# Test Language Processing: A Novel Approach for Automated Software Testing

Mukesh Mann and Om Prakash Sangwan

*School of Information & Communication Technology, Gautam Buddha University, Greater Noida, Uttar- Pradesh*
mukesh.gbu@gmail.com , sangwan_op@yahoo.co.in

*Abstract :* **The assurance of software reliability partially depends on testing. However it is interesting to note that testing itself needs to be reliable. The generation of test data is not an easy and straightforward process as it requires massive efforts and time. Numbers of approaches are available with their proclaimed advantages and limitations, but accessibility of any one of them is a subject dependent. Time is a critical factor in deciding cost of any project. A deep insight has shown that executing manual test cases are time consuming and tedious activity.**

**In this paper we have proposed a novel approach for test case generation and execution which contains faith of both automation and manual tester. The proposed methodology is named as "Test Language Processing (TLP)" which is served as a comprehensive test approach that takes the responsibility of automation plan, design and the execution of functional test cases based on dictionary oriented solution. In our work we have illustrated how the TLP could serve as beginning of a dictionary/keyword oriented approach to deliver testing as a service in a much better way than the traditional testing approaches. The proposed methodology is applied to an open source application called Vtiger CRM5. Experimented results shows that automation time using TLP approach indicates a significant time saving in testing.**

*Keywords. :* **Automation, Manual Testing, Test Language Processing, Dictionary, Keyword driven.**

## I. INTRODUCTION

In the agile environment delivery of functional quality product has taken its priority. Software functional quality reflects how well it complies with or conforms to a given design, based on specifications and functional requirements or, this attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the market place as a worthwhile product [1]. Therefore to deliver such functional qualities different testing approaches have been proposed during last few decades. One such technique is boundary value analysis in which we concentrate on input values and design the test cases with input values that are on or close to boundary values. Other functional testing techniques include robustness testing which is the extension of boundary value analysis. Equivalence testing is also a functional test generation technique in which entire input domain can be divided into at least two equivalence classes: one containing valid inputs and other containing all invalid inputs [2]. Cause and effect graphing technique is also a functional testing which considers the combinations of various inputs which were not available in boundary value analysis and equivalence class testing.

Test automation is the next logical step for organizations progressing towards establishing a mature quality assurance program. There are many alternatives while deciding to invest in test automation tools. Making the correct investment is crucial success of initiatives [3].

## II. RELATED WORK

During the last few years stress has been given on test case automation in which we have a high Return on Investment (ROI). Automated test data generation is an activity that generates test data automatically for the software under test. The oath of software reliability somewhat depends on testing. Automating the test process is a sound engineering approach, which can make the testing efficient, cost effective and consistent. A number of automatic test case generation exist like random testing in which test data is generated arbitrarily and software will execute by taking that data as its input. Symbolic execution is also an automated test generation method. Many early techniques of test data generation used symbolic execution for the generation of test data in which we assign a symbolic value to variables instead of actual values so as to generate an expression in terms of input variables.

Test data generation in program testing is the process of identifying a set of test data which satisfies given testing criterion. Most of the existing test data generators use symbolic evaluation to derive test data [5,7,10,12,14]. Symbolic execution is not the execution of a program in its true sense, but rather the process of assigning expression to program variables as a path is followed through the code structure [4]. We may define

a constraint system with the help of input variables which determines the conditions that are necessary for traversal of a given path but this technique has a problem of infinite loops. As against symbolic execution, dynamic test data generation technique requires actual execution of the program with some selected inputs and the value of variables are known during the execution of the program. Dynamic test data generation is based on the idea that if some desired test requirement is not satisfied, the data collected during execution can still be used to determine which test come closest to satisfaction requirement.

To reduce the burden of manually writing unit tests, many automated test generation techniques have been proposed [9, 11, 13, 15, 17,18]. The use of tools for the generation of test data is still in its infancy, although some software industries have been using their own tools for the generation of test data. The output of such a tool is a set of test data, which include a sequence of inputs to the system under test. One possible way to improve the effectiveness of automated testing techniques is to use a formal specifications to guide test generation [8, 16]. However such a specifications are often absent in practice. Various industrial tools are developed to support automatic test execution such as Rational Functional Tester from IBM and Quick Test Professional from Mercury [6], these tools accept manually created, automatically generated, predefined test sequence and executes the sequence without human intervention and supervision.

