Effective Test Case Generation for Load Testing of Web Server using River Formation Dynamics

Thesis submitted in partial fulfillment of the requirements for the award of degree

of

Master of Engineering

in

Software Engineering

Submitted By

Gurleen Kaur Sandhu

(Roll No. 801431008)

Under the supervision of

Ms. Rupinderdeep Kaur

Lecturer, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

June 2016

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Effective Test Case Generation for Load Testing of Web Server using River Formation Dynamics*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Rupinderdeep Kaur* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Gurleen.

(Gurleen Kaur Sandhu)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Rupinderdeep K.

(Rupinderdeep Kaur)

Lecturer, CSED

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. S. Bhatia)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

"The most important thing is God's blessing and if you believe in God and you believe in yourself, you have nothing to worry about." At the outset, I thank the Almighty God for showering eternal blessings and benevolence on me for successful completion of my research work.

First of all I would like to thank my supervisor, **Ms. Rupinderdeep Kaur** whose continuous support and encouragement have kept me going. My supervisor is undoubtedly the person who has the strongest influence on this work. As an advisor, she taught me practices and skills that I will use in my academic career. She helped me to develop my own views and opinions about the research I undertook.

I am equally grateful to **Dr. Maninder Singh**, Head, Department of Computer Science and Engineering, and **Dr. Rupali Bhardwaj,** P.G. Co-ordinator, Department of Computer Science and Engineering, for their support and motivation that encouraged me for the dissertation work.

I also like to offer my sincere thanks to all faculty members, teaching and non-teaching staff of Department of Computer Science and Engineering (CSED) and staff of central library, Thapar University, Patiala for their assistance.

I extend my special thanks to my parents and friends for their constant encouragement during the entire course of my work.

(Gurleen Kaur Sandhu)

ABSTRACT

Load Testing is one of the important tasks while evaluating the performance of any system whether it is a software module or hardware component. The load testing is generally associated with the software systems in which the incoming traffic on the software module is pushed and then checked by which aspect and performance it is behaving. Using load testing, the overloading or over-traffic in the incoming packets can be tested. Web Server is one of the classical software systems which are tested under various scenarios and the load in terms of input variables. In any web application, there are number of input variables and parameters which are required to be uploaded or processed during acceptance by the web server. Using load testing, the random values or files can be processed in the web server on increasing order so that the threshold of tolerance can be checked. Using this approach, the load testing of web server can be implemented. A number of approaches and algorithms are devised and implemented so far by number of researchers; still nature inspired algorithms are getting fame due to their higher optimization criteria. In this research work, a nature inspired approach river formation dynamics is proposed and implemented for the test case generation. It is found in the implementation that this approach is performing effectively and providing better and refined test cases which can be processed by the web server.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Software Testing

Software testing is a testing technique which conducts an analysis to present stakeholders on the information for eminence of the artifact or overhaul under investigation. Software testing presents the objective, autonomous observation of the software to tolerate the production to value and realize the risks of software performance. Test techniques consist of the progression of executing a program or application among the meaning of pronouncement software bugs that can be errors or extra defects.

This progression involves the effecting of a software module or structure element to evaluate one or further properties of significance [1]. In general, these properties indicate the extent to which the component or system under test:

- Guidance on Design and Development
- Inputs with integrity and better performance
- Effective Execution Time
- Usable
- Execute in robust environment
- Achieving the trust

Figure 1.1 - Software Testing

Software testing is a mixture of some testing techniques as shown in Figure 1.1. The different testing approaches have been described in Table 1.1 as the Full-lifecycle Object-Oriented Testing technique (Bertolino A., 2007) [2].

Table 1.1 - Testing Approaches (Bertolino A., 2007) [2]

FLOOT Technique	**Description**
Black-box testing	Verification without actual back-end manipulation and analysis in the code
Boundary-value testing	Testing of unusual or counterpart
Class testing	Classes and Instances under investigation
Class-integration testing	The format of classes and objects in investigation and deep analysis
Code review	The actual source code is analyzed
Component testing	Each and every component and module in analysis
Coverage testing	Certification and deep investigation of line based code
Design review	Technical review of the design and associated components
Inheritance-regression testing	Super class and bases classes with inheritance
Integration testing	Assorted components and integration
Method testing	Member functions and analysis
Model review	Review of deep model
Path testing	Logic paths
Prototype review	Prototype based review and analysis
Prove it with code	Code and associated functions
Regression testing	Earlier tested behaviors and its impact on the upcoming modules

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can

also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects) [3].

The process refers to the execution of system element for the evaluation of one or more properties of concentration. “In broad, these properties point toward the point to which the module or classification under analysis meets the requirements that guided its design and development, responds correctly to all kinds of inputs, performs its functions surrounded by an tolerable time, is suitably usable, can be installed and run in its intended environments, and achieves the common result as shown in Figure 1.2.

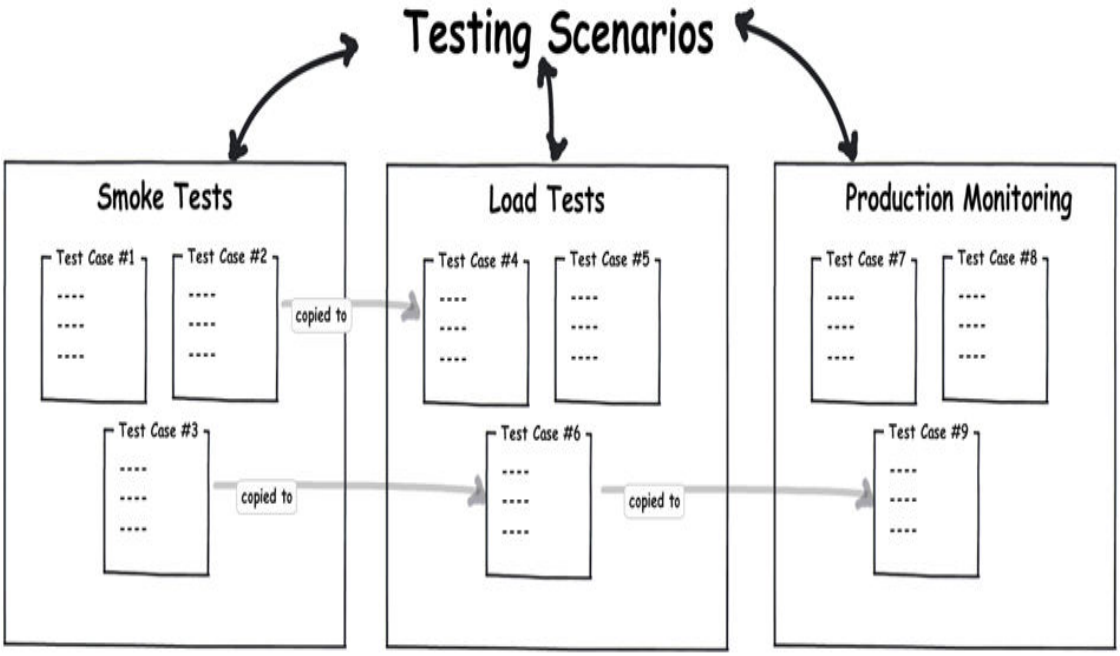


Figure 1.2 - Testing Scenarios

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects).” The job of testing is an iterative process as

when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones [4].

Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors. Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted as shown in Figure 1.3.

For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an Agile approach, requirements, programming, and testing are often done concurrently [5].

Figure 1.3 - Assorted Testing Strategies [6]

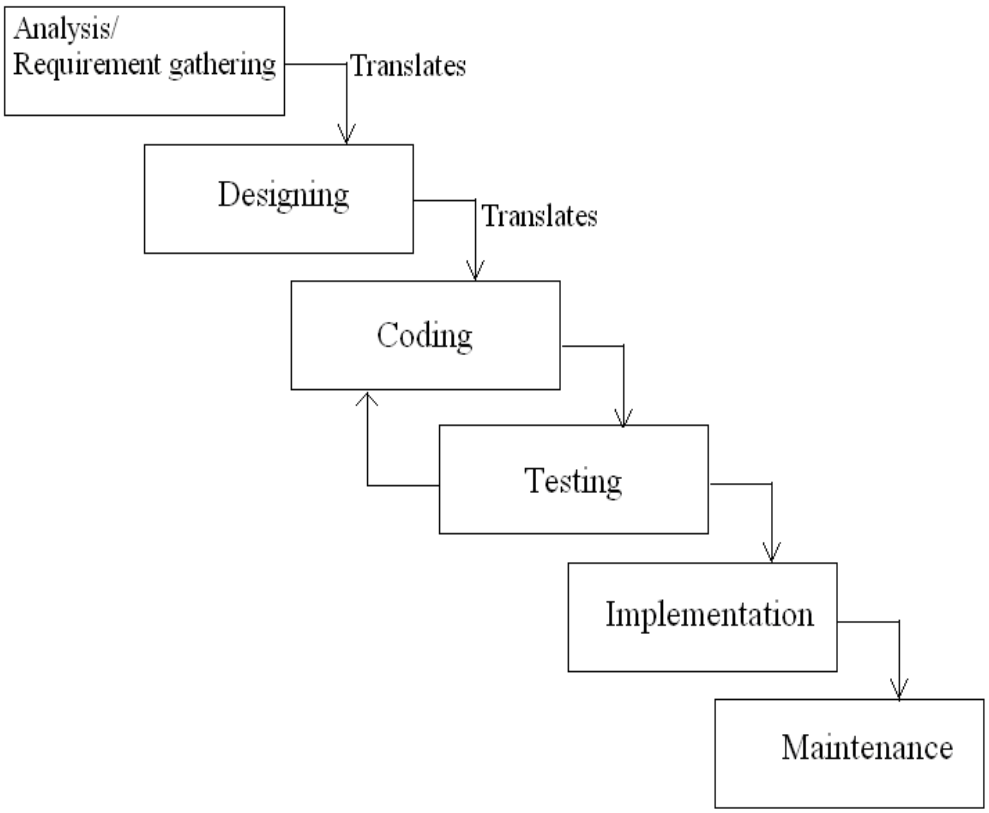


Figure 1.4 - Software Testing Life Cycle [3]

Figure 1.4 depicts the classical and generic approach of software testing life cycle [3]. There are number of approaches and phases involved in the overall process. The various phases, activities and other related parameters of the software testing life cycle are stated as shown in Table 1.2 [4].

Interestingly, no matter how well-defined a Software Testing Life Cycle you have in your project or organization, there are chances that you will invariably witness the following widely-popular cycle are testing and cursing.

