ISSN: 2321-7782 (Online)
Impact Factor: 6.047

Volume 4, Issue 7, July 2016

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Survey Paper / Case Study

Available online at: www.ijarcsms.com

Software Design Pattern Static Validation Using Cyclomatic Complexity and UML Approach

Saurabh Srivastav[1]
Research scholar, SGVU,
Jaipur, India

Sapan Gupta[2]
Asst. Professor, SGVU,
Jaipur, India

Abstract: ***Software design is considered to the versatile design on which the whole of the functionality of the software depends. Softwares considered being one of the most critically important commodities of our lives. Software contributes to a fast, reliable, less effort taking and high computation environment in which the output and computation totally and automatically depends on the machines. Cyclomatic complexity is a software measurement tool or rather software metric which defines the comp1exity of the software whi1e processing or running the program. Cyclomatic complexity being one of the superlative metrics for the analysis of the softwares as it tells about total executable path followed while the processing of the program. Overall, it deals with the functional behavior of the system. The above mentioned formula can be implemented in a CFG. UML is an acronym or abbreviation for "Unified Modeling language". It is used for the modeling the software and analysis purpose. It is basically a pictorial language which tells about the different attributes of the softwares. It is represented by the different notation and associations. These notations and attributes combined together in a unique fashion to form a different diagram. We deal into the development of the System in the UML.***

This paper deals with the analytics of the software in terms of UML and Cyclomatic complexity.

Keywords: Cyclomatic Complexity, UML, Software, Metric, CFG.

I. INTRODUCTION

Computers and Software, no doubt, are one of the biggest innovations in the history of mankind. Software contributes to a fast, reliable, less effort taking and high computation environment in which the output and computation totally and automatically depends on the machines. Software Engineering is considered to be the science of Development and Maintenance of the software. We oftenly use different types of SDLC cycle for the development purpose.

- Waterfall model.
- Spiral model.
- Prototype model.
- Incremental model.
- RAD model etc.

The software is required to fulfill the different criteria as per the software quality assurance (SQA). Software quality is the feature of the software which tells about the aptness of the given software and helps to generate the software project with higher precision and efficiency.

Quality control is the sort of iterative and repeated mechanism to analyse the present behavior of the system to the expected behavior/outcome.

Cyclomatic complexity= E- N+2;

Where, E is total no of edges in CFG, N is total no of nodes in CFG. CFG is the acronym used for the Control Flow Graph. Sometimes, the whole of the single module is considered as the node in the cyclomatic complexity and other level passing statements as the edges.

UML is an acronym or abbreviation for "Unified Modeling language". It is used for the modeling the software and analysis purpose. It is basically a pictorial language which tells about the different attributes of the softwares. It is represented by the different notation and associations. These notations and attributes combined together in a unique fashion to form a different diagram.

There are generally 5 major types of the UML diagrams, which are the following:

- Use case diagram
- Class diagram
- Sequence diagram
- State chart diagram
- Activity diagram

II. LITERATURE SURVEY

The past has been filled with a lot of the diverse research in the subject for the making us capable enough for the use of the software with higher functionality. This section deals with the researchers along with the their research work done in the past.

Tom Mens has already observed the evolution pattern of the software. He proposed a formal definition of evolution complexity to precisely quantify the cost of adjusting a particular implementation to a change ("shift") in the requirements [1]. As a validation, we show that this definition formalizes intuition about the evolvability of design patterns.

Noraini M. discussing the consistencies of the complexity of the code supposedly produced by students based on the programming questions found in the programming assessment throughout a semester of studies [2].

Christian Liebchen put forward an unitary classification accommodating these three classes and further including the following four relevant classes: 2-bases (or planar bases), weakly fundamental cycle bases, totally uni-modular cycle bases, and integral cycle bases [3].

A. yadav proposed role of Encapsulation in improving reliability of an object oriented design [4]. It is found that higher encapsulation in design increases reliability and decreases complexity of design.

Bin lei proposed the methodology for checking the consistency and robustness testing of the software modules [5].