## III. METHODOLOGY

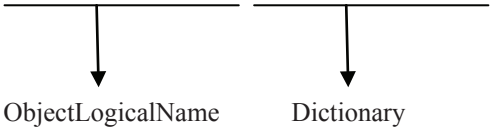
Our approach in based on the processing of tester specified natural language called as Test Language Processing. A Test language is defined as:

"A tester specified dictionary of keywords that facilitate communication among testers and other subject-matter specialists."

Where a keyword is a natural language english word.
The manual tester will specify the keyword according to his/ her understanding but usually meaningful keywords are specified along with their values.

a) Structure of test language

It comprises of dictionary, which contains words called "Keywords" and parameters.




Where ObjectLogicalName is the name of object that the tester is looking for and the keyword is a common english word specified in Natural Language. Keyword tells us what type of operation we want on a particular object. For example for an object like Login_UserName we always make an input as an operation so we can take "INPUT" as a keyword to specify the type of operation on this object. Parameter to the object tells us object's logical value. We focus here that the keyword can be in Natural Language but we usually specify meaningful keywords so that the testing team can better understand the keyword terms.

b) The Test Language Processing (TLP): Infrastructure Management

As we have already mentioned that the key role played in TLP are of functional testers and automation testers. So functional tester is responsible for updating of keywords and automation tester's responsibility is to develop and organize test scripts so as to call and implement the keywords specified by functional tester. Figure1 shows ind ividual responsibilities of two teams.

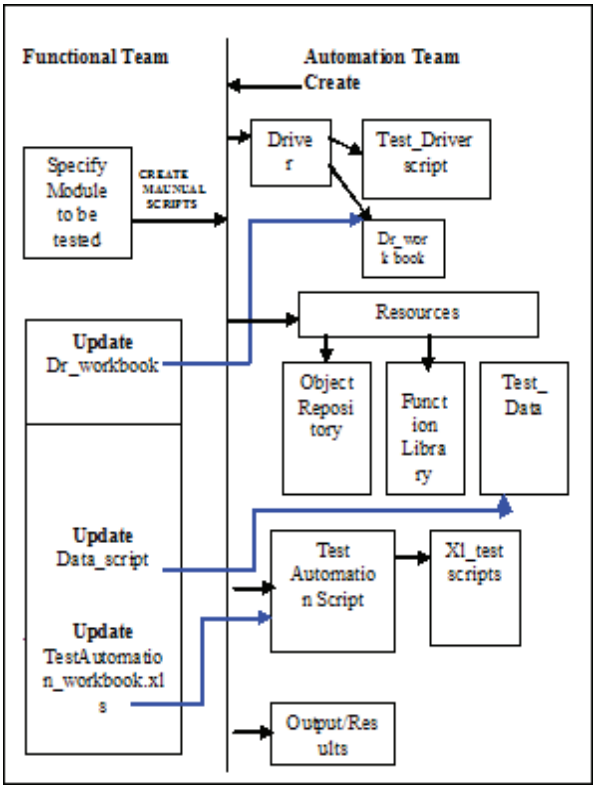



FIGURE 1: TEAM RESPONSIBILITIES IN TLP APPROACH

Where

- Module: - It the sub part or module of an application for which we want to implement TLP.
- Driver: - It a folder which contain Test Driver script and driver workbook in excel format.
- Test driver Script: - It is the actual code which implements TLP.
- Driver Workbook: - It is an excel file where we Specify Which Module to be Tested Using TLP approach.
- Test Automation Script: - It is a folder which contains actual Keywords specified by functional Tester in excel format called XL_testscripts and this is driven by driver Workbook.
- Functional Libraries: - it contains actual functions implementing Test driver Scripts.
- Object Repository: - A resource used by function libraries to identify objects in application under test.
- Output: This contains the implementation results.

c) Working Model for TLP

As explained in TLP infrastructure, The TLP methodology is implemented collaboratively by both functional and automation testing team as shown in figure 2.