Table 1.2 - Phases, Activities and related Parameters of STLC [4]

Phase	**Activity**	**Deliverables**	**Necessity**
Requirements/ Design Review	Review the software requirements/ design	• Review Defect Reports	Curiosity
Test Planning	Plan for tests, once you have gathered a general idea of what needs to be tested.	• Test Plan • Test Estimation • Test Schedule	Farsightedness
Test Designing	Design/ detail your tests on the basis of detailed requirements/design of the software.	• Test Cases / Test Scripts/Test Data • Requirements Traceability Matrix	Creativity
Test Environment Setup	Setup the test environment (server/ client/ network, *etc.*) with the goal to replicate the end-users environment.	• Test Environment	Rich company
Test Execution	Execute your Test Cases/ Scripts in the Test Environment to check if they pass.	• Test Results (Incremental) • Defect Reports	Patience
Test Reporting	Prepare various reports for your stakeholders.	• Test Results (Final) • Test/ Defect Metrics • Test Closure Report • Who Worked Late & on Weekends (WWLW) Report	Diplomacy

1.2 Software Testing Life Cycle

Following is the list of phases associated with Software Testing Life Cycle:

1. Requirements phase
2. Planning Phase
3. Analysis phase
4. Design Phase
5. Implementation Phase
6. Execution Phase
7. Conclusion Phase
8. Closure Phase

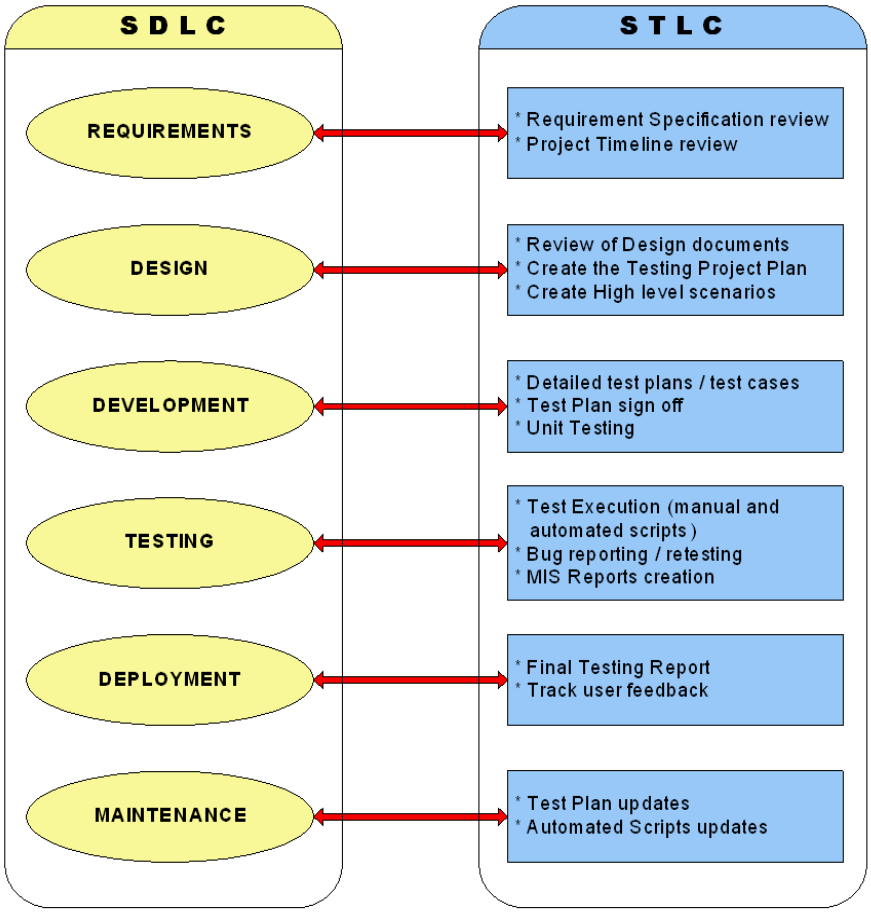


Figure 1.5 - SDLC and STLC

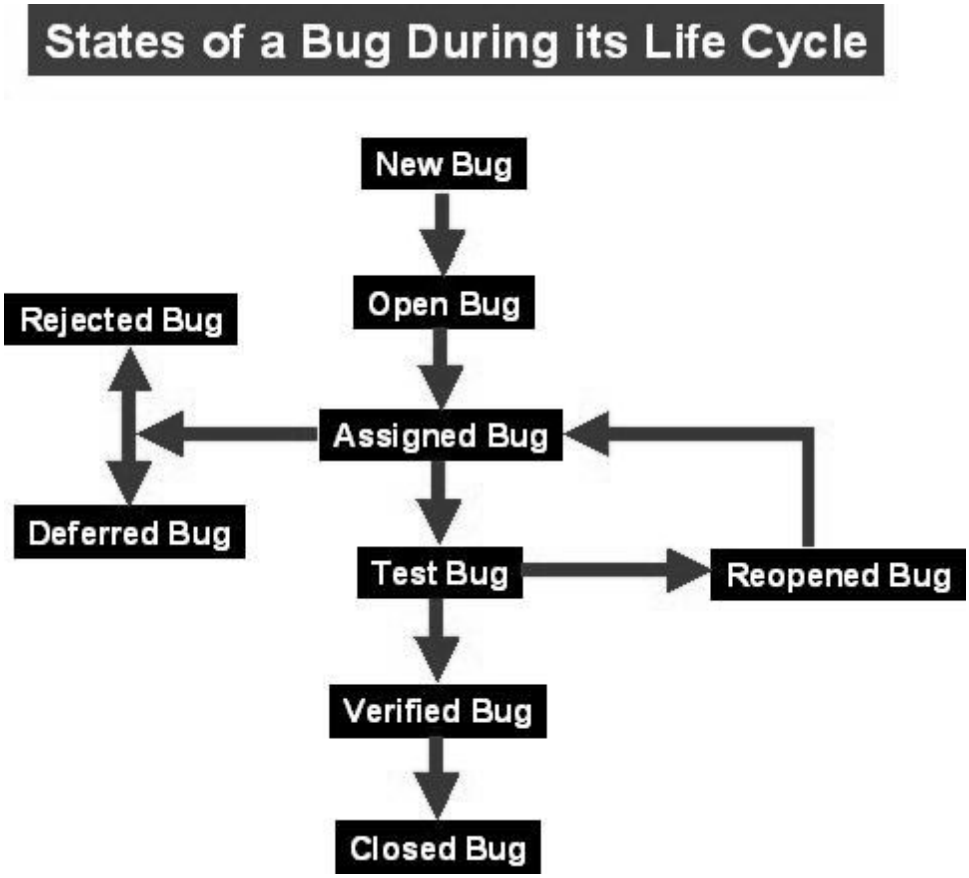


Figure 1.6 - Software Testing Life Cycle and Bug Management

The Figure 1.5 shows the different phases of STLC in correlation to SDLC and Figure 1.6 shows the bug management in STLC.

Software Testing Life Cycle (STLC) refers to the process of testing that is executed in planned and systematic manner. In this process, assorted activities are performed to improve the overall quality of product. Here a number of objects and activities are involved [7, 8].

Following is the list of steps which are involved and associated with the STLC. Every step is having the unique deliverable:

1. Requirement Analysis
2. Test Planning

3. Test Case Development
4. Environment Setup
5. Test Execution
6. Test Cycle Closure

More specifically, the further step is relying on the previous step or it can be said that the next step is not started till the previous step is not finished. This is possible in very ideal point, but very practically not always true.

1.2.1 Requirement Analysis

Requirement Analysis is the very first step in Software Testing Life Cycle (STLC). In this step Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements. "If any conflict, missing or not understood any requirement, then QA team follow up with the various stakeholders like Business Analyst, System Architecture, Client, Technical Manager/Lead *etc.* to better understand the detail knowledge of requirement.

From very first step, QA is involved in the STLC which helps to prevent the introducing defects into Software under test. The requirements can be either Functional or Non-Functional like Performance, Security testing. Also requirement and automation feasibility of the project can be done in this stage (if applicable).

1.2.2 Test Planning

Test Planning is most important phase of software testing life cycle where all testing strategy is defined. This phase also called as Test Strategy phase. In this phase, typically test manager (or test lead based on company to company) involved to determine the effort and cost estimates for entire project. This phase will be kicked off once the requirement gathering phase is completed & based on the requirement analysis, start preparing the test plan. The result of test planning phase will be test plan or test strategy and testing effort estimation documents. Once test planning phase is completed the QA team can start with test cases development activity.

1.2.3 Test Case Development

The test case development activity is started once the test planning activity is finished. This is the phase of STLC where testing team write down the detailed test cases. Along with test cases testing team also prepare the test data if any required for testing. Once the test cases are ready then these test cases are reviewed by peer members or QA lead.

Also the Requirement Traceability Matrix (RTM) is prepared. The Requirement Traceability Matrix is an industry-accepted format for tracking requirements where each test case is mapped with the requirement. Using this RTM we can track backward and forward traceability.

1.2.4 Test Environment Setup

Setting up the test environment is vital part of the STLC. Basically test environment decides on which conditions software is tested. This is independent activity and can be started parallel with Test Case Development. In process of setting up testing environment test team is not involved in it. Based on company to company may be developer or customer creates the testing environment. Mean while testing team should prepare the smoke test cases to check the readiness of the test environment setup.

1.2.5 Test Execution

Once the preparation of Test Case Development and Test Environment setup is completed then test execution phase can be kicked off. In this phase testing team start executing test cases based on prepared test planning and prepared test cases in the prior step.

Once the test case is passed then same can be marked as Passed. If any test case is failed then corresponding defect can be reported to developer team *via.* bug tracking system and bug can be linked for corresponding test case for further analysis. Ideally every failed test case should be associated with at least single bug. Using this linking

we can get the failed test case with bug associated with it. Once the bug fixed by development team then same test case can be executed based on your test planning.

If any of the test cases are blocked due to any defect then such test cases can be marked as Blocked, so report can be generated based on how many test cases passed, failed, blocked or did not run *etc.* Once the defects are fixed, same Failed or Blocked test cases can be executed again to retest the functionality.

1.2.6 Closure of Test

It refers to the calling out of testing team so that the overall performance and bugs report can be fetched out and further predictions can be done.

CHAPTER 2
LITERATURE REVIEW

A number of research papers, articles, conference papers and manuscripts are investigation for the literature review. It is found from the literature review that nature inspire algorithms are giving better results and effectiveness whenever we want optimized results.

2.1 Excerpts from the Research Papers

In order to plan and support the research work, a number of research papers have to be analyzed. Following are the excerpts from the different research work performed by number of academicians and researchers:

Kuhn (2004) *et. al.* [1] Exhaustive testing of computer software is intractable, but empirical studies of software failures suggest that testing can in some cases be effectively exhaustive. We show that software failures in a variety of domains were caused by combinations of relatively few conditions. These results have important implications for testing. If all faults in a system can be triggered by a combination of n or fewer parameters, then testing all n-tuples of parameters is effectively equivalent to exhaustive testing, if software behavior is not dependent on complex event sequences and variables have a small set of discrete values.

Bertolino (2007) *et. al.* [2] The authors in this work present the assorted testing approaches as the Full-lifecycle Object-Oriented Testing technique. The details on different approaches with their relative advantages and scope are underlined with effectiveness.

Tracey *et. al.* (1998) [3] One of the major costs in a software project is the construction of test-data. This paper outlines a generalized test-case data generation framework based on optimization techniques. The framework can incorporate a

number of testing criteria, for both functional and non-functional properties. Application of the optimization framework to testing specification failures and exception conditions is illustrated. The results of a number of small case studies are presented and show the efficiency and effectiveness of this dynamic optimisation-based approach to generating test-data.

Rayadurgam *et. al.* (2001) [4] In this research work a number of aspects are covered and addressed for the analysis of complexity with the web based applications. The work identifies the test cases and generated bugs based on assorted inputs which can be dynamically updated so that the overall effectiveness and consistency can be investigated.

Ali *et. al.* (2010) [5] In the research work done by the authors, the work develops effective and integrity based methods which makes use of user data for the building of models and related performance aspects. The user data set is captured and deeply investigated so that the further predictions can be done. In this research work, the web based testing automation is done so that the anytime anywhere availability of the testing environment can be set and implemented without any platform dependence.

Perrouin *et. al.* (2010) [6] Software Product Lines (SPL) is difficult to validate due to combinatorics induced by variability across their features. This leads to combinatorial explosion of the number of derivable products. Exhaustive testing in such a large space of products is infeasible. One possible option is to test SPLs by generating test cases that cover all possible T feature interactions (T-wise). T-wise dramatically reduces the number of test products while ensuring reasonable SPL coverage. However, automatic generation of test cases satisfying T-wise using SAT solvers raises two issues. The encoding of SPL models and T-wise criteria into a set of formulas acceptable by the solver and their satisfaction which fails when processed all-at-once. We propose a scalable toolset using Alloy to automatically generate test cases satisfying T-wise from SPL models. We define strategies to split T-wise combinations into solvable subsets. This work design and compute metrics to evaluate strategies on aspect OPTIMA, a concrete transactional SPL.