Lei mi et. al. proposed method of specification mutation testing is effective to detect the

inconsistency in the specification and the program [6].

Joseph kempka proposed two different local searches that may be used in conjunction with the AVM, Geometric and Lattice Search [7]. A theoretical runtime analysis proves that under certain conditions, the use of these searches results in better performance compared to the original AVM.

R. peng et. al. studied the fault detection process (FDP) and fault correction process (FCP) with the incorporation of testing effort function and imperfect debugging. In order to ensure high reliability, it is essential for software to undergo a testing phase, during which faults can be detected and corrected by debuggers [8]. The testing resource allocation during this phase,

which is usually depicted by the testing effort function, considerably influences not only the fault detection rate but also the time to correct a detected fault.

III. Proposed Methodology

The above mentioned topic viz. "Software design pattern Static-validation using cyclomatic complexity and UML approach" deals with the testing the programs efficiency with the use of the pattern of the given measuring tool. These tools can be implemented as the metric for the softwares. We will be verifying the flow label and the flow control in the coding of the softwares.

Our expected pattern for the carrying out of the research is finding out the anomalies and discrepant section of the codes which hamper the space and time complexity and to find the feasible solution to the rising problem.

There are a lot of tools and compilers like Matlab, QTP, The bugginie etc. available to perform the tasks relating to analysis of the code and check for the fault and defect in the code. Answer in the terms of their enhancing efficiency lies there in the internal structure of the program. Our major focus is to verify the code using the pictorial method and map the problem domain to the solution domain.

Our proposed methodology is to statically check the software in terms of their internal layout structure and answer to the question for the productivity constraint along with the reliability issues.

IV. Simulation and Result

Apart from the above mentioned terminologies, the other key terms used in the projects are mentioned below:

- KLOC: It stands for the kilo-lines of codes or more precisely, l000 lines of code. It is the entity to measure the size of any code in the terms of the number of line. We get to know about the dimensions of the codes.
- Function point: It is abbreviated as the F.P. It is the entity which is oftenly used for acquaint the size of the software in the terms of functionality.

The function point can be compared with that of the size of the code (in LOC)

