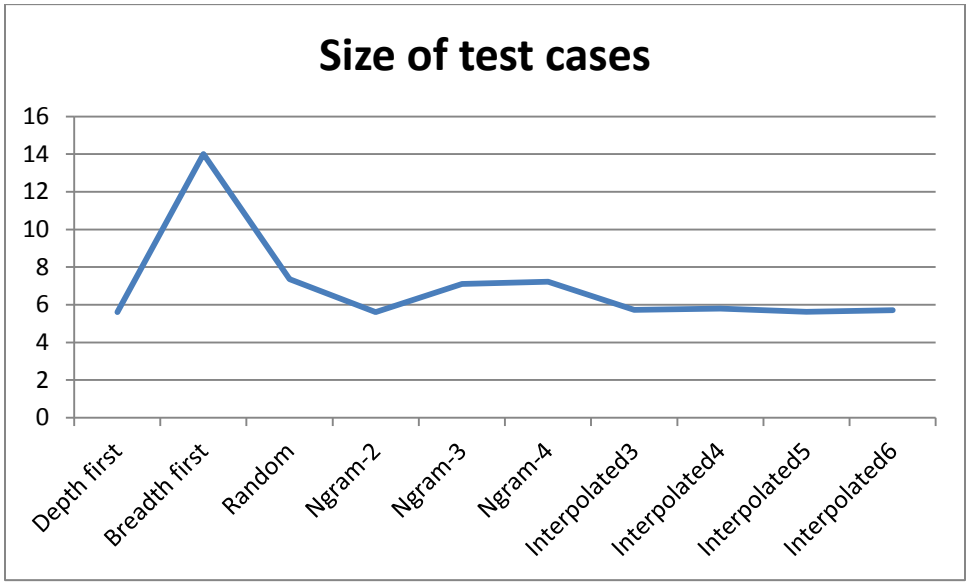

Figure 4.8(Size of test cases Vs total test sequences in cyclos

Feasibility of Breadth first, Depth first and Random Approaches more than N-gram and interpolated N-gram approaches (INTERP), it will increase when decrease N-gram value of n.

4.3.4

	Length of test case
Depth first	6.22
Breadth first	2.86
Random	5
Ngram-2	6
Ngram-3	5.40
Ngram-4	5.60
Interpolated3	7.02
Interpolated4	6.92
Interpolated5	6.90
Interpolated6	6.94

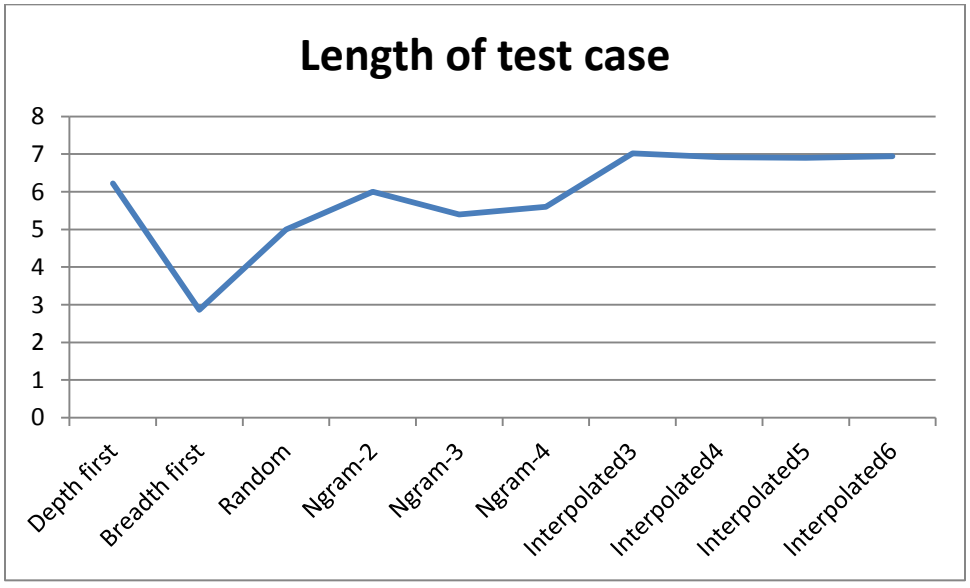

Figure 4.9(Length of test cases Vs total test sequences in cyclos)

Feasibility of Breadth first, Depth first and Random Approaches less than N-gram and interpolated N-gram approaches (INTERP), it will increase when increase N-gram value of n. It show prediction increase when N-gram increase.

4.3.5

	Feasibility (%)	Coverage (%)	Size of test cases	Lengthof test case
Depth first	53	36	5.61	6.22
Breadth first	74	78	14	2.86
Random	64	34	7.36	5
Ngram-2	72	94	5.61	6
Ngram-3	32	99	7.11	5.40
Ngram-4	73	94	7.22	5.60
Interpolated3	34	56	5.73	7.02
Interpolated4	67	94	5.79	6.92
Interpolated5	77	57	5.63	6.90
Interpolated6	80	94	5.71	6.94