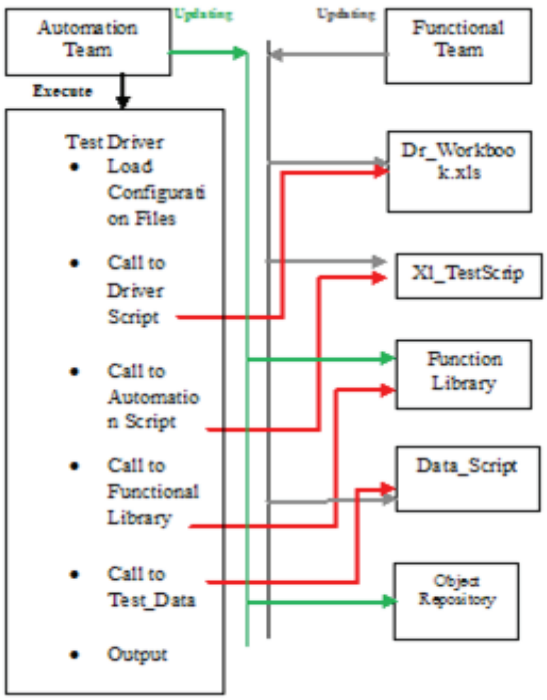



FIGURE 2. WORKING MODEL FOR TLP

A step by step approach is discussed below:

1. After clearly defining the responsibilities as explained in TLP infrastructure, the automation team starts writing the test scripts. Configuration file (that contain the path of system under test application) is first loaded into the QTP environment.
2. Automation tester now starts writing the actual code to drive the sequence mentioned in the driver script. The code is designed in such a way that it drives the automation script sequence according to the driver script sequence. Therefore call to driver scripts followed by automation scripts is clearly coded in the framework. The functional libraries are used to provide the actual functional implementation for each task mentioned in the test automation script. So designing and coding of functional library becomes necessity, without which the framework will not run successfully.
3. The rest responsibilities of individual team regarding updating their tasks are same as mentioned in the TLP infrastructure management.
4. The output is lastly designed in such a way that it will show the total testing time for a particular module, number of test cases successfully executed and screenshots for the failed test cases.

## IV. EXPERIMENTAL RESULTS

We have implemented TLP methodology for Sign in -Sign out button of an open source application called VTiger CRM5 [19]. It is an open source customer relationship management application that provides a number of modules to manage CRM tasks. Twenty six lead members (called Subject from now) were taken having 1 to 3 years of industry experience. They were asked to test the sign in sign out module (a reusable module in application) with same data set formed on the basis of boundary value analysis. In other way we can say that each lead is login once and then it is logout and this process is carried out by different twenty six leads ( with same set of data for each lead). We make ten iterations (although we may do more iterations but we have only considered only ten to indicate the average testing time for a particular Lead) for each lead because it might be possible that the login functionality may get change after first login or second login or third login and so on. So we login the said application ten times with different authenticated users as specified in SRS.

Data Set Used

We have applied boundary value analysis (BVA) for each of the subject lead. For example we take lead as specified in SRS as

Authentic username =”admin”

Authentic password =”admin”

We apply BVA to the username and password field and form the data set as shown in table 1.

TABLE 1: DATA SET FORMED USING BOUNDARY VALUE ANALYSIS FOR USER AND PASSWORD FIELD

| Username | Password | Status |
|---|---|---|
| Admin | @#$ | Failed |
| @#$ | Admin | Failed |
| Admin | 00abc | Failed |
| 00abc | Admin | Failed |
| ADMIN | ADMIN | Failed |
| JavaScript | JavaScript | Failed |
| Admin | JavaScript | Failed |
| “” | “” | Failed |
| Admin | Admin | Passed |

The other functionalities for this module are clearly defined in data set formed on the basis of boundary value analysis for each Lead.

The developed methodology is executed in QTP10.00 version.
The time to complete this task manually is shown in table 2.