LOC= *Language factor * Function point*

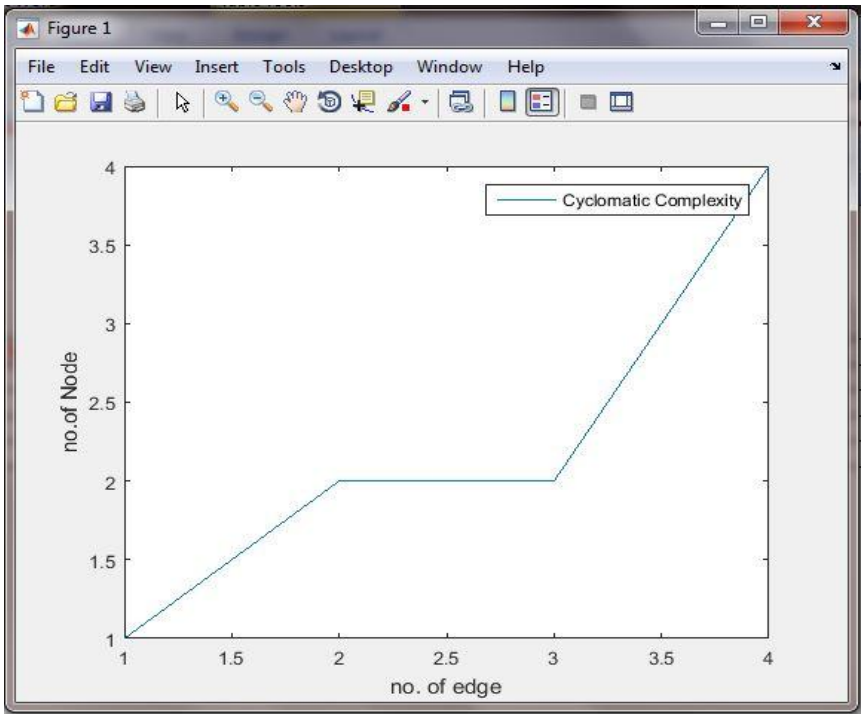

Fig: Showing graph for deviation in cyclomatic complexity

SAMPLE	SIZE	EDGES	NODES	CC
1	11	4	5	1
2	25	8	8	2
3	14	7	7	2
4	28	13	11	4

Fig: Chart showing data-set obtained from different samples

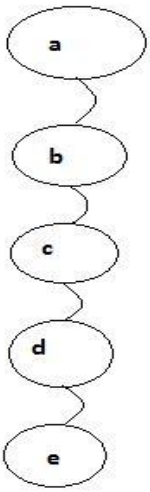

Fig: Showing CFG for sample 1

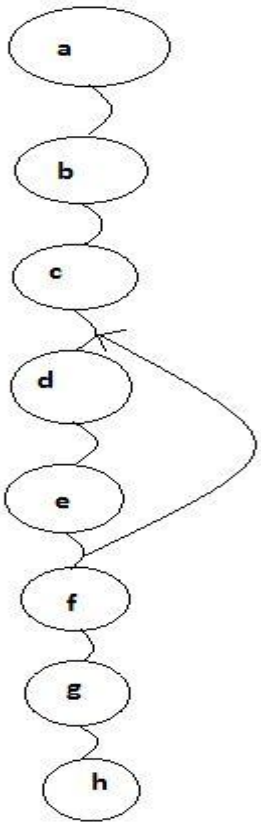

Fig: Showing CFG for sample 2

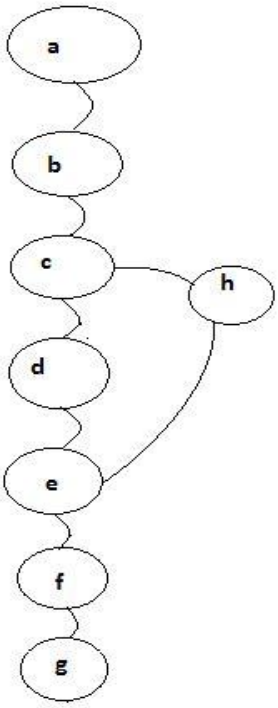

Fig: Showing CFG for sample 3

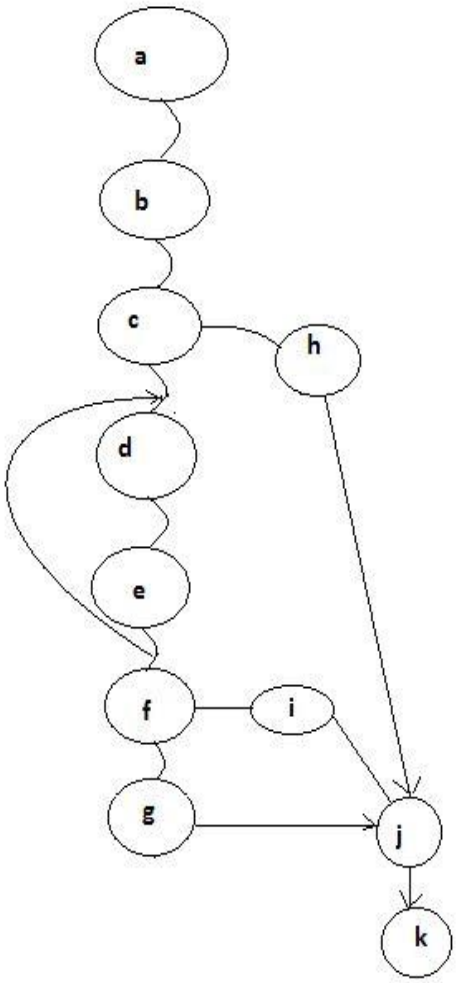

Fig: Showing CFG for sample 4

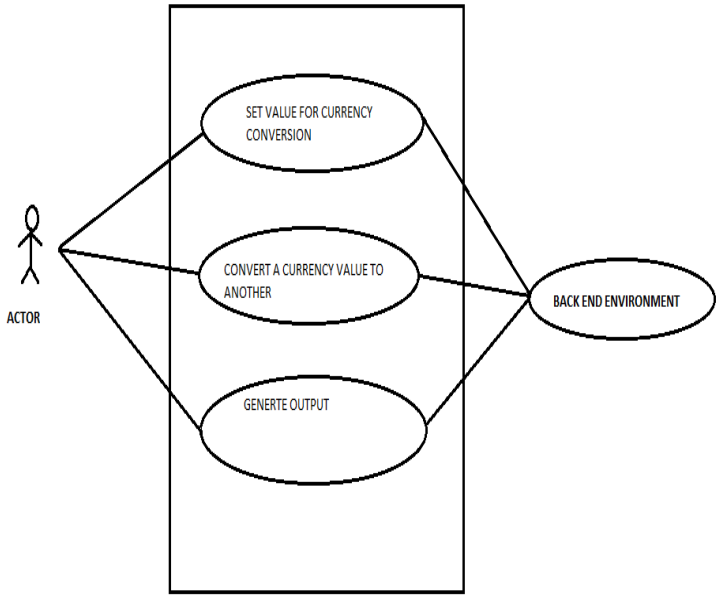

fig: UML Use case for Sample 5

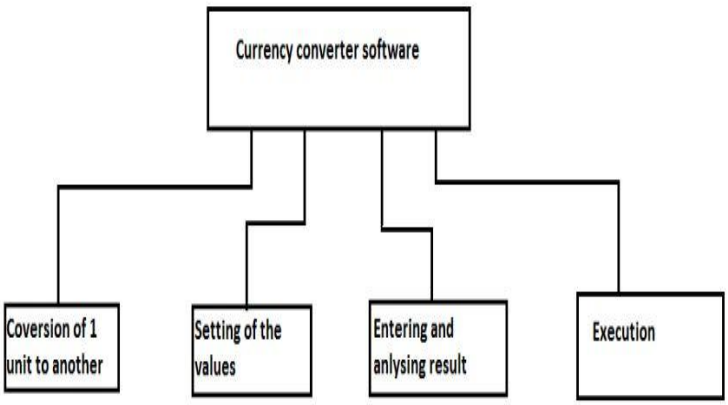

fig: UML class for Sample 5

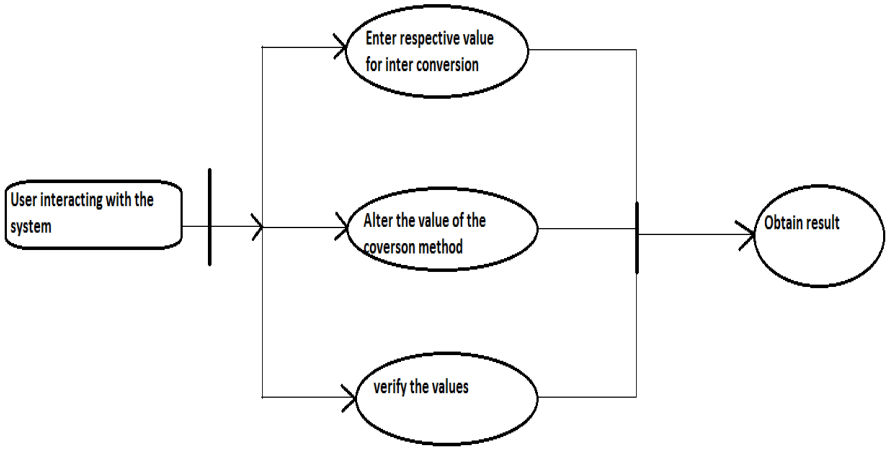

fig: UML class for Sample 5

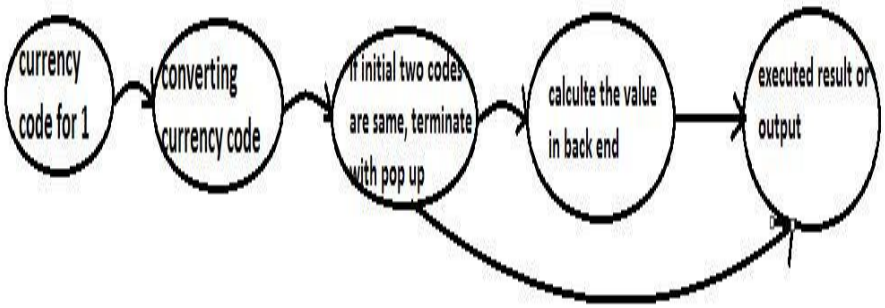

fig: UML class for Sample 5

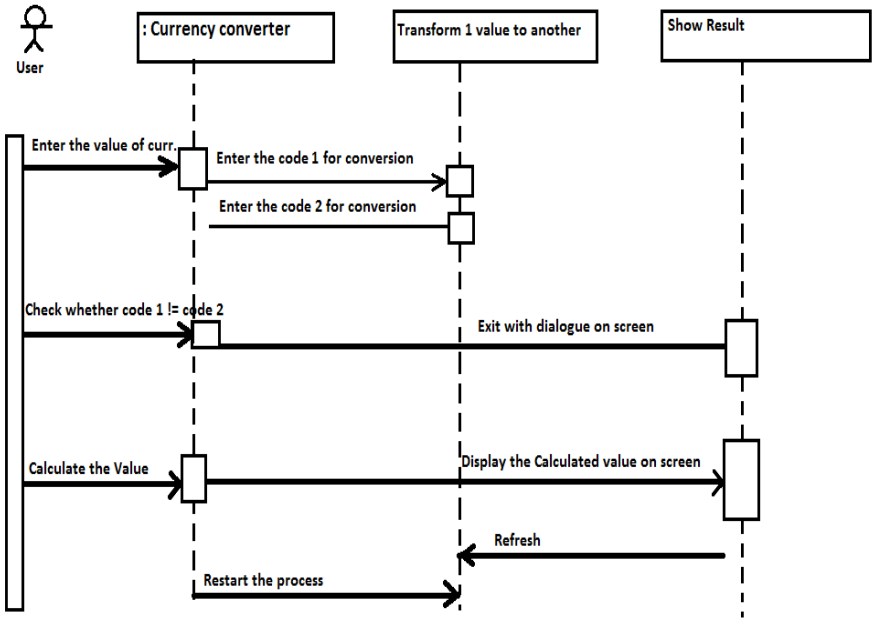

Fig: UML for sample 5

V. CONCLUSION