Rasal *et. al.* (2015) [7] Web services applications are built by the integration of many loosely coupled and reusable services using open standards. Testing Web service is important in detecting faults and assessing quality attributes. A difficulty in testing Web services applications is the unavailability of the source code for both the application builder and the broker. This paper propose a solution to this problem by providing a formal, specification-based approach for automatically generating test cases for web services based on the WSDL input messages parts XML Schema datatypes. Examples of using this approach are then given in order to give evidence of its usefulness. The role of the application builders and the brokers in using this approach to test Web services is also described.

Yan *et. al.* (2015) [8] Modified condition/decision coverage is a structural coverage criterion requiring that each condition within a decision is shown by execution to independently and correctly affect the outcome of the decision. This criterion was developed to help meet the need for extensive testing of complex boolean expressions in safety-critical applications. The paper describes the modified condition/decision coverage criterion, its properties and areas for further work.

Farrukh Shahzad *et. al.* [9] proposed the process control board based implementation so that hardware as well as software based testing of servers against malware can be avoided. In this proposed approach, the device based analysis and predictions can be done.

K. Allix *et. al.* [10] analyzes a set of applications having malware for deep investigation. In this approach, the authors present an effective model for packets evaluation and analytics.

Shahid Alam *et. al.* [11] In this work, the authors suggests another system called MARD which is meant for the Metamorphic Malware based Analysis and Real Time based Analysis for security secure the end nodes which are regularly accessed. Using

this approach and model, the overall testing of the system is done and predictions are made so that the performance can be improved.

M.S Alam *et. al.* [12] applied machine learning based approach using classifier on Android dataset having 48919 records. The key objective was to find out and extract the Random Forest and evaluate the attacks predictions.

M. Zubair Rafique *et. al.* [13] presents the FIRMA that is a device which give a better pool of structure for changing the malware attacks. It avoids and repels back the attacks generated by malware and improves the overall performance of system. If the system is working against the malware and related aspects, the load testing is successful.

N. Idika *et. al.* [14] presents the load testing of the systems against assorted datasets of malware so that the performance can be enhanced despite of number of attacks and enormous traffic. In total, this work examined 45 different datasets detection techniques and compared them against one another in order to make decision making process. This decision making process was provide to help the users for right selection of technique in order to develop a secure app or system.

Kriti *et. al.* [15] has suggested an approach called as Bi-features approach. This approach is used for two purposes, first to increase the precision of existing static mono-features analysis and secondly, to withdraw the drawbacks of the existing techniques. This paper provides three different categories of malicious software, which are based on their payload, enabling vulnerability and propagation mechanisms against the load testing aspects.

Imtithal *et. al.* [16] explained two classes of malicious software *i.e.*, ordinary malware and network –based malware. In this paper, various detection technologies are mentioned to tackle malware. Host-based intrusion detection is one of them. The other technique mentioned is network-based intrusion detection. A combination of the two techniques *i.e.,* host-based and network-based intrusion called hybrid is also

presented. One more technique explained in this paper is application-protocol based intrusion detection system (APIDS).

Jyoti Landage *et. al.* [17] explain the need to research and develop new techniques in order to detect known as well as unknown malwares as the previously known techniques are not capable of detecting unknown malwares. The new proposed methods for detecting unknown traffic are data-mining and machine learning methods. Using this approach, the overall performance can be enhanced with the use of different test sets and cases.

Martin Overton *et. al.* [18] provided a completely different approach to detect the abnormal traffic with the use of Snort. This work gives the detailed investigation for the use IDS system. They also gave assurance regarding efficient and fast detection and removal of abnormal and unwanted traffic from system which improves the overall performance of the web server.

Das *et. al.* [19] proposed hardware architecture GuardOL. This architecture is used to perform online malware detection. In order to capture highly malicious software, frequency-centric model for feature construction using system call patterns of known malware and benign samples is developed and later a machine learning approach is developed in FPGA to train classifier using these features.

J. Rabek *et. al.* [20] developed a host based technique called as DOME used for detecting the traffic and unwanted footprints in executable codes. This technique claims to be practically successful, simple to implement and applicable to all the real world applications. This mentioned technique is very effective in case of obfuscated code and dynamically generated code. DOME identifies the locations of system calls within the software executable using static analysis. At runtime the executable are monitored to verify whether the location identified using static analysis is same from where the observed system call is made.

S. Shah *et. al.* [21] proposed a technique which is used to identify files which are not valid for the test cases. The selection of the file is done by extracting the most relevant features using Fisher Score. The extracted features are then applied as input into the neural networks, which are actually the fields of Portable Executable (PE) structure.

The following Table 2.1 displays the significant points of the literature survey. It shows the various authors, their work and the year of publication of their paper.

Table 2.1 – Key Points of the Literature Review

Author	**Year of Publication**	**Work**
McMinn *et. al.*	2011	Search Based Software Testing using Genetic Algorithm
Arcuri *et. al.*	2011	Developed own tool and approach for search based software engineering
Harman *et. al.*	2011	Implemented nature inspired algorithm for fault tolerant systems
Fraser *et. al.*	2012	Real World projects evaluated using heuristic approaches
Anand *et. al.*	2013	Implementation of auto test case generation
Galeotti *et. al.*	2013	Use Genetic Algorithm
Li *et. al.*	2004	Implementation of ant colony optimization
Srivastava *et. al.*	2010	State Transition Testing in Software

		Engineering
Farrukh Shahzad *et. al.*	2013	Device Based Implementation
K. Allix *et. al.*	2014	Network Analytics
Shahid Alam *et. al.*	2014	Real Time Analytics and Predictions
M.S.Alam *et. al.*	2013	Mining and Predictions
M. Zubair Rafique *et. al.*	2013	Model Based Approach
N. Idika *et. al.*	2007	A novel classification scheme
Kriti *et. al.*	2013	Bi-feature approach
Imtithal *et. al.*	2013	Host-based and network-based intrusion detection, APIDS.
Jyoti Landage *et. al.*	2013	Data- mining and machine learning methods.
Martin Overton *et. al.*	2005	Packets Analysis for Load Testing and Abnormal Traffic Investigation
Sanjeev Das *et. al.*	2016	GuardOL
J. Rabek *et. al.*	2003	DOME
S. Shah *et. al.*	2013	Fisher Score

CHAPTER 3

PROBLEM STATEMENT

As the sphere of Software Testing and Test Case Generation is much expanded, there is lots of room for research to be carried out by the scholars and practitioners.

3.1 Research Gap

- The classical approach for generation of random test cases is not effective because of the biasing factors and non-tolerance.
- In existing work, the abnormal or invalid inputs can be taken as the test cases which are generally not required.
- There is scope of local optima in the existing work that can be improved using meta-heuristic approaches.

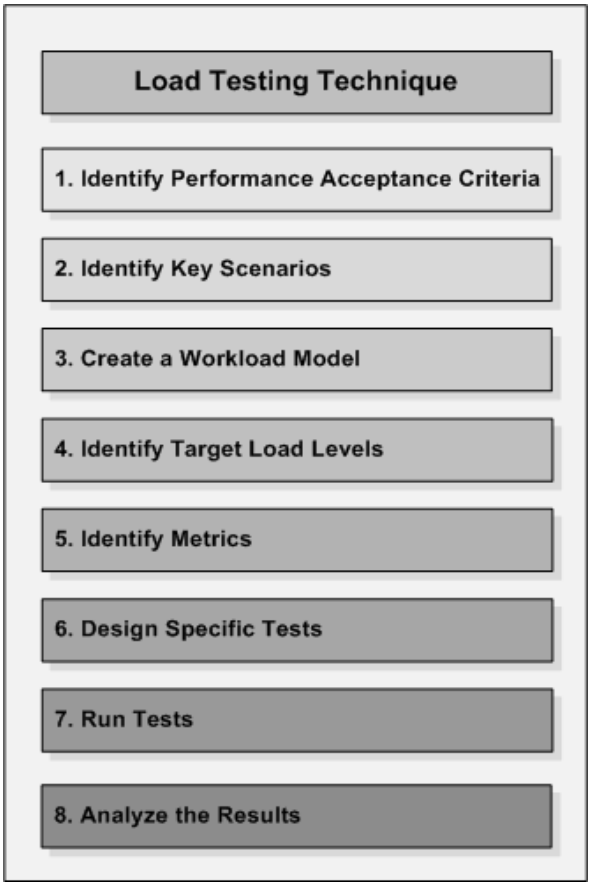


Figure 3.1 - Load Testing Approach and Flow

The Load testing approach to be followed is shown in Figure 3.1 as a sequence of different tasks. The traditional approach of software testing is shown in Figure 3.2. Hence, load testing approach and traditional software testing can be compared.

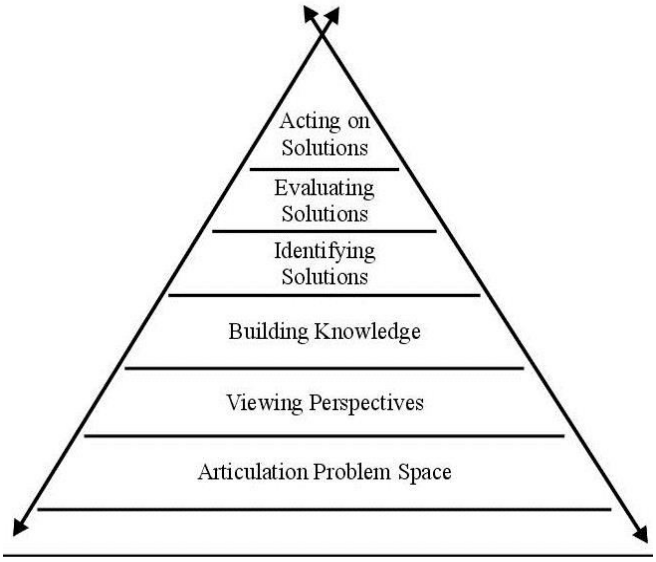


Figure 3.2 - Traditional Approach of Software Testing

3.2 Problem Statement

The test case generation is one of the widely used and important tasks for evaluation of performance of any software. The existing work can be improved for higher accuracy and better optimization factors with the valid set of test cases.

3.3 Objectives

The foremost goals of this research are as follows:

1. To evaluate the automated test case generation techniques.
2. To perform the pragmatic comparative analysis towards test case generation techniques.
3. Propose a new algorithm using River Formation Dynamics for Test Case Generation.
4. The test cases (files) for the load testing shall be generated using RFD algorithm.
5. Comparison between classical and proposed approach on multiple parameters like:

- Execution Time
- Complexity
- Cost Factor
- Overall Performance

The goals have also been shown in diagrammatical form in Figure 3.3. This figure shows the flow of proposed work and its sequence.