TABLE 2: MANUAL TESTING TIME (IN SECONDS) TAKEN BY EACH SUBJECT FOR DIFFERENT ITERATION

| **Subject wise (Time) /No. of Iterations** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Subject 1** | 12.3 | 22.5 | 34.86 | 48.39 | 68.55 | 80.77 | 91.55 | 101.32 | 112.36 | 124.56 |
| **Subject 2** | 13.4 | 22.22 | 34.68 | 49.23 | 69.26 | 80.99 | 91.56 | 102.2 | 112.986 | 124.56 |
| **Subject 3** | 13.5 | 23.3 | 35.22 | 48.77 | 69.23 | 80.78 | 92.52 | 101.85 | 112.69 | 124.35 |
| **Subject 4** | 12.1 | 22.8 | 34.77 | 48.85 | 69.56 | 80.69 | 91.67 | 101.65 | 112.68 | 124.94 |
| **Subject 5** | 12.6 | 21.7 | 35.44 | 48.99 | 69.47 | 81.2 | 91.61 | 101.98 | 112.78 | 124.68 |
| **Subject 6** | 12.8 | 22.6 | 34.68 | 48.68 | 68.65 | 81.5 | 91.64 | 101.79 | 112.39 | 125.35 |
| **Subject 7** | 14 | 22.8 | 34.58 | 49.56 | 68.89 | 80.47 | 91.56 | 101.89 | 112.98 | 125.36 |
| **Subject 8** | 13.6 | 21.2 | 35.45 | 49.65 | 69.82 | 80.96 | 91.63 | 101.94 | 112.78 | 124.98 |
| **Subject 9** | 13.9 | 22.1 | 34.98 | 49.22 | 69.36 | 80.67 | 91.56 | 101.92 | 112.98 | 124.97 |
| **Subject 10** | 12.3 | 21.9 | 35.78 | 48.38 | 68.49 | 80.79 | 91.67 | 101.97 | 112.48 | 125.85 |
| **Subject 11** | 12.7 | 22.4 | 35.98 | 48.97 | 68.75 | 80.97 | 91.87 | 102.56 | 113.12 | 124.65 |
| **Subject 12** | 13.6 | 22.6 | 34.56 | 49.56 | 68.73 | 80.96 | 91.98 | 101.99 | 113.6 | 124.98 |
| **Subject 13** | 14.2 | 22.2 | 34.87 | 48.29 | 68.71 | 81.4 | 91.92 | 101.98 | 113.84 | 125.94 |
| **Subject 14** | 13.4 | 21.4 | 35.66 | 49.56 | 68.72 | 81.94 | 91.24 | 101.79 | 112.68 | 125.67 |
| **Subject 15** | 13.8 | 21.7 | 34.82 | 49.58 | 68.73 | 81.64 | 91.56 | 101.75 | 112.98 | 124.68 |
| **Subject 16** | 13.8 | 22.8 | 35.88 | 48.67 | 68.79 | 81.64 | 91.75 | 101.89 | 112.96 | 124.67 |
| **Subject 17** | 13.7 | 21.6 | 35.95 | 48.69 | 68.81 | .81.67 | 91.81 | 101.91 | 112.78 | 125.69 |
| **Subject 18** | 12.4 | 22.6 | 34.69 | 49.81 | 68.51 | 81.67 | 91.38 | 101.99 | 113.69 | 125.46 |
| **Subject 19** | 12.01 | 23.5 | 35.66 | 49.2 | 69.52 | 80.64 | 91.68 | 101.69 | 113.16 | 124.69 |
| **Subject 20** | 14.2 | 22.8 | 36.21 | 49.69 | 69.64 | 80.69 | 92.2 | 102.89 | 112.89 | 124.99 |
| **Subject 21** | 14.6 | 21.2 | 35.82 | 48.25 | 69.56 | 80.79 | 91.64 | 102.59 | 112.96 | 125.67 |
| **Subject 22** | 13.9 | 22.6 | 34.11 | 49.78 | 98.76 | 81.68 | 91.19 | 102.11 | 112.89 | 124.59 |
| **Subject 23** | 12.9 | 22.9 | 34.98 | 48.69 | 68.91 | 81.97 | 91.18 | 101.85 | 112.98 | 125.67 |
| **Subject 24** | 12.8 | 22.4 | 34.56 | 49.2 | 68.38 | 81.79 | 91.38 | 101.65 | 112.93 | 125.49 |
| **Subject 25** | 13.4 | 21.9 | 35.6 | 49.6 | 68.64 | 80.93 | 92.12 | 102.11 | 113.53 | 125.97 |
| **Subject 26** | 13.6 | 22.1 | 35.66 | 48.67 | 68.73 | 80.96 | 92.64 | 102.14 | 112.98 | 125.69 |

We add 5 second latency time which means that the input characteristics by individual depends on number of factors like typing speed, sharpness in doing their work. So for each subject we add a 5 second latency time which represent the average common delay in making input to a particular object in the application as shown in table 3.