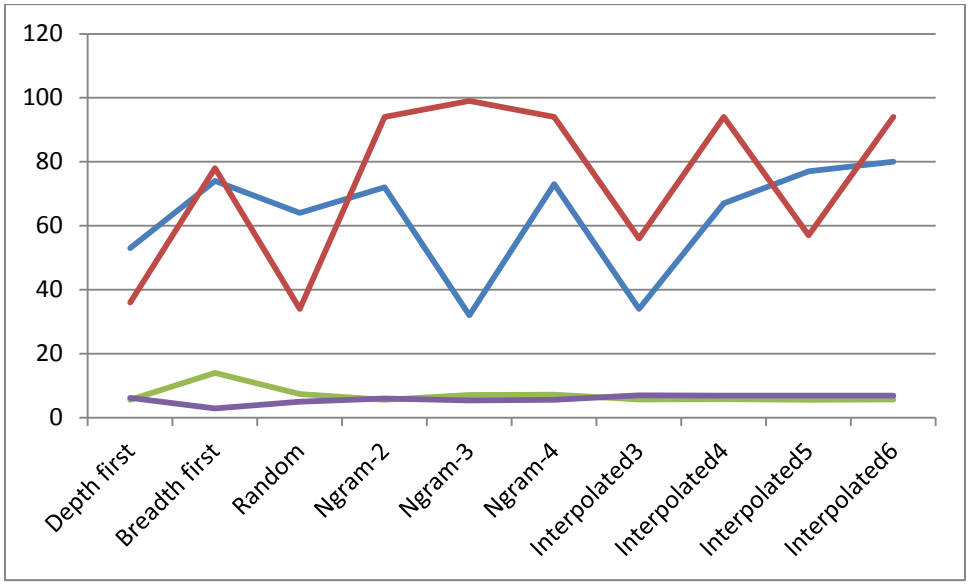

Figure 4.10(Ratio between all test cases Vs total test sequences in cyclos)

4.4 Feasibility with all test cases in the Organizer Software

	Feasibility	Coverage	Size of test cases	Length of test case
Depth first	22	90	20	11
Breadth first	23	98	51	3
Random	16	93	18	16
Ngram-2	13	99	16	16.12
Ngram-3	70	94	15	15.77
Ngram-4	80	98	15.63	16
Interpolated3	76	96	16.18	16.15
Interpolated4	30	99	15.61	17.18
Interpolated5	55	98	15.33	17.67
Interpolated6	81	34	15.27	18.28
Interpolated7	38	97	15.43	17
Interpolated8	66	99	15.19	18.58
Interpolated9	88	97	15.28	18
Interpolated10	40	99	15.1	18.60

4.4.1

	FEASIBILITY
Depth first	Feasibility
Breadth first	22
Random	23
Ngram-2	16
Ngram-3	13
Ngram-4	70
Interpolated3	80
Interpolated4	76
Interpolated5	30
Interpolated6	55
Interpolated7	81
Interpolated8	38
Interpolated9	66
Interpolated10	88

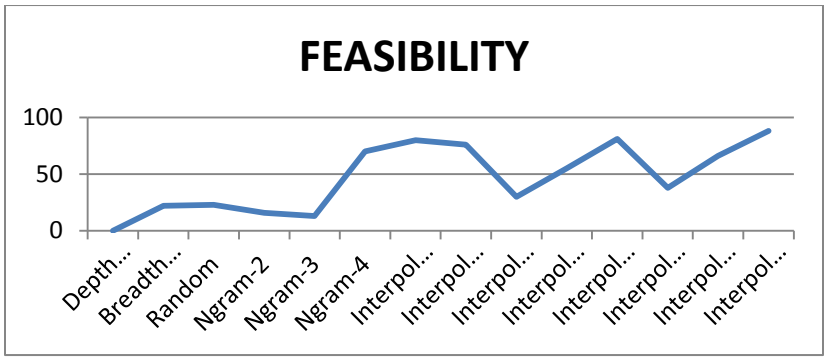

Figure 4.11(Feasibility Vs Total test sequences in organizer)

Feasibility of BFV, DFV and RAND Approaches less than N-gram and interpolated N-gram approaches (INTERP), it will decrease when increase N-gram value of n. It show prediction increase when N-gram increase.

4.4.2

	Coverage(%)
Depth first	90
Breadth first	98
Random	93
Ngram-2	99
Ngram-3	94
Ngram-4	98
Interpolated3	96
Interpolated4	99
Interpolated5	98
Interpolated6	34
Interpolated7	97
Interpolated8	99
Interpolated9	97
Interpolated10	99

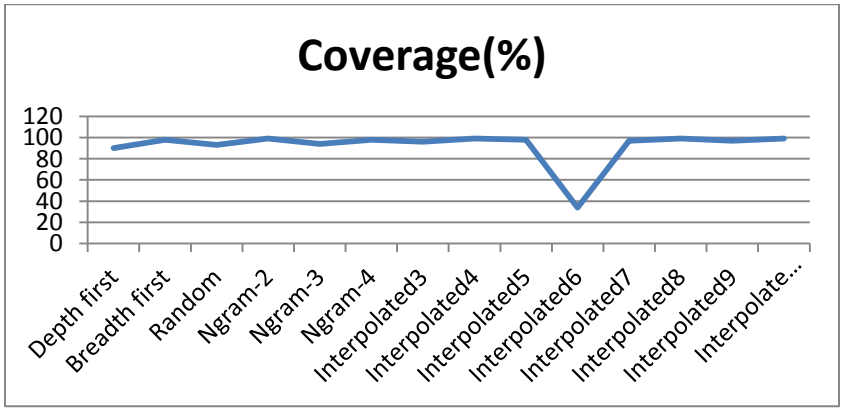

Figure 4.12(Coverage Vs total test sequences in organizer)

Here step by step feasibility of Breadth first, Depth first and random approaches increases when N-Gram value increase and simultaneously N-Gram approaches (Interpolated) increases.