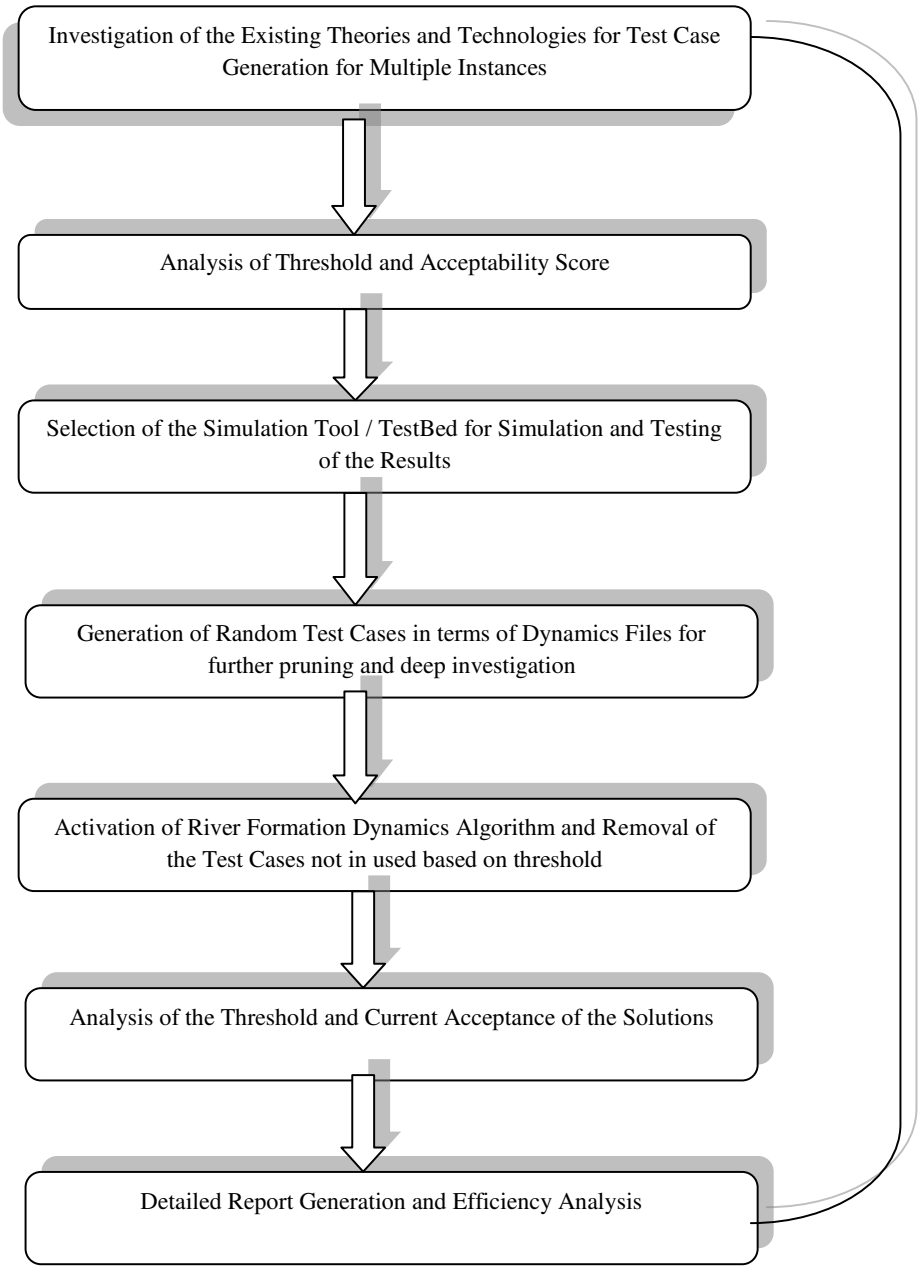


Figure 3.3 - Proposed Flow Diagram and Sequence

CHAPTER 4
IMPLEMENTATION OF PROPOSED WORK

Using load testing, the random values or files can be processed in the web server on increasing order so that the threshold of tolerance can be checked. Using this approach, the load testing of web server can be implemented. A number of approaches and algorithms are devised and implemented so far by number of researchers; still nature inspired algorithms are getting fame due to their higher optimization criteria. In this work, a nature inspired approach river formation dynamics is proposed and implemented for the test case generation. It is found in the implementation that this approach is performing effectively and providing better and refined test cases which can be processed by the web server.

4.1 Proposed Work

Following is the work proposed for generating test cases to evaluate the load testing of web-server:

- Generation of Test Cases based on the attributes of Form in the Web Application.
- Simulation and Implementation of the Fuzzy Random Values generated from the values generator.
- Analysis and deep investigation of the performance of database engine and apache web server based on different file and attribute sizes / length.

The proposed model and its aspects are shown in Figure 4.1 along with the various attributes of it. The work flow used in the model is shown in Figure 4.2.

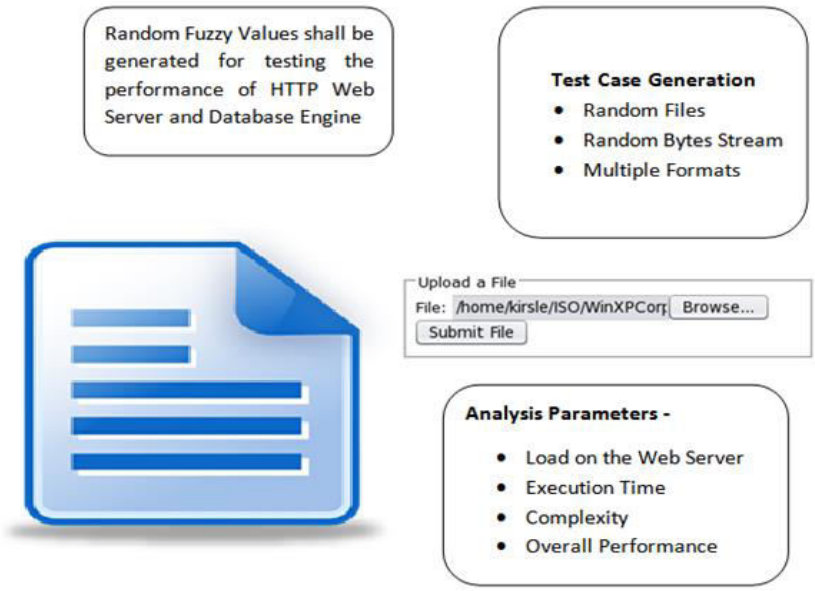


Figure 4.1 - Proposed Model and Aspects

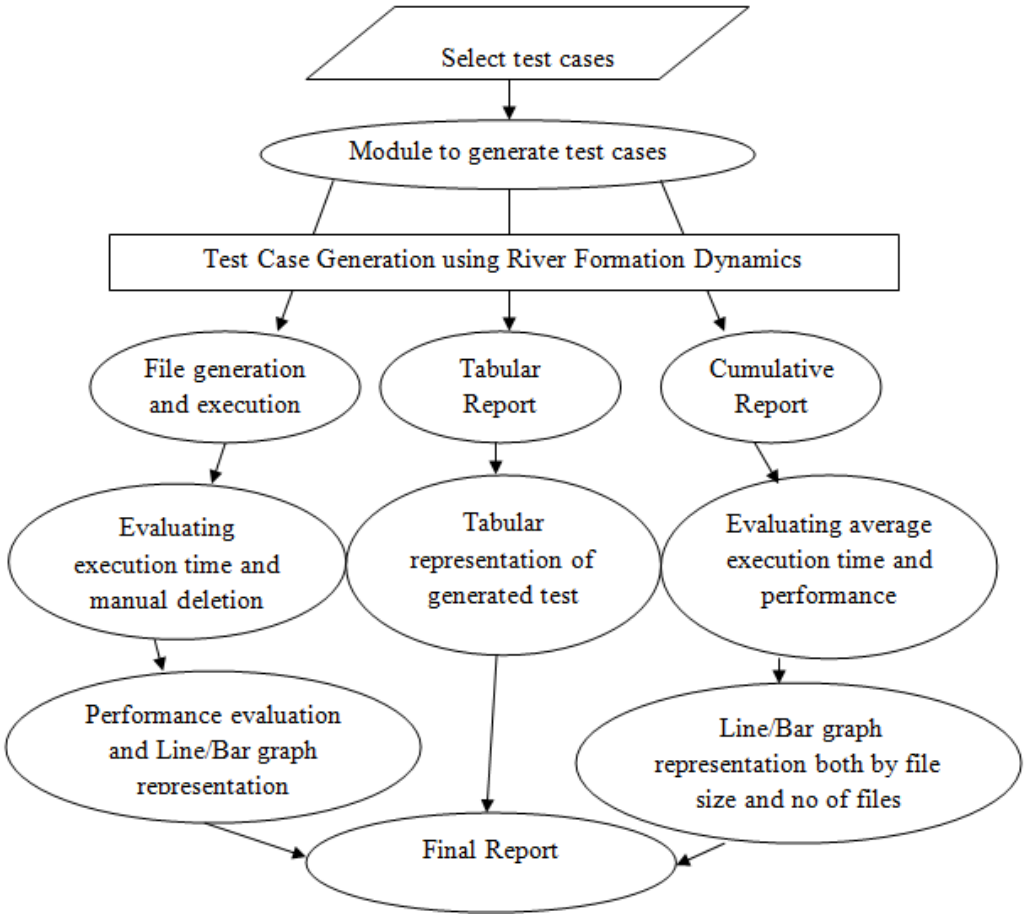


Figure 4.2 - Proposed Flow and Modules

4.2 Tools / Technologies Used

Following tools / technologies shall be used for implementation / simulation scenarios as shown in Figure 4.3:

- PHP 5
- Apache Web Server 2
- MySQL Database Engine 5
- MS Windows 8
- Dia - For Diagrammatic Representation Stable Release

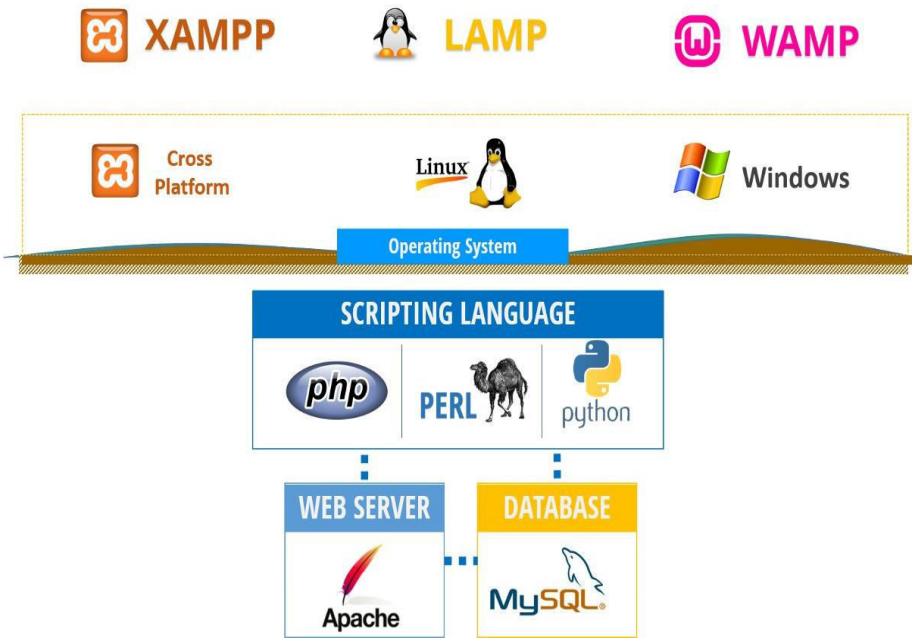


Figure 4.3 - Tools and Techniques

Here deployment has been done and remote databases have been distributed at multiple locations. Every database is having the database relations in separate so that the approach and purpose of the research work can be justified and proved.