TABLE 3: TIME (IN SECONDS) AFTER LATENCY FOR DIFFERENT ITERATIONS USING MANUAL

| S. No. | Average Time | Latency Time | Total Time in sec. |
|---|---|---|---|
| 1 | 13.26961538 | 5 | 18.26961538 |
| 2 | 22.30076923 | 5 | 27.30076923 |
| 3 | 35.20961538 | 5 | 40.20961538 |
| 4 | 49.07423077 | 5 | 54.07423077 |
| 5 | 70.12192308 | 5 | 75.12192308 |
| 6 | 81.1396 | 5 | 86.1396 |
| 7 | 91.71192308 | 5 | 96.71192308 |
| 8 | 101.9769231 | 5 | 106.9769231 |
| 9 | 112.9644615 | 5 | 117.9644615 |
| 10 | 125.1576923 | 5 | 130.1576923 |

And finally we add this latency time in the average time taken for each iteration by particular subject which give us the total time span for a particular iteration.

The same task was achieved using TLP automation approach and corresponding results obtained are shown in Table 4.

TABLE 4: TIME (IN SECONDS) FOR DIFFERENT ITERATIONS USING TLP

| Module name | Iteration | Time (sec) |
|---|---|---|
| | | |
| Sign in -Sign-out | 1 | 10.4687 |
| | 2 | 19.3863 |
| | 3 | 31.7996 |
| | 4 | 42.106 |
| | 5 | 53.5829 |
| | 6 | 65.5924 |
| | 7 | 76.8963 |
| | 8 | 88.3092 |
| | 9 | 100.7638 |
| | 10 | 111.5791 |

The total time taken in each approach is compared in table 5.

TABLE 5: MANUAL VS AUTOMATION TIME USING TLP APPROACH

| S. No. | Time in Sec. (Manual) | Time in Sec. (TLP) |
|---|---|---|
| 1 | 18.26961538 | 10.4687 |
| 2 | 27.30076923 | 19.3863 |
| 3 | 40.20961538 | 31.7996 |
| 4 | 54.07423077 | 42.106 |
| 5 | 75.12192308 | 53.5829 |
| 6 | 86.1396 | 65.5924 |
| 7 | 96.71192308 | 76.8963 |
| 8 | 106.9769231 | 88.3092 |
| 9 | 117.9644615 | 100.7638 |
| 10 | 130.1576923 | 111.5791 |

Comparisons of these two approaches is also shown graphically in figure 2, which clearly indicates a significant time saving in testing for just small number of iterations .In real time scenarios where the number of iteration are very high for a particular module we can save a huge mount of time value using TLP approach.

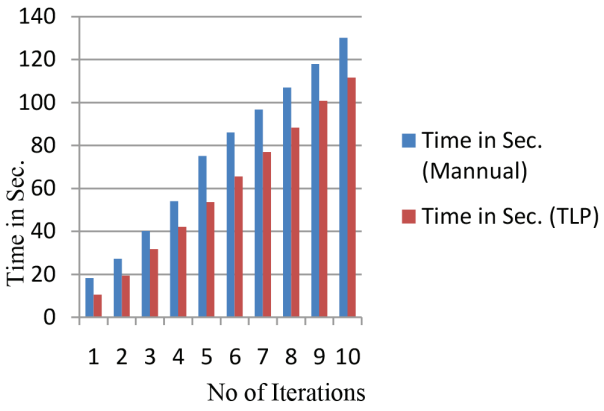



FIGURE 2: MANUAL TESTING TIME VS. AUTOMATION TIME USING TLP APPROACH

If we need to test a particular module say M2 in a Web application then it might be possible that we need to go through a particular module say M1 again and again. So it is very necessary that each time we test a module M2 the consistency of M1 is maintained. To check this consistency we need to make regression as a fundamental activity in our testing process.