4.4.3

	Size of test cases
Depth first	20
Breadth first	51
Random	18
Ngram-2	16
Ngram-3	15
Ngram-4	15.63
Interpolated3	16.18
Interpolated4	15.61
Interpolated5	15.33
Interpolated6	15.27
Interpolated7	15.43
Interpolated8	15.19
Interpolated9	15.28
Interpolated10	15.1

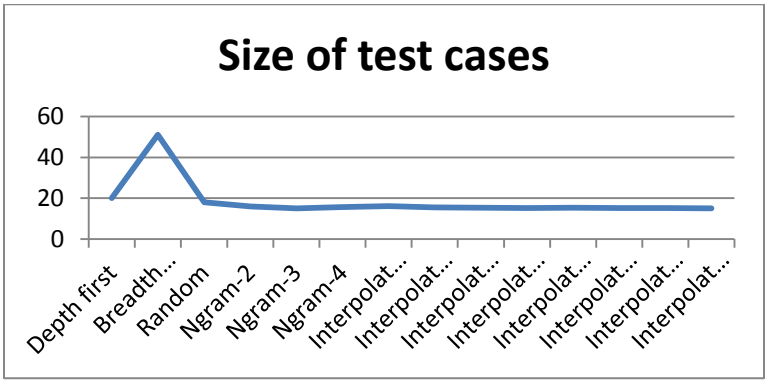

Figure 4.13(Size of test cases Vs total test sequences in organizer)

Feasibility of Breadth first, Depth first and Random Approaches more than N-gram and interpolated N-gram approaches (INTERP), it will increase when decrease N-gram value of n. It show prediction decrease when N-gram decrease.

4.4.4

	Length of test case
Depth first	11
Breadth first	3
Random	16
Ngram-2	16.12
Ngram-3	15.77
Ngram-4	16
Interpolated3	16.15
Interpolated4	17.18
Interpolated5	17.67
Interpolated6	18.28
Interpolated7	17
Interpolated8	18.58
Interpolated9	18
Interpolated10	18.60

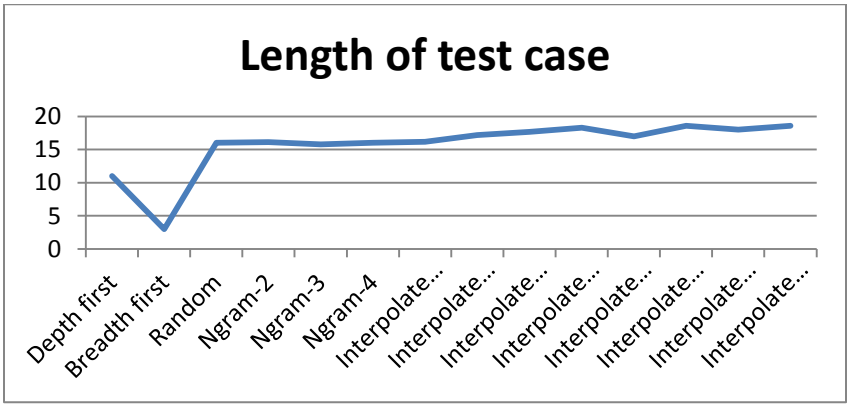

Figure 4.14(Length of test case Vs total test sequences in organizer)

Feasibility of Breadth first, Depth first and Random Approaches increases when N-gram and interpolated N-gram approaches (INTERP) will increases.

4.4.5

	Feasibility	Coverage	Size of test cases	Length of test case
Depth first	22	90	20	11
Breadth first	23	98	51	3
Random	16	93	18	16
Ngram-2	13	99	16	16.12
Ngram-3	70	94	15	15.77
Ngram-4	80	98	15.63	16
Interpolated3	76	96	16.18	16.15
Interpolated4	30	99	15.61	17.18
Interpolated5	55	98	15.33	17.67
Interpolated6	81	34	15.27	18.28
Interpolated7	38	97	15.43	17
Interpolated8	66	99	15.19	18.58
Interpolated9	88	97	15.28	18
Interpolated10	40	99	15.1	18.60

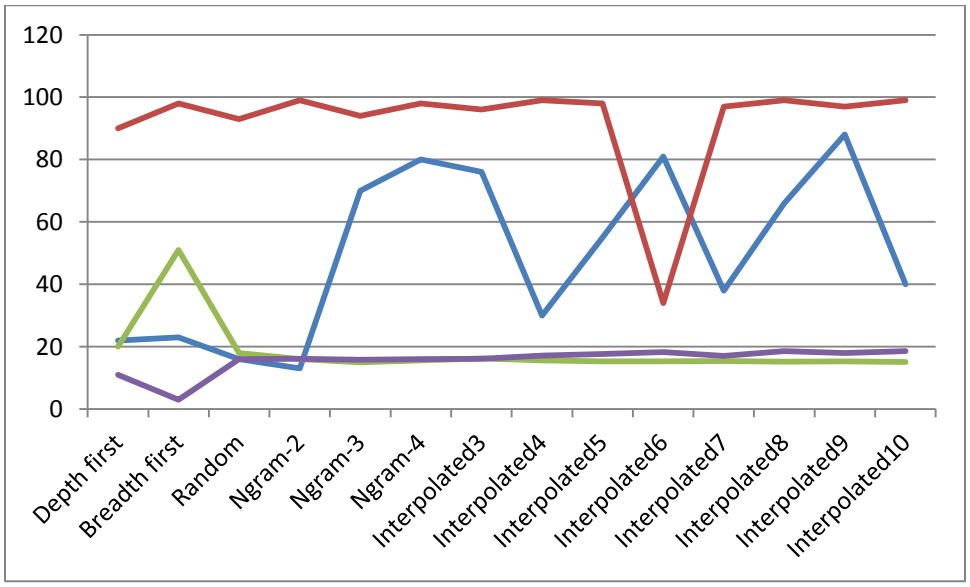

Figure 4.15(Ratio between total test cases and total test sequences in organizer)