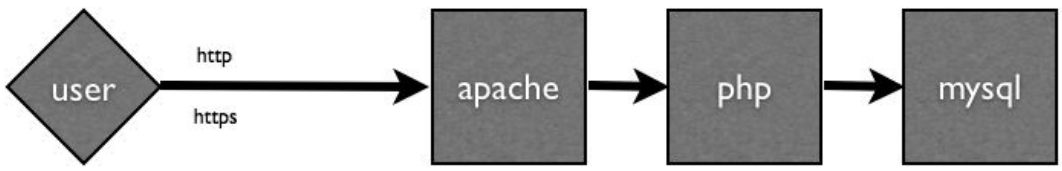


Figure 4.4 - LAMP Stack based on Secured and Non-Secured Connection from User

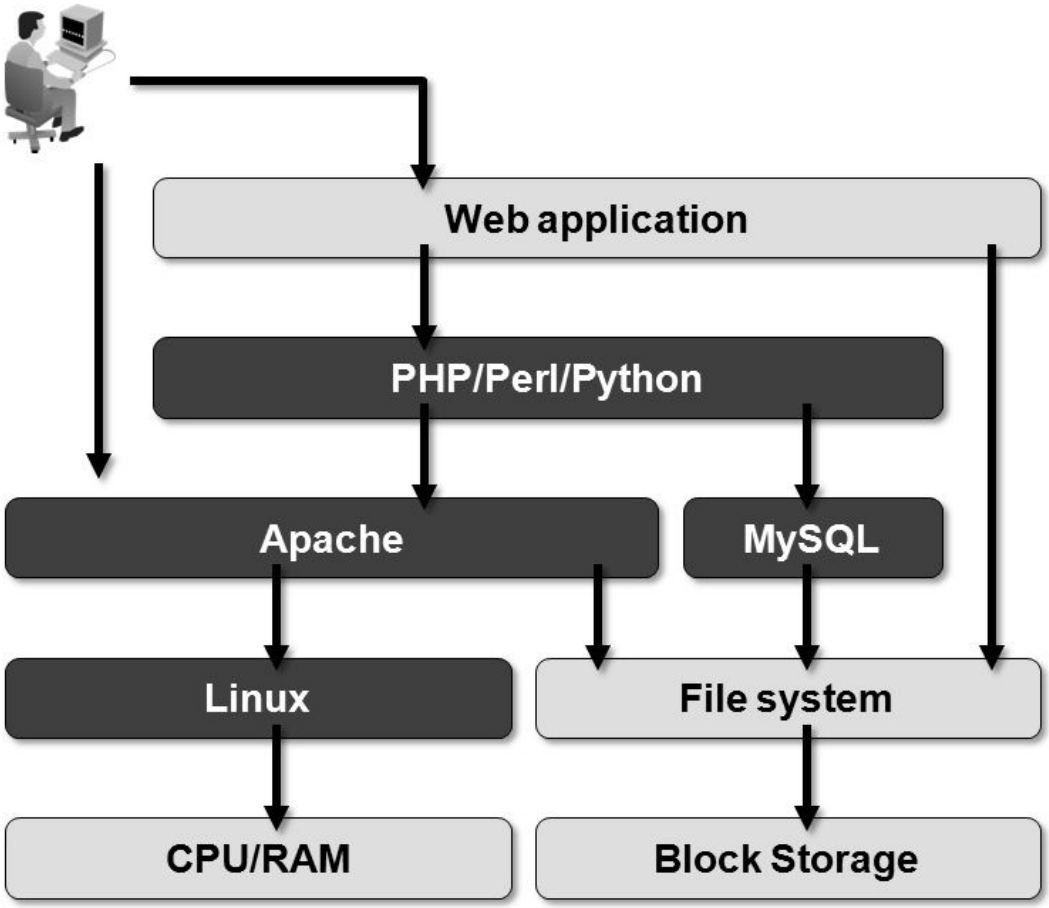


Figure 4.5 - LAMP Based Architecture of the Web Application [11]

Figure 4.4 represents the secured and non-secured connection based LAMP stack from users [5]. Figure 4.5 represents the architecture of the web applications executed with LAMP Platform [11]. The figure shows the components and associates under process and cooperating in the execution of the web pages.

4.3 Implementation Scenario

Using a unique and effective developed simulator, any test cases with any instances are randomly generated using proposed approach River Formation Dynamics Approach and testing is done on Apache HTTP Web Server.

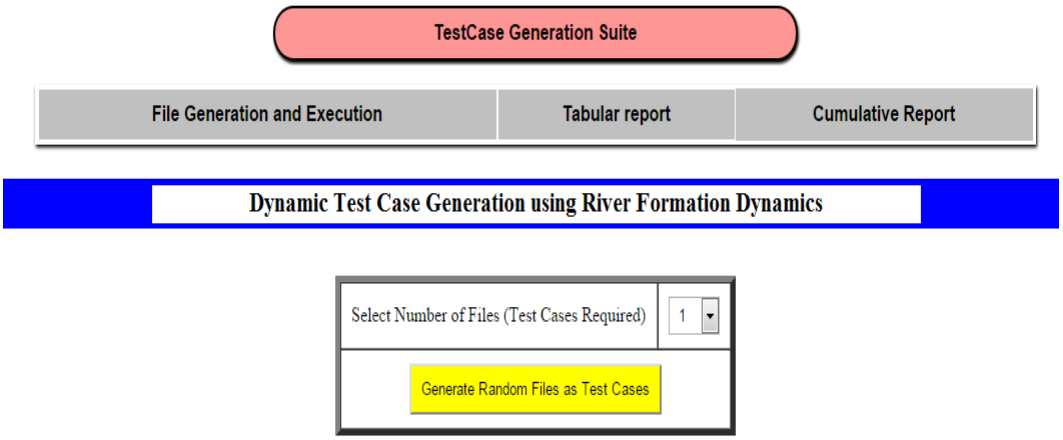


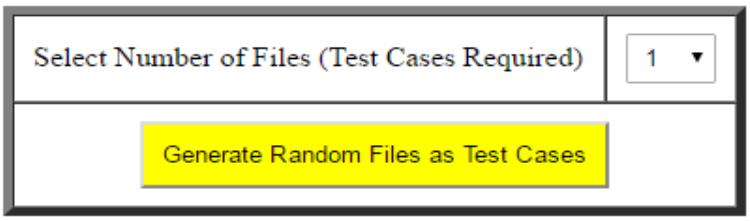
Figure 4.6 - Simulation Scenario

Figure 4.6 shows the screen-shot of the Test-Case Generation Suite. Here the random files can be generated in the form of test cases. A tabular report can also be generated along with a cumulative report.

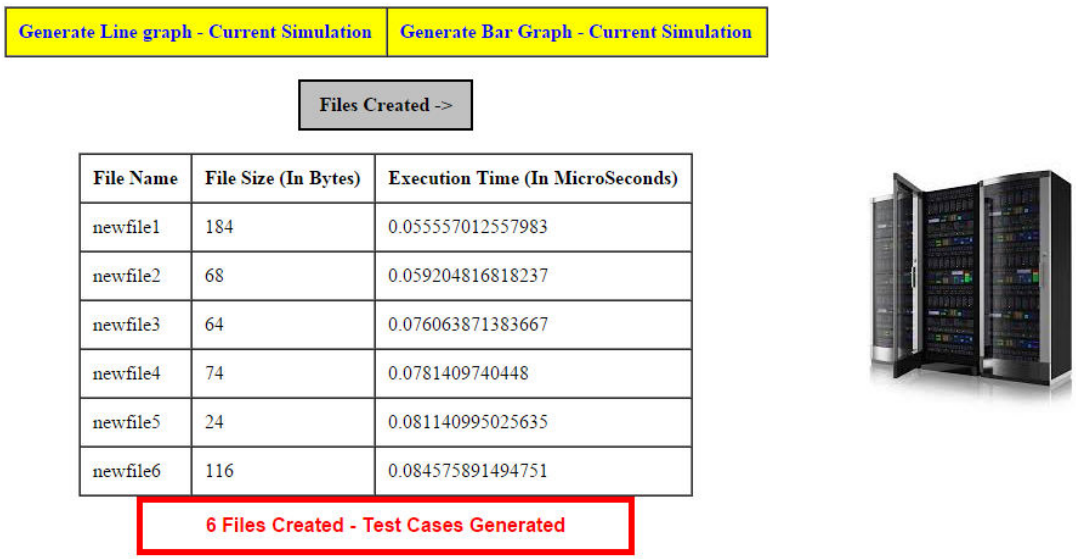
CHAPTER 5

RESULTS AND COMPARISONS

After implementing the proposed system, some results were obtained which illustrate the difference between the prior methods of load testing and the proposed method. The Figure 5.1 (a) and (b) show the screenshots of the implemented system. In Figure 5.1 (a), shown is the system's window through which one can choose the number of test cases desired to be generated by the user. Here in Figure 5.1 (b), six test cases have been randomly generated and the figure shows the file number, its size in bytes and the execution time in microseconds.



(a)



(b)

Figure 5.1 (a) - Generation of Test Cases, **(b) -** Test Cases Generated

In the simulated system, a graph can be generated between the file size and the execution time of the test cases generated randomly. This graph can be a line graph or a bar graph. Figure 5.1 (b) shows two buttons to generate line graph and bar graph for the test cases generated. Corresponding to Figure 5.1 (b), Figure 5.2 shows such a line graph generated for the six test case files.

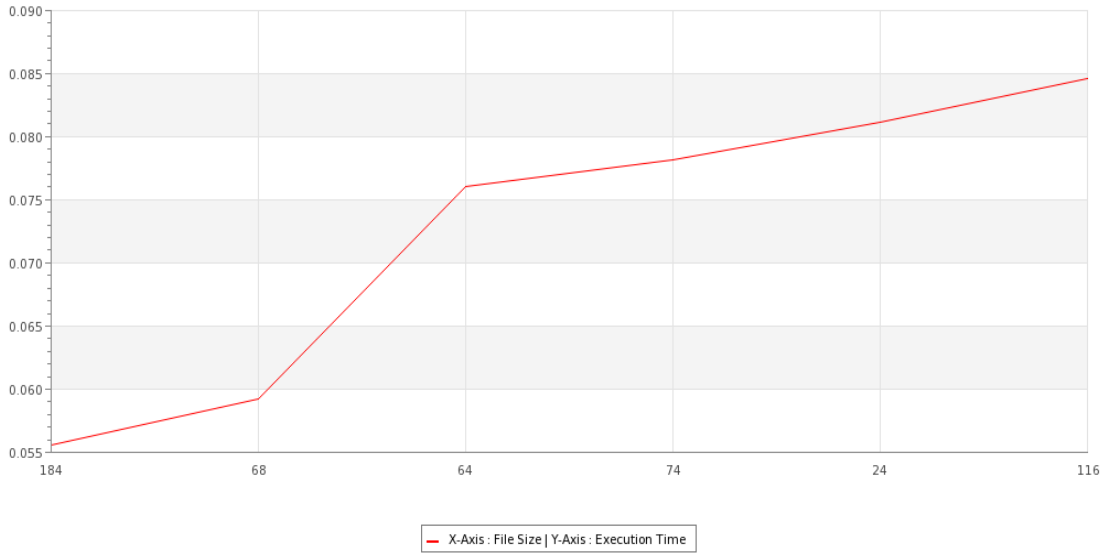


Figure 5.2 - Analysis of the Execution Time based on Assorted File Sizes

The simulated system records all the data for all the files generates so that it can be used to compare later on. The system stores the number of files that were generated each time along with the average file size generated and the average execution time for generated files as shown in Table 5.1 such that we can later on compare the files with respect to their execution time or their file size. The line graph generated corresponding to Table 5.1 has been shown in Figure 5.3 having all the number of generated files.

Table 5.1 - Time Analysis for the Generated Test Cases

Number of files	Average File Size (In Bytes)	Average execution time
1	90	0.132084846
1	90	0.144667149
1	122	0.129695892
1	130	0.154615879
2	33	0.284078836
2	59	0.653749943
2	60	0.211390972
2	46	0.271922112
2	136	0.272706032
2	98	0.328162909
2	131	0.273870945
2	108	0.326394081
2	124	0.231347799
2	114	0.247904062
2	92	0.254753113
2	125	0.285777092
2	121	0.26686883
2	108	0.365943909
3	51.33	0.247397184
3	149.333	0.396366119
3	98.6667	0.371848106
4	110.5	0.365255117
4	53	0.017513037
4	45.5	0.016750813
5	80.8	0.40917182
5	81.8	0.379914999
5	106.4	0.368839025

5	102.8	0.430104017
5	93.6	1.132224083
6	81.6667	0.509562016
6	151.1666667	0.485296965
7	110.2857143	0.658923149
8	88.25	0.674484015
9	117	0.738033056
9	119.11	0.854396105
10	96.2	0.88614893
11	119.63	1.158645868
11	84.18181818	0.025059938
12	95.5	0.959079027
12	108.5	1.051032066
12	103.333	1.01554203
12	93.6667	1.070648909
14	107.7142857	0.032500982
15	114.933	1.135213852
15	99.33	1.159090042
17	120	1.418420076
19	91.57894737	1.410146952
24	101.9166667	2.064305067
48	89.33	5.975867033

A more detailed comparison can be done while taking into consideration same number of files generated each time and comparing their average file sizes and their average execution time. Such a comparison has been shown in Table 5.2 taking the number of files to be fixed *i.e.* 2 in this case. Corresponding to the data in Table 5.2, a line graph has been generated which shows how the execution time varies each time having same number of files but different average file size as shown in Figure 5.4.