The cyclomatic complexity tells about the number of executable path from the starting node to the final node in the CFG. Size can be measured in the line of codes. If the size of the given project is larger then, it takes much larger amount of the time for the properly running of the module of the code.

The above result shows that the software can be very predominantly be shown in the form of the above mentioned attributes. From the above results we can infer the following data regarding the cyclomatic complexity:

If the cyclomatic complexity of given software is in the range of 0-3:

This means that the software contains less no of the loops, iterations, if-else statements etc. There is least amount of the labor required for the traversal of the software codes from the beginning to the end of the software. The software can be considered to be of the primitive type and the value of the cyclomatic complexity reflects the number of the of independent path to be followed for the execution.

If the software is having the cyclomatic complexity value in the range of 4-8:

The software may comprise of the bit of the difficulty in the execution of the program. The software may be the having few of the iterative statements, if else ladders, and some elemental concept regarding the loops. The amount of CC conveys the quantitative view of the execution path from initial node to the final node.

If the software is having the cyclomatic complexity value > 8:

The software consists of lots of the "if else if" and the iterations. The software may be having the lots of difficulty in the analysis of the code because of the higher amount of the flow paths. It is also true that the total number of the functionality also increases and the Lines of code also increases with the increase in the Cyclomatic complexity.

From the above analysis, we can deduce that :

- $cc \propto Functionality\ of\ the\ software$
- **CC doesn't depend on size in LOC**
- **CC can be directly calculated by number of iteration and switch statements**

Here, CC represents the cyclomatic complexity.