4.5 Feasibility with all test cases in the Task Freak software

	Feasibility(%)	Coverage(%)	Size of test cases	Length of test case	
Depth first	72	91	7.40	15.2	
Breadth first	82	34	38	3.11	
Random	98	94	6	33.	
Ngram-2	98	100	7.38	39.68	
Ngram-3	50	96	7	39.64	
Ngram-4	92	100	7.73	34.15	
Interpolated3	88	99	7.6	39.32	TaskFreak
Interpolated4	99	56	7.39	40.05	
Interpolated5	90	98	7.28	38.31	
Interpolated6	80	75	7.2	40.	
Interpolated7	90	96	7.07	40.58	
Interpolated8	34	76	7.4	40.45	
Interpolated9	82	98	7.22	39	
Interpolated10	94	76	7.1	39	

4.5.1

	Feasibility (%)
Depth first	72
Breadth first	82
Random	98
Ngram-2	98
Ngram-3	50
Ngram-4	92
Interpolated3	88
Interpolated4	99
Interpolated5	90
Interpolated6	80
Interpolated7	90
Interpolated8	34
Interpolated9	82
Interpolated10	94

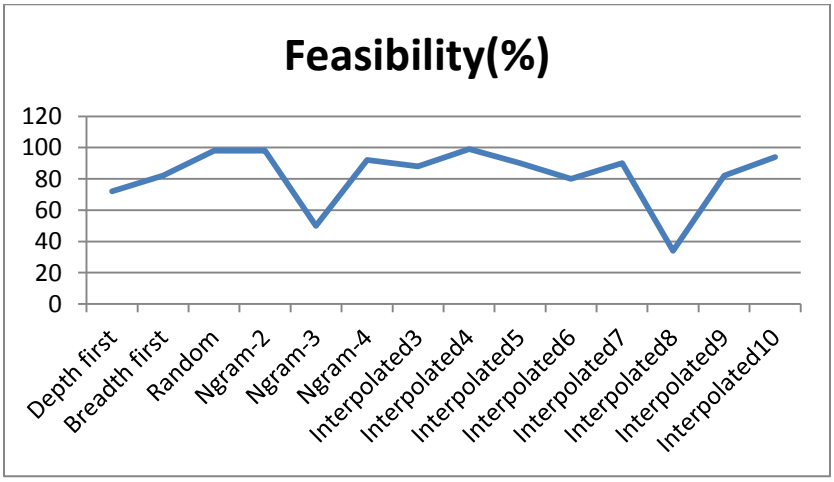

Figure 4.16(Feasibility Vs total test sequences in task freak)

Feasibility of Breadth first, Depth first increases and Random Approaches decreases when N-gram and interpolated N-gram approaches (INTERP) increases.

4.5.2

	Coverage(%)
Depth first	91
Breadth first	34
Random	94
Ngram-2	100
Ngram-3	96
Ngram-4	100
Interpolated3	99
Interpolated4	56
Interpolated5	98
Interpolated6	75
Interpolated7	96
Interpolated8	76
Interpolated9	98
Interpolated10	76

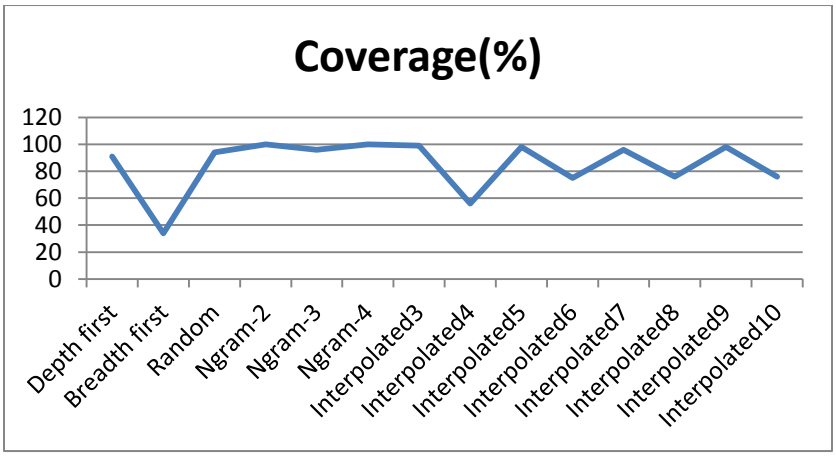

Figure 4.17(Coverage Vs total test sequences in task freak)

Here step by step feasibility of Breadth first, Depth first and Random approaches increases when N-Gram value increase and simultaneously N-Gram approaches (Interpolated) increases.