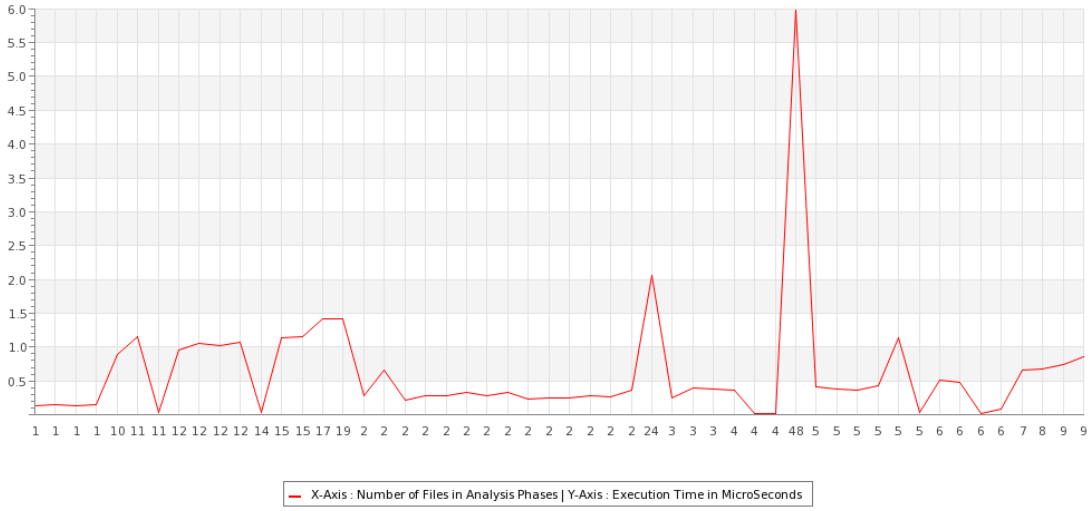


Figure 5.3 - Number of Files and Relative Execution Time

Table 5.2 - Simulation Aspects of Test Cases with Time

Simulation Attempt	Number of files	Average File Size (In Bytes)	Average execution time
1	2	33	0.284078836
2	2	46	0.271922112
3	2	59	0.653749943
4	2	60	0.211390972
5	2	92	0.254753113
6	2	98	0.328162909
7	2	108	0.326394081
8	2	108	0.365943909
9	2	114	0.247904062
10	2	121	0.26686883
11	2	124	0.231347799
12	2	125	0.285777092
13	2	131	0.273870945
14	2	136	0.272706032

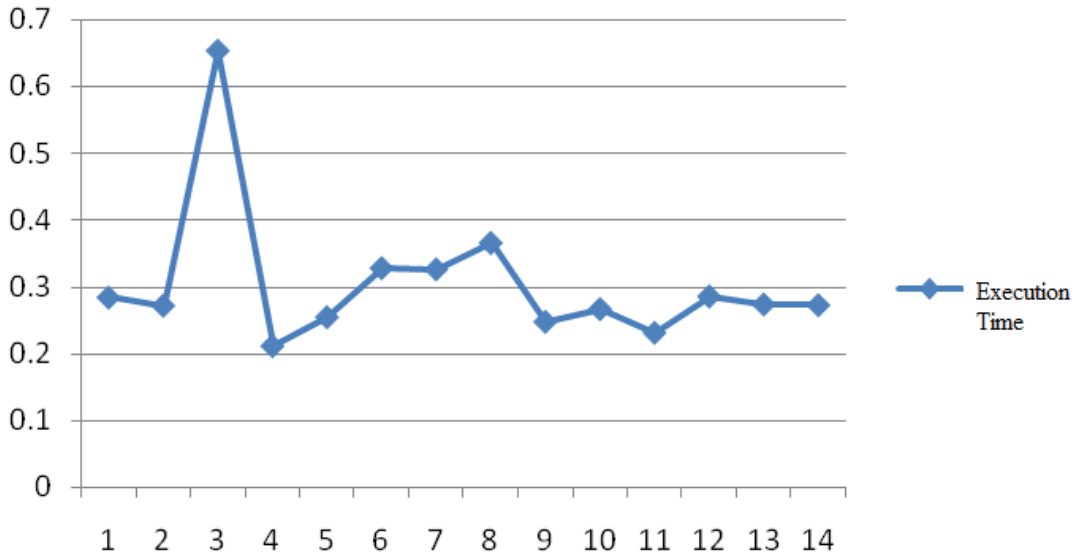


Figure 5.4 - Analysis of Integrity and Consistency

Another comparison has been made based on same number of files generated each time and comparing their file sizes and execution time. Table 5.3 shows such a comparison taking 5 numbers of files each time. Again corresponding to Table 5.3, a line graph has been generated to see how execution time varies as shown in Figure 5.5.

Table 5.3 - Simulation Results

Simulation Attempt	Number of files	Average File Size (In Bytes)	Average execution time
1	5	80.8	0.40917182
2	5	81.8	0.379914999
3	5	93.6	1.132224083
4	5	102.8	0.430104017
5	5	106.4	0.368839025

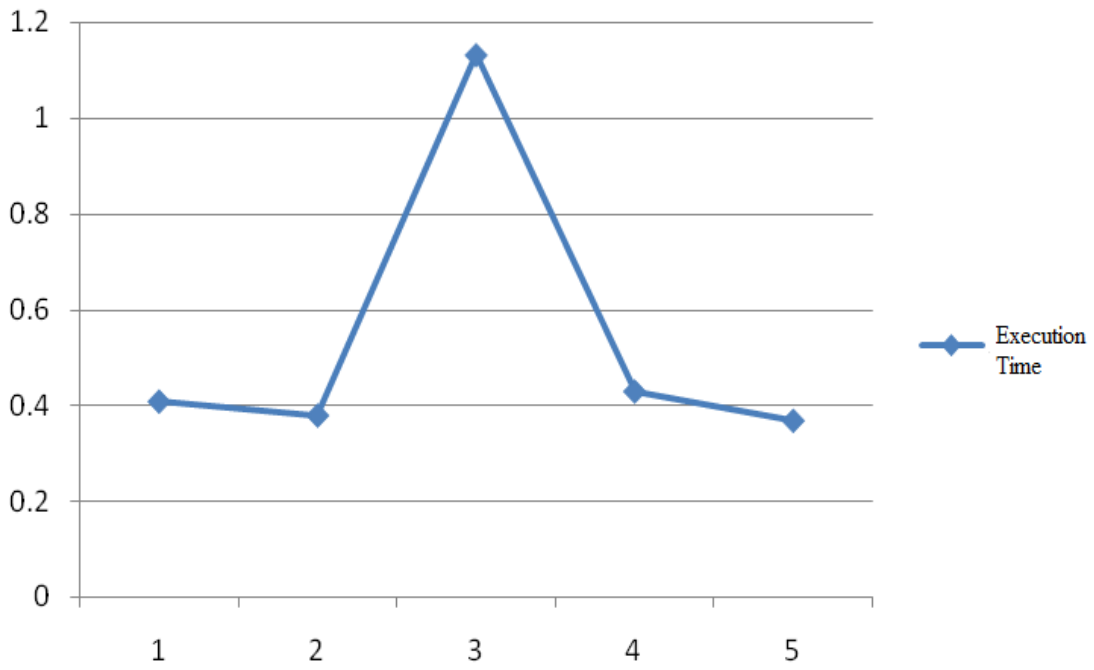


Figure 5.5 - Performance Evaluation in terms of Execution Time

Taking all the test cases from Table 5.1 having number of files generated greater than 5; a table has been generated as shown in Table 5.4. The line graph generated corresponding to Table 5.4 has been shown in Figure 5.6.

Table 5.4 - Results of Simulation in terms of Files in Different Slots of Sizes

Simulation Attempt	Number of Test Cases	File Size (in Bytes)	Turnaround / Process Time(in Microseconds)
1	6	81.66	0.50
2	6	151.16	0.48
3	7	110.28	0.65
4	8	88.25	0.674484015
5	9	117	0.738033056
6	9	119.11	0.854396105
7	10	96.2	0.88614893
8	11	119.63	1.158645868
9	11	84.18	0.025059938

10	12	95.5	0.959079027
11	12	108.5	1.051032066
12	12	103.33	1.01554203
13	12	93.66	1.070648909
14	14	107.714	0.032500982
15	15	114.933	1.135213852
16	15	99.33	1.159090042
17	17	120	1.418420076
18	19	91.57894737	1.410146952
19	24	101.9166667	2.064305067
20	48	89.33	5.975867033

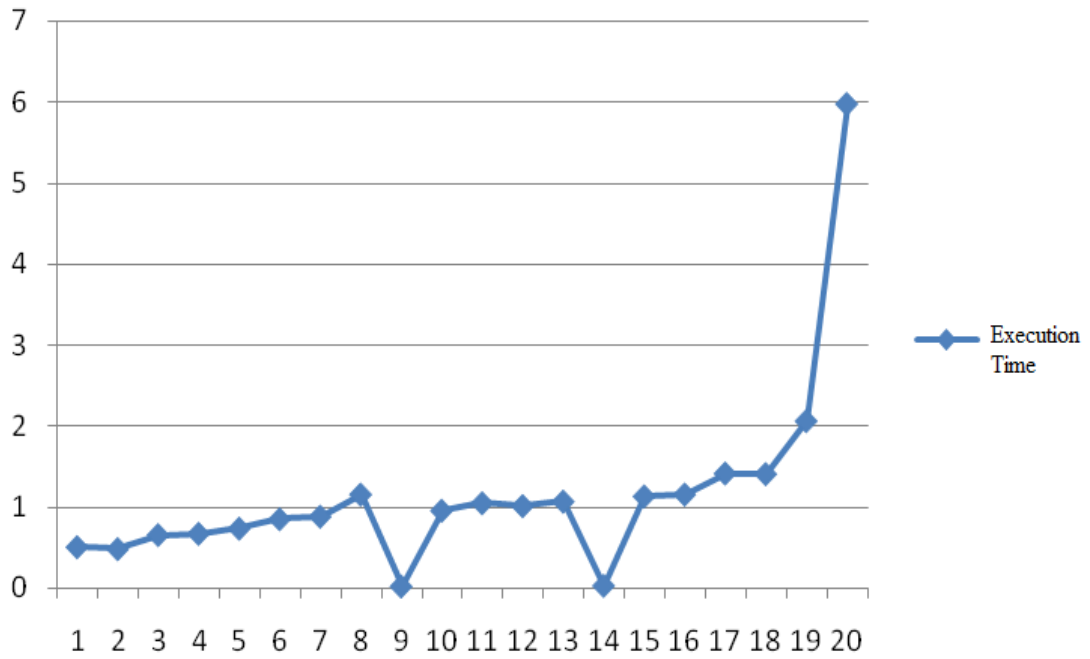


Figure 5.6 - Analysis of Test Cases with Implementation Attempts

For implementation, the execution time of algorithm is taken as the key parameter and factor by which the overall complexity and cost factor and then related performance can be evaluated.

Factors and Parameters Evaluated:

1. Execution Time
2. Cost Factor
3. Performance

Execution Time **-** Execution Time is measured from the implementation results and fetched analogous to turnaround time of the process. The algorithmic approach is implemented and the function for the timestamp and execution time is implemented.

Cost Factor **-** Cost Factor of the process is directly proportion to the process time. If time is more, the cost factor associated will be more. During simulation / implementation, the execution time / evaluation time of the root causes identification is measured and logged in the file system.