## V. CONCLUSION AND FUTURE SCOPE

In this paper we have proposed TLP as a new approach to save a level significance amount of time during testing. One major problem that we face while using this ap-

proach was the time consume in developing scripts and a very high programming skill requirement. In the long term when number of modules in the application are high and we need to go through a particular module to check the other modules then this consumption of time in script development become less important than the overall performance achievement of time that could not be possible with manual testing approach. In future we can extend this work for proofing the concept of reusability of scripts, maintainability of test scripts and automatically managing the overall testing infrastructure which currently requires lot of human intervention.

## REFERENCES


[1] S. Pressman, Software Engineering: A Practitioner's Approach (Sixth, International ed.), McGraw-Hill Education Pressman, p. 388, 2005.

[2] Y. Singh, Software Testing ,Cambridge University Press ,pp-63, 2012.

[3] Keane.Inc/WP_ROTA,2006-11,pp-4,Roi On Test automation, unpublished.

[4] P.MCMinn " Serach Based Software Test Data Generation : A Survey ", Softwatre Testing , Verification and Reiability , Vol 14,No.. 2 ,pp. 105-156, June 2004.

[5] J. Bicevskis, J. Borzovs, U. Straujums, A. Zarins, and E. Miller, SMOTL-A system to construct samples for data processing pro- gram debugging," IEEE Trans. Software Eng., vol. SE-5, no. 1, pp. 60-66,Jan. 1979.

[6] S.D. Hendrick, et. al., "Market Analysis: Worldwide Distributed Automated Software Quality Tools 2005-2009 Forecast and 2004 Vendor Shares", IDC. July 2005.

[7] R. Boyer, B. Elspas, and K. Levitt. "SELECT-A formal system for testing and debugging programs by symbolic execution," SIGPLAN Notices, Vol. 10, No. 6, pp. 234-245. June 1975.

[8] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes."Generating finite state machines from abstract state machines", In ISSTA, 2002.

[9] S. Artzi, M. D. Ernst, A. Kiezun, C. Pacheco, and J. H.Perkins. "Finding the needles in the haystack: Generating legal test inputs for object-oriented programs", In 1st Workshop on Model-Based Testing and Object-Oriented, Oct, 2006.

[10] L. Clarke, "A system to generate test data and symbolically execute programs, " IEEE Trans. Sofrware Eng., vol. SE-2, no. 3, pp. 215- 222, Sept. 1976.

[11] C.Csallner and Y. Smaragdakis. JCrasher, "An automatic robustness tester for Java", In Software: Practice and Experience, Vol. 34, No. 11, pages 1025–1050, 2004.

[12] W. Howden, "Symbolic testing and the DISSECT symbolic evalua- tion system," IEEE Trans. Software Eng., vol. SE-4, no. 4, pp. 266- 278. 1977.

[13] I. Ciupa and A. Leitner. "Automatic testing based on design by contract", In Proceedings of Net.ObjectDays, 2005.

[14] C. Ramamoorthy, S. Ho, and W. Chen, "On the automated genera- tion of program test data," IEEE Trans. Software Eng., vol. SE-2, no. 4. PD. 293-300, Dec. 1976.

[15] M. Jorde, S. Elbaum, and M. B. Dwyer, "Increasing test granularity by aggregating unit tests", In ASE, 2008.

[16] C. D. Turner and D. J. Robson. "The state-based testing ofobject-oriented programs", In ICSM, 1993.

[17] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. "Ball.Feedback-directed random test generation", In ICSE, 2007.

[18] S. Thummalapenta, T. Xie, N. Tillmann, P. de Halleux, and W. Schulte.

"MSeqGen: Object-oriented unit-test generation via mining source code", In ESEC/FSE, 2009.

[19] http://sourceforge.net/projects/vtigercrm/files/vtiger%20CRM%20Release%20Archive/vtiger%20CRM%205/


## ABOUT THE AUTHORS

**Mukesh Mann** received his M.Tech in Information and Communication Technology from School of ICT, Gautam Buddha University, India. He did his B.Tech in Computer Science and Engineering from Kurukshetra University, Haryana,India. .His area of research is Software Engineering and Automated Software Testing.

**Om Prakash Sangwan** received his PhD and M.Tech in Computer Science & Engineering from Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India. His area of research is Software Engineering and Soft Computing. He is life member of Computer Society of India. Presently he is working Gautam Buddha University, Greater Noida(UP),India.