4.5.3

	Size of test cases
Depth first	7.40
Breadth first	38
Random	6
Ngram-2	7.38
Ngram-3	7
Ngram-4	7.73
Interpolated3	7.6
Interpolated4	7.39
Interpolated5	7.28
Interpolated6	7.2
Interpolated7	7.07
Interpolated8	7.4
Interpolated9	7.22
Interpolated10	7.1

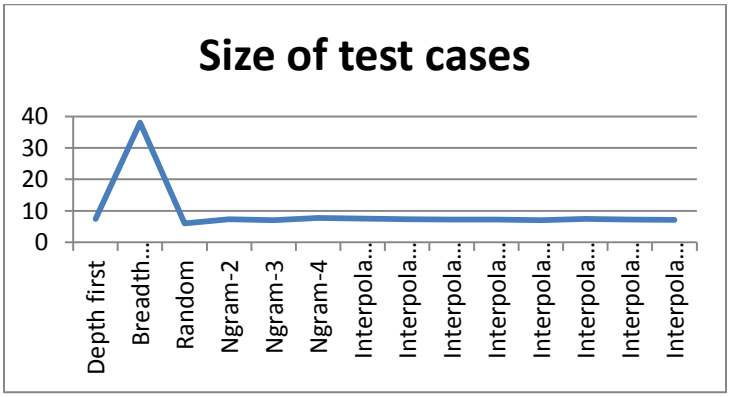

Figure 4.18(Size of test cases Vs total test sequences in task freak)

Feasibility of Breadth first, Depth first and Random Approaches more than N-gram and interpolated N-gram approaches (INTERP), it will increase when decrease N-gram value of n.

4.5.4

	Length of test case
DFV	15.2
Depth first	3.11
Breadth first	33.
Random	39.68
Ngram-2	39.64
Ngram-3	34.15
Ngram-4	39.32
Interpolated3	40.05
Interpolated4	38.31
Interpolated5	40.
Interpolated6	40.58
Interpolated7	40.45
Interpolated8	39
Interpolated9	39

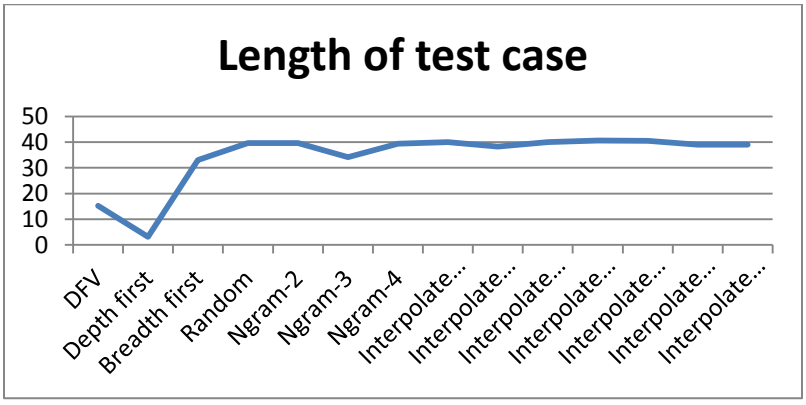

Figure 4.19(Length of test cases Vs total test sequences in task freak)

Feasibility of Breadth first, Depth first and Random Approaches less than N-gram and interpolated N-gram approaches (INTERP), it will decrease when increase N-gram value of n. It show prediction increase when N-gram increase.

4.5.5

	Feasibility(%)	Coverage(%)	Size of test cases	Length of test case
Depth first	72	91	7.40	15.2
Breadth first	82	34	38	3.11
Random	98	94	6	33.
Ngram-2	98	100	7.38	39.68
Ngram-3	50	96	7	39.64
Ngram-4	92	100	7.73	34.15
Interpolated3	88	99	7.6	39.32
Interpolated4	99	56	7.39	40.05
Interpolated5	90	98	7.28	38.31
Interpolated6	80	75	7.2	40.
Interpolated7	90	96	7.07	40.58
Interpolated8	34	76	7.4	40.45
Interpolated9	82	98	7.22	39
Interpolated10	94	76	7.1	39

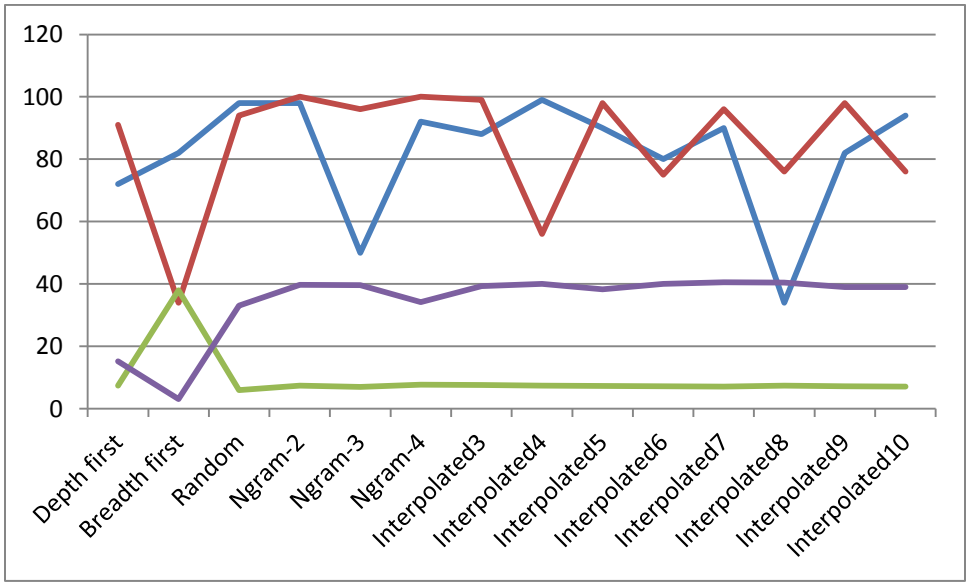

Figure 4.20(Ratio between total test case Vs total test sequences in organizer)