The cost factor is one of the key parameter that is taken into the performance evaluation because it is very important to check the optimization time. The equation 1 shows the formula to calculate cost factor using time stamp.

$$C_F = ((TS_i - TS_{i-1}) * O_f)/D_S \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Cost Factor = ((Time Stamp by the using Meta-heuristic River Formation Dynamics Approach - Initial Time Stamp) * Overall Optimization Factor) / DataSets in Analysis

Performance **-** Performance is the ability of a system to consistently perform its intended or required function or mission, on demand and without degradation or failure. Equation 2 shows the formula to calculate performance on the basis of cost factor.

$$P = (((1 / C_F) *100) + R_n) – E_{OV} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Performance = (((1 / Cost Factor) *100) + Random Noise) – Expected Optimization Value

With the span of time, the Performance Factor evaluated using all the meta-heuristic approaches and found that RFD approach which is having association rule mining is performing better and effectively in fetching and predicting the key test cases in the overall set generated.

Table 5.5 compares the classical approach and the proposed approach on the basis of the execution time taking 5, 10, 20 and 50 number of test cases only. Suppose 5 number of files are taken, the obtained execution time in RFD is less than that obtained in the classical approach. This comparison has been plotted in graph as shown in Figure 5.7 taking Execution time and Number of test cases on the axis. Figure 5.7 (a) and (b) show the line graph and bar graph generated respectively.

Table 5.5 - Comparison of Execution Time between Classical and Proposed Approach

Number of Test Cases	**Execution time in River Formation Dynamics**	**Execution time in Classical Approach**
5	0.023234	0.082355
10	0.0632828	0.079888
20	0.0829292	0.09778
50	0.091177	0.099922

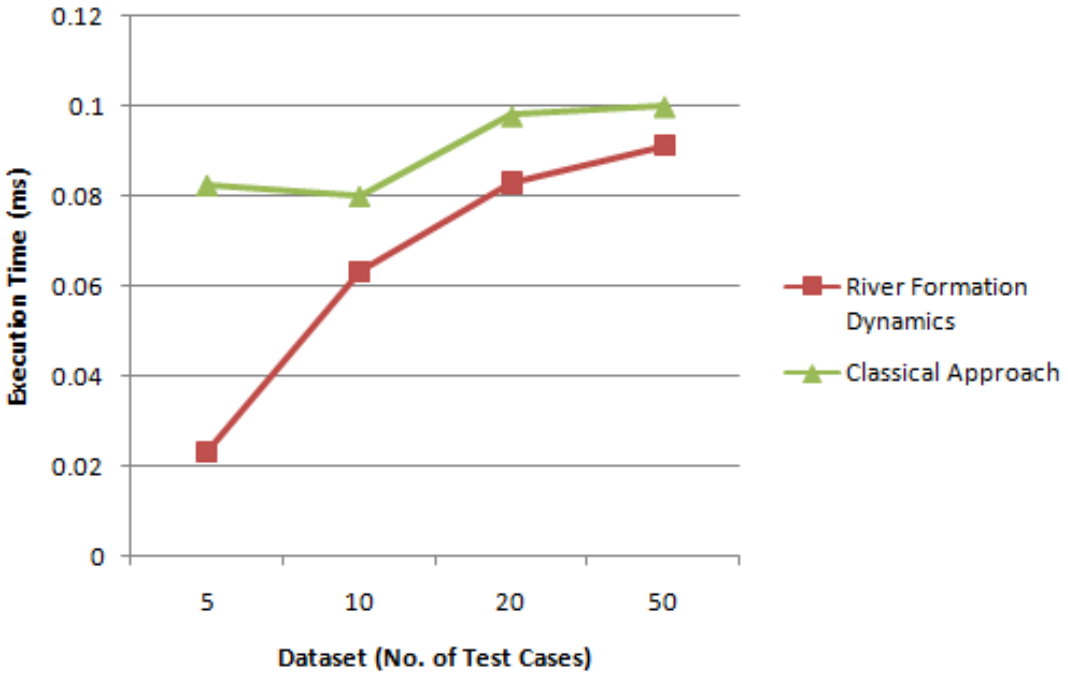


Figure 5.7(a) - Line Graph for Execution Time in Classical and Proposed Approach

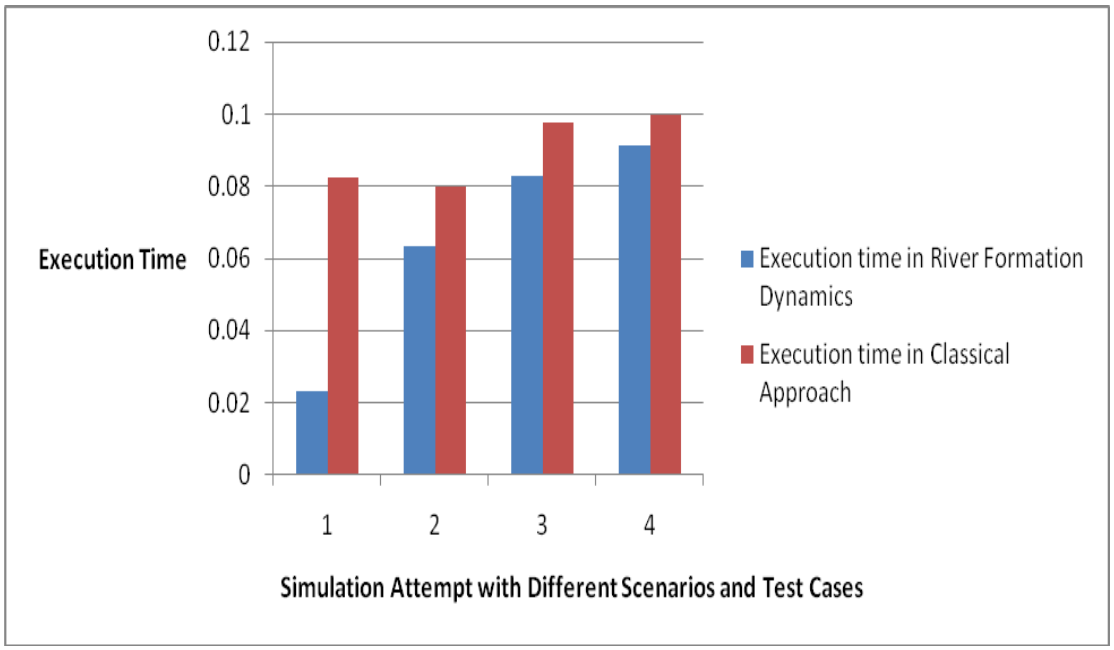


Figure 5.7(b) - Bar Graph for Execution Time in Classical and Proposed Approach

Similar to the above comparison made between the classical approach and the proposed approach based on the execution time, another comparison has been made based on the cost factor in both the approaches. Table 5.6 shows a comparison

between RFD and the classical approach based on cost factor. A graph has been plotted for the above comparison as shown in Figure 5.8.

Table 5.6 - Comparison of Cost Factor between Classical and Proposed Approach

Number of Test Cases	Cost Factor in River Formation Dynamics	Cost Factor in Classical Approach
5	60	80
10	64	87
20	70	92
50	73	94

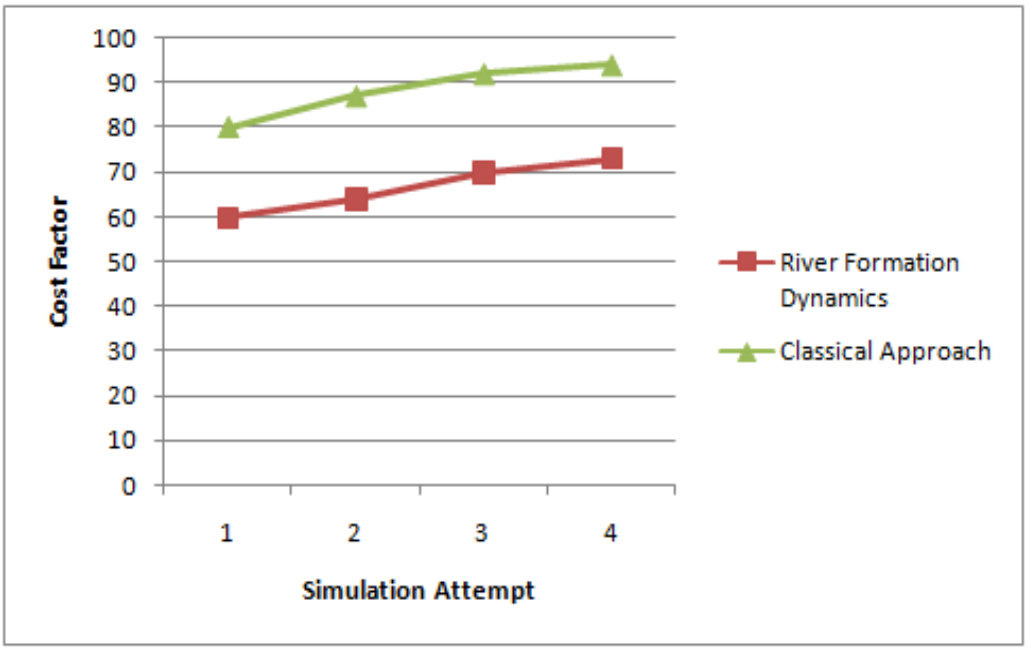


Figure 5.8 - Comparison of Cost Factor in Classical and Proposed Approach

Similarly, a comparison between the performances of both the approaches have been shown in Table 5.7 and the corresponding line graph has been plotted in Figure 5.9.

Table 5.7 - Comparison of Performance in Classical and Proposed Approach

Number of Test Cases	Performance in River Formation Dynamics	Performance in Classical Approach
5	80	64
10	86	67
20	89	50
50	92	78

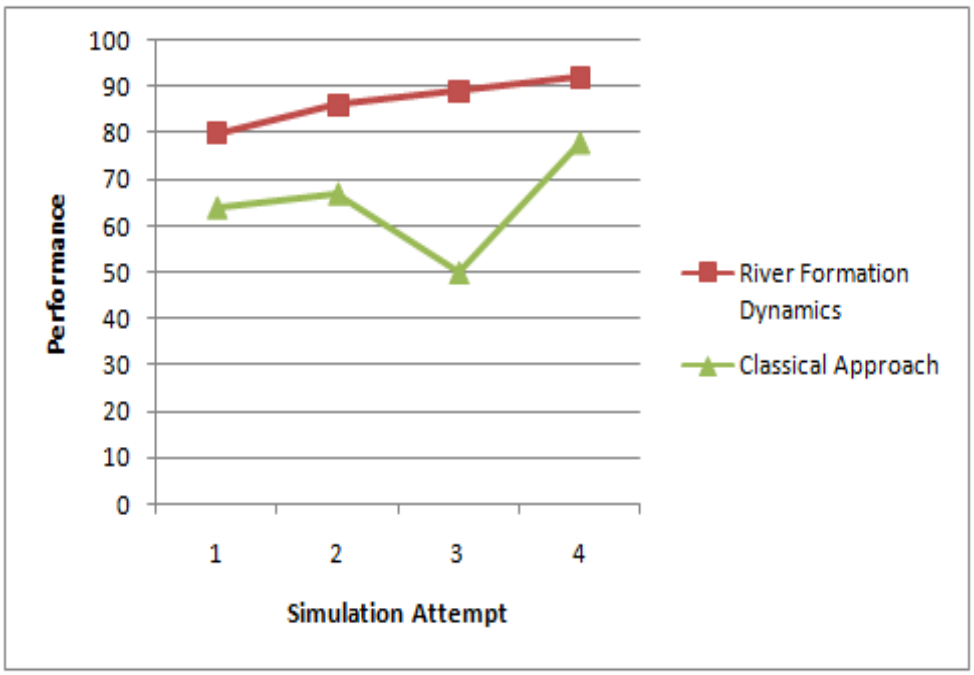


Figure 5.9 - Comparison of Performance Factor in Classical and Proposed Approach

The performance shown in Table 5.7 using River Formation Dynamics has been taken in ascending order in Table 5.8 so as to compare the two performances obtained using the two different approaches. The corresponding line graph and bar graph have been generated as shown in Figure 5.10 (a) and (b) respectively.

Table 5.8 – Performance using RFD taken in Ascending Order

Simulation Attempt	Performance in River Formation Dynamics	Performance in Classical Approach
1	89	67
2	87	69
3	95	56
4	93	78

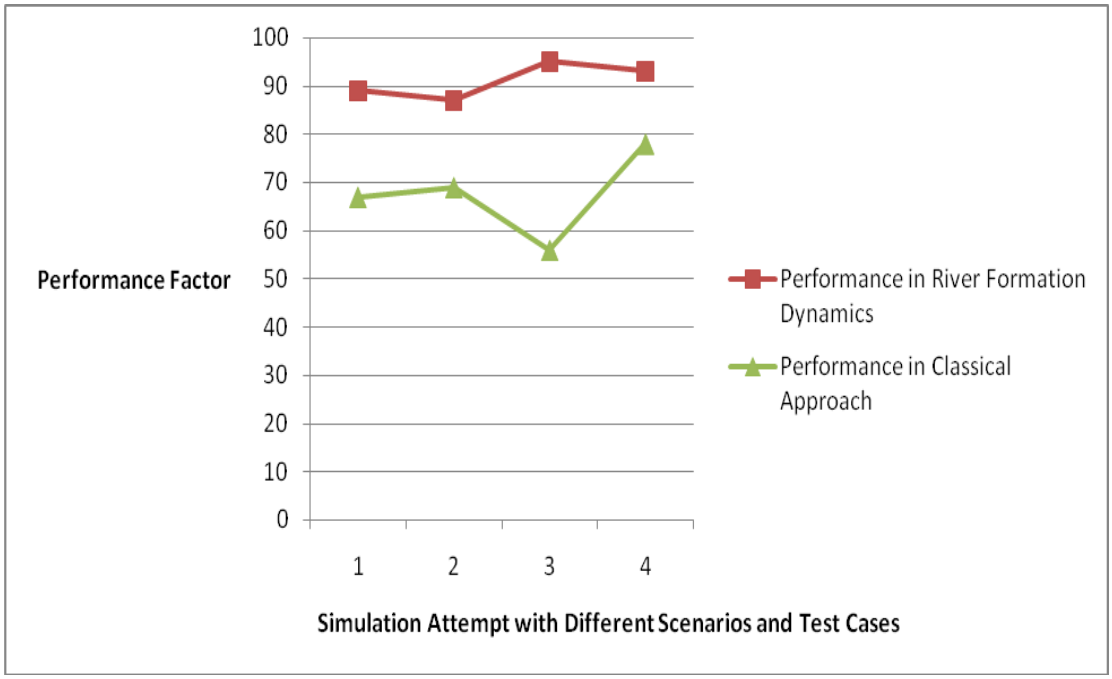


Figure 5.10(a) - Line Graph for Performance Factor in Classical and Proposed Approach

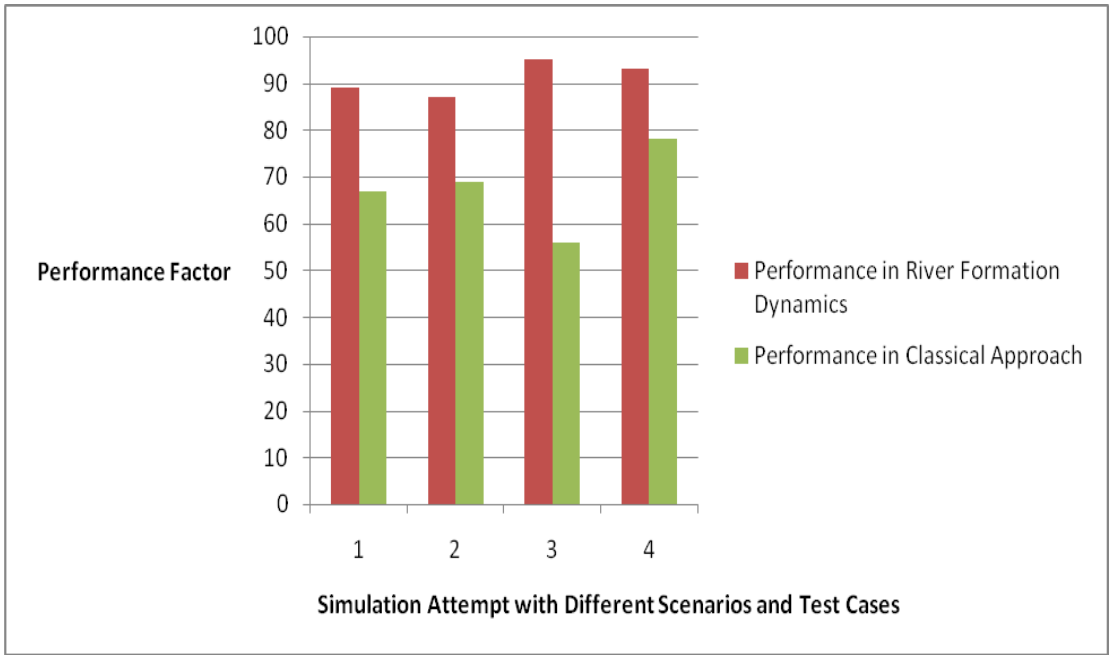


Figure 5.10(b) - Bar Graph for Performance Factor in Classical and Proposed Approach

It is clear from the results that RFD is better approach in terms of overall performance and cost factor.

CHAPTER 6
CONCLUSION & FUTURE WORK

Following are the conclusions made from the simulated system and the methodology used behind along with the future work that can be implemented to make the system work even better.

6.1 Conclusion

In this work, a novel algorithmic approach and model is proposed that will generate the test cases for web application based form. In this approach, the form values shall be inserted and processed by the script and overall performance of the system including web server and database engine shall be evaluated based on multiple parameters. Using this approach, the tolerance level and threshold of database engine as well as server can be set with effective evaluation.

6.2 Future Work

For future work, this work plan can be expanded in the following directions:

- The other meta-heuristic based implementations can be implemented which include approaches like the ant colony optimization, honey bee algorithm, simulated annealing and many other others. Such algorithmic approaches should provide better results when we move more towards meta-heuristics.
- This research work mainly discusses the test case generation using greedy and optimization using river formation dynamics which can be tested and optimized using other hybrid approaches.
- This algorithm can also be extended to be used for the processing of heterogeneous programming paradigms.

REFERENCES

[1] Kuhn, D. R., Wallace, D. R., & Gallo, A. M. (2004). Software fault interactions and implications for software testing. IEEE transactions on software engineering, 30(6), 418-421.

[2] Bertolino, A. Mooney, C. Z. (2007, May). "Software testing research: Achievements, challenges, dreams. In 2007 Future of Software Engineering (pp. 85-103). IEEE Computer Society.

[3] N. Tracey, Clark, J. A., & Mander, K. (1998). The way forward for unifying dynamic test-case generation: The optimisation-based approach. In Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications (DCIA).

[4] Rayadurgam, S., & Heimdahl, M. P. E. (2001). Coverage based test-case generation using model checkers. In Engineering of Computer Based Systems, 2001. ECBS 2001.Proceedings. Eighth Annual IEEE International Conference and Workshop on the (pp. 83-91). IEEE.

[5] Ali, S., Briand, L. C., Hemmati, H., & Panesar-Walawege, R. K. (2010). A systematic review of the application and empirical investigation of search-based test case generation. Software Engineering, IEEE Transactions on, 36(6), 742-762.

[6] Perrouin, G., Sen, S., Klein, J., Baudry, B., & Le Traon, Y. (2010, April). Automated and scalable t-wise test case generation strategies for software product lines. In Software Testing, Verification and Validation (ICST), 2010 Third International Conference on (pp. 459-468). IEEE.

[7] Rasal, Y. M., & Nagpure, S. (2015). Web Application: Performance Testing Using Reactive Based Framework. IJRCCT, 4(2), 114-118.

[8] Yan, M., Sun, H., Liu, X., Deng, T., & Wang, X. (2015). Delivering Web service load testing as a service with a global cloud. Concurrency and Computation: Practice and Experience, 27(3), 526-545.

[9] F. Shahzad, M. Shahzad and M. Farooq, In-execution Dynamic Malware Analysis and Detection by Mining Information in Process Control Blocks of Linux Operating System, Data Mining for Information Security, Vol. 231, pp.45-63, 2013.

[10] K. Allix, Q. Jerome, T.F. Bissyande, J. Klein, R. State and Y.L. Traon, A Forensic Analysis of Android Malware--How is Malware Written and How it Could Be Detected?, 38th IEEE Annual Computer Software and Applications Conference (COMPSAC), Vasteras, pp. 384-393, July 2014.

[11] S. Alam, R. N. Horspool and I. Traore, MARD: A framework for metamorphic malware analysis and real-time detection, 28th IEEE International Conference on Advanced Information Networking and Applications (AINA), Victoria, pp. 480-489, May 2014.

[12] M.S Alam and S.T. Vuong, Random forest classification for Detecting Android Malware, IEEE conference on Green Computing and Communications (Green Com) and Internet of Things (I Things/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing , Beijing, pp. 663-669, August 2013.

[13] M. Z. Rafique and J. Caballero, Firma: Malware clustering and network signature generation with mixed network behaviors, In Research in Attacks, Intrusions, and Defenses, Vol. 8145, pp. 144-163, 2013.

[14] N. Idika and A.P Mathur, A Survey of Malware Detection Techniques, Department of Computer Science, Purdue University, West Lafayette, pp. 1-48, February 2007.

[15] K. Mathur, S. Hiranwal, A Survey on Techniques in Detection and Analyzing Malware Executables, International Journal of Advanced Research in Computer Science and software engineering, Vol. 3, Issue 4, April 2013.

[16] A. Saeed, A. Semant, A. M.A. Abuagoub, A Survey on Malware and Malware Detection Systems, International Journal of Computer Applications, Vol. 67, Issue 16, pp. 25-31, April 2013.

[17] J. Landage, Prof. M. P. Wankhade, Malware and Malware Detection Techniques: A Survey, International Journal of Engineering Research & Technology (IJERT), Vol. 2 , Issue 12, pp. 1-8 , December 2013.

[18] M. Overton, Anti-Malware Tools: Intrusion Detection Systems, EICAR conference, Malta, pp. 1-22, May 2005.

[19] S. Das, Y. Liu, W. Zhang, and M. Chandramohan ,Semantics-Based Online Malware Detection: Towards Efficient Real-Time Protection Against Malware, IEEE

transactions on information forensics and security, Vol. 11, Issue 2, pp. 289 - 302 , February 2016.

[20] J. Rabek, R. Khazan, S. Lewandowski and R. Cunningham, Detection of injected, dynamically generated, and obfuscated malicious code, Proceedings of the ACM Workshop on Rapid Malcode, USA, pp. 76–82, 2003.

[21] S. Shah, H. Jani, S. Shetty, K. Bhowmick, Virus Detection using Artificial Neural Networks, International Journal of Computer Applications, Vol. 84, Issue 5, December 2013."

Annexure I

List of Publications

- Gurleen Kaur Sandhu and Rupinderdeep Kaur, Effective Test Case Generation for Load Testing of Web Server using River Formation Dynamics in International Conference on Electrical, Electronics, Computer Science, Management and Mechanical Engineering (ICE2CSM2E2016) under International Institute of Technology & Research [Published]

Annexure II

Video Link

https://www.youtube.com/watch?v=RtmekfcPsBA

Annexure III

Plagiarism Certificate

Turnitin Originality Report

g by G G

From thesis (PhD)

- Processed on 04-Jul-2016 17:10 IST
- ID: 687824155
- Word Count: 6567

Similarity Index
12%
Similarity by Source

Internet Sources:
11%
Publications:
4%
Student Papers:
N/A

sources:

1 1% match (Internet from 09-Mar-2016)
http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html

2 1% match (Internet from 23-May-2016)
http://www.softwaretestingclass.com/software-testing-life-cycle-stlc/

3 1% match (Internet from 09-Jun-2016)
http://softwaretestingfundamentals.com/software-testing-life-cycle/