4.6 Feasibility with all test cases in the Hit List software

	Feasibility(%)	Coverage (%)	Size of test cases	Length of test case	
Depth first	20	70	6	11	
Breadth first	28	72	15	5.4	
Random	7	75	3.6	32	
Ngram-2	42	96	4	31.78	
Ngram-3	70	66	3	35	
Ngram-4	40	95	3.38	31	Hit List
Interpolated3	46	76	4.5	24	
Interpolated4	50	76	3.98	24.20	
Interpolated5	38	70	3.99	24.85	
Interpolated6	60	54	3.65	26.58	
Interpolated7	19	75	3.78	24.08	
Interpolated8	60	56	3.82	25.1	
Interpolated9	68	75	3.82	23.49	
Interpolated10	93	50	3.67	25	

4.6.1

	Feasibility(%)
Depth first	20
Breadth first	28
Random	7
Ngram-2	42
Ngram-3	70
Ngram-4	40
Interpolated3	46
Interpolated4	50
Interpolated5	38
Interpolated6	60
Interpolated7	19
Interpolated8	60
Interpolated9	68
Interpolated10	93

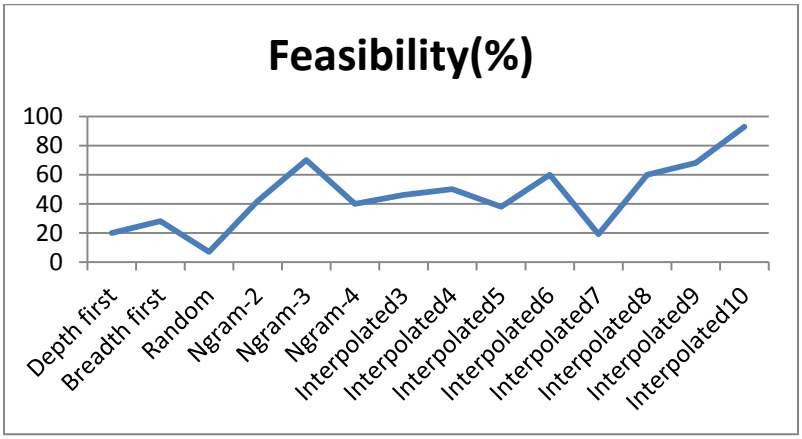

Figure 4.21(Feasibility Vs total test sequences in Hit list)

Feasibility of Breadth first, Depth first and Random Approaches less than N-gram and interpolated N-gram approaches (INTERP), it will increase when increase N-gram value of n. It show prediction increase when N-gram increase.

4.6.2

	Coverage(%)
Depth first	70
Breadth first	72
Random	75
Ngram-2	96
Ngram-3	66
Ngram-4	95
Interpolated3	76
Interpolated4	76
Interpolated5	70
Interpolated6	54
Interpolated7	75
Interpolated8	56
Interpolated9	75
Interpolated10	50

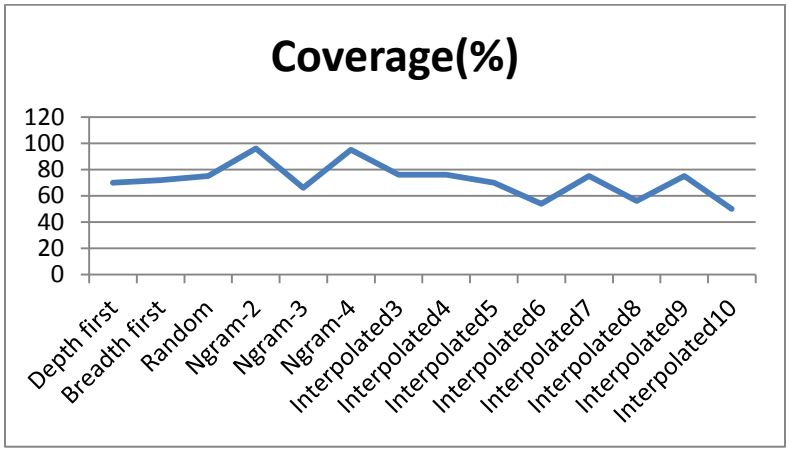

Figure 4.22(Coverage Vs total test sequences in Hit list)

Feasibility of Breadth first, Depth first and Random Approaches remain same as N-gram and interpolated N-gram approaches (INTERP).

4.6.3

	Size of test cases
Depth first	6
Breadth first	15
Random	3.6
Ngram-2	4
Ngram-3	3
Ngram-4	3.38
Interpolated3	4.5
Interpolated4	3.98
Interpolated5	3.99
Interpolated6	3.65
Interpolated7	3.78
Interpolated8	3.82
Interpolated9	3.82
Interpolated10	3.67

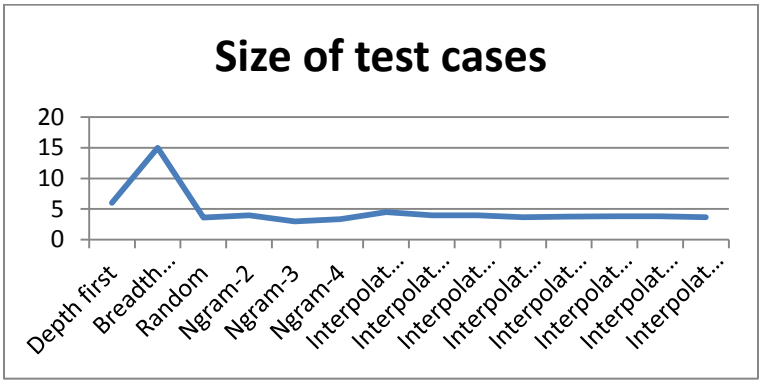

Figure 4.23(Size of test cases Vs total test sequences in Hit list)

Feasibility of Breadth first, Depth first and Random Approaches more than N-gram and interpolated N-gram approaches (INTERP), it will increase when decrease N-gram value of n. It show prediction increase when N-gram decrease.

4.6.4

	Length of test case
Depth first	11
Breadth first	5.4
Random	32
Ngram-2	31.78
Ngram-3	35
Ngram-4	31
Interpolated3	24
Interpolated4	24.20
Interpolated5	24.85
Interpolated6	26.58
Interpolated7	24.08
Interpolated8	25.1
Interpolated9	23.49
Interpolated10	25

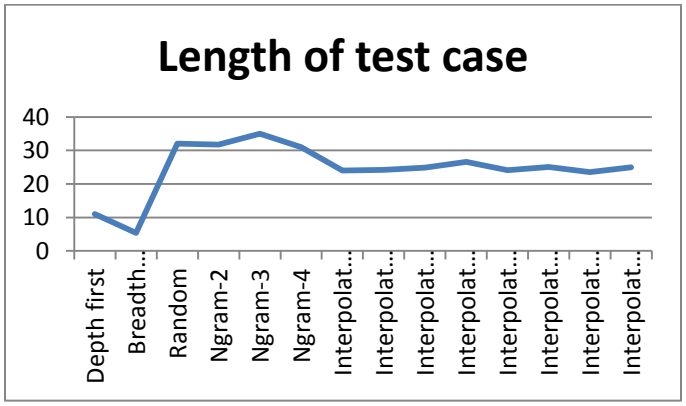

Figure 4.24(Length of test cases Vs total test sequences in Hit list)

Feasibility of Breadth first, Depth first and Random Approaches less than N-gram and interpolated N-gram approaches (INTERP), it will increase when increase N-gram value of n. It show prediction increase when N-gram increase.

4.6.5

	Feasibility (%)	Coverage (%)	Size of test cases	Lengthof test case
Depth first	20	70	6	11
Breadth first	28	72	15	5.4
Random	7	75	3.6	32
Ngram-2	42	96	4	31.78
Ngram-3	70	66	3	35
Ngram-4	40	95	3.38	31
Interpolated3	46	76	4.5	24
Interpolated4	50	76	3.98	24.20
Interpolated5	38	70	3.99	24.85
Interpolated6	60	54	3.65	26.58
Interpolated7	19	75	3.78	24.08
Interpolated8	60	56	3.82	25.1
Interpolated9	68	75	3.82	23.49
Interpolated10	93	50	3.67	25

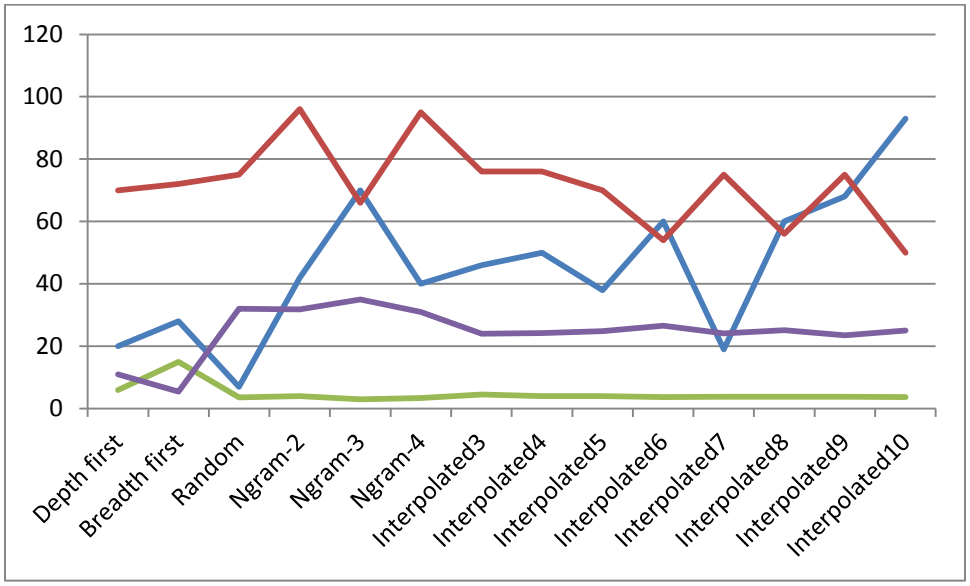

Figure 4.25(Ratio between total test cases Vs total test sequences in Hit list)

4.7 Comparison between Existing and Proposed Methodology

S.No.	Existing	Proposed
1.	Graph based algorithm have used for finding the path between test cases.	NLP algorithms use for finding the n-grams like naive bayes, which we have used.
2.	Graph based algorithm used LIFO and FIFO queue for implementing the test case sequences ,but itis not reduce of test cases	N-gram statistics find the deterministic path for test case with the help of n gram.
3.	The recursion used by randomized test case finding algorithm ,Depth first and breadth first algorithm with the help of LIFO and FIFO data structure.	Using the probabilistic approach and N-gram statistics.
4.	Depth first and breadth first search iterative check the source that's why increase the complexity	Avoiding the generation of infeasible solution by finding the deterministic path with the help of N-gram statistics

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

In this thesis we have working on statics of N-gram because we resemble these two problem one of prediction of word in sentence by using of previous words and second one model based testing in which predict the path of next test case by using previous test cases by n gram approaches.

We have check four metrics for analysis our results, these are coverage, feasibility, length of test cases, number of test cases in one test suite. We have compare with previous approach like depth first, Breadth first and random .N-gram approach given significance difference from previous approaches.

In future we can use the N gram approach to web application and verification of hardware because in both cases same problem as model based test cases , so we can generalize our model.

REFERENCES

[1] A. Pretschner, M.Utting (2012) "A taxonomy of model based testing approaches"

[2] Andrews, J.H, Briand, L.C.Labiche, Y, (2012) "Is mutation an appropriate tool for testing experiments? In: International Conference on Software Engineering." , pp. 402–411. ACM, New York, NY, USA

[3] Bernhard Rumpe , (2014) " Executable Modelling with UML- A vision or a nightmare".

[4] ChristelBaier, Joost–Pieter Katoen, (2010) " Invariant-based automatic testing of modern web applications. IEEE Transactions on Software Engineering", 38(1):35–53.

[5] Cyrilla Artho, Armin Biere, Martina Seide, (2913) "Model-Based Testing for verification Back-ends".

[6] Cristran Cadar, Patrice Godefroid, Sarfraz Khurshid, Corina S. Pasareanu,, Kaushik Sen, Willeam Visser, (2011) "Symbolic execution for Software Testing in Practice".

[7] Dias Neto, Vorobev, E., Lapschies, FZahlten, (2011) "Automated model-based testing with RT-Tester.", Tech. rep., Universit ̈t Bremen

[8] Dudekula Mohammad Rafi, Katem Reddy, Kiran Moses, Kai Petersen, (2012). "Benefits and limitations of Automated Software Testing: Literature Review & practitioner survey".

[9] El-Far, Whittaker, (2007) "Word sense disambiguation: A survey. ACM Computing Surveys", v. 41, n. 2, p. 1–69.

[10] Gervazi, Zowghi, (2011) **"**Testing Web Applications by Modelling with FSMs. Software and System Modelling", Vol 4, n. 3, pages 326–345.

[11] H.Samih, H. Lguen, R. Bogushk, (2014) "Deriving Usage Model Variants for Model-Based Testing".

[12] Julien Botella, Fabrice Bouquet, Jean Francois, Capuron, Franck Lebeau, Bruno Legard, Florence Schadle, (2013) "Model-Based Testing of Cryptographic components lessons learned from experience".

[13] Jurafsky, Martin, (2004) "Principles of model checking" . Cambridge, MA, USA: The MIT Press,. 975 p.

[14] KedianMu, ZhiJin, Ruqian Lu, Weiru Li, (2008)."Combining model-based and combinatorial testing for effective test case generation. In Process of the ACM International Symposium on Software Testing and Analysis (ISSTA"), pages 100–110.

[15] Mark Harman, Phil Mcminn, Muzammil Shahbaz, Shin Yoo (2013) "A comprehensive survey of Trends in oracle For Software Testing".

[16] Mohamed Mussa, Ferhat Khandek, (2012) "Towards a Model Based Approach for Integration Testing".

[17] Petra Brosch, U. Egly, Schastian Gabmeyer, Yerti Kappel, Martina Seisi, Hans Compits, Magdalena Widl, Maneel Wimmer, (2012) "Towards scenario-based testing of UML Diagrams"

[18] Sarma, Mall, (2005)"Using model checking to generate tests from specifications".

[19] Utting, Legear, (2006)"Formal methods in industry: achievements, problems", future. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 28, Shanghai, China. Proceedings... New York, NY, USA: ACM, 2006. P.761–768.

[20] YoavBergner, Stefan Droschler, Gerd Kortemeyer, Saif Rayyan, Daniel Seaton , (2012) "Collaborative filtering Analysis of Student Response Data: Machine- Learning Item Response Theory"

[21] Ackerman, A.F, (2014) "An Introduction to Software engineering",
Software Engineering Education and Training (CSEE&T), 2014 IEEE 27th Conference on
DOI: 10.1109/CSEET.2014.6816803.

[22] G Van Rossum, (1993) "An Introduction to python for UNIX/C programmers

[23] RS Pressman, (2005) "Software engineering: a practitioner's approach"

[24] V Ambriola, G Tortora, (1993) "An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering", Volume I

Websites

[25] http://en.wikipedia.org/wiki/Software_engineering

[26] http://en.wikipedia.org/wiki/Software_testing

[27] http://www.softwaretestinghelp.com/types-of-software-testing/

[28] http://www.cs.tut.fi/tapahtumat/testaus08/Olli-Pekka.pdf

[29] http://en.wikipedia.org/wiki/Model-based_testing

[30] http://www.tutorialspoint.com/python/