In (A) says that the cyclomatic complexity increases with the increase in the functionality of the software (cyclomatic complexity is directly proportion to the functionality.)

In (B) says that the cyclomatic complexity have no effect with the increase in the Size of the software and vice versa

Analysis of the UML diagrams:

UML diagrams are the blue-prints of the softwares. These are of the variety of types viz. class diagram, activity diagram, sequence diagram, use case diagrams etc. These diagrams represent the different mode of the functional behavior of the software.

The Use case diagram represents the Actor (User) interaction of the system. If the functionality of the software is higher then, it must be reflected in the environment module of the software and show the facts and the Actors interaction with these.

This can also be visualized by using the Graph theory. The Use case also supports the cyclomatic complexity in the terms of the functionality. The sequence diagram represents the sequential step for the execution of the program. This explains the total path followed during the traversal of the compiler during the program execution. While dealing with it qualitatively, we can say it has the modular approach during the execution of the program.

The activity diagram in the UML describes the list of the activity a system can be supposedly be doing. The software can be modelled on the basis of the activity it can do. The activity is representation of the system of works and the steps followed are bonded together.

Class diagram represents the internal structure of the system in a way that the system is amalgamated form of the different class or the module. It has a root node or the parental node in which the different classes or the modules are attached.

The State chart diagram of the UML represents current state as well as the transition state of the system. The state chart diagram is used to represent the different state of the given software. The current state is also called the present state and the upcoming or the state to be altered shortly is termed as the transition state.

These UMLs are the representational unit which can tell us to implement the system through the representational method and try to resolve any defect in the system. It is also highly indispensable unit in the analysis of the software system. It is thus, truly called the Blue Print of the software.

We can also establish the relationship between the Cyclomatic-complexity and the UML as the higher the CC values, larger will have the UML attributes.

The higher value of the CC says it has different or the multiple flow paths. Every path brings about the different functionality to the system. So, while depicting it to the UMLs, the attribute of the UML will increase.

References

1. Mens T.- On the evaluation complexity of design patterns – Elsevier, Electronic Notes in Theoretical Computer Science 127 (2005) 147–163.
2. Noraini M. et.al.- The Use of Cyclomatic Complexity Metrics in Programming Performance's Assessment, Elsevier- Procedia - Social and Behavioral Sciences 90 (2013) 497 – 503.
3. Liebchen C. et. al.- Classes of cycle bases, Elsevier, Discrete Applied Mathematics 155 (2007) 337 – 355.
4. Yadav A. et.al.- Development of Encapsulated class complexity metric, Elsevier, Procedia Technology 4 (2012) 754 – 760.
5. Lei B. et.al.- Robustness testing for the software components, Elsevier, Science of Computer Programming 75 (2010) 879-897.
6. Mi L. et.al.- A method for specification mutation testing based on UML state diagram for consistency checking, Elsevier, Procedia Engineering 15 (2011) 110 – 114
7. Kempka J. et.al.- Design and analysis of different alternating variable searches for search-based software testing, Elsevier, Theoretical computer science. (2015), http://dx.doi.org/10.1016/j.tcs.2014.12.009
8. Peng R. et. al.- Testing effort dependent software reliability model for imperfect debugging process considering both detection and correlation, Elsevier, Reliability Engineering and System Safety 126(2014)37–43
9. Lee J.- Dynamic reverse code generation for backward execution, Elsevier, Electronic Notes in Theoretical Computer Science 174 (2007) 37–54
10. Mossoly M.- Global Projects: A Conceptual Review on Execution Attitude in Multinational Corporations, Elsevier, Procedia - Social and Behavioral Sciences 194 (2015) 125 – 133.
11. Gad Al Rab W.- Analyzing the impact of the UMLs relations on the word-sense Disambiguation accuracy, Elsevier, Procedia Computer Science 21 (2013) 295 – 301.
12. Combrenero M.- Modeling distributed service systems with resources using UML, Elsevier, Procedia Computer Science 18 (2013) 140 – 148
13. Lugato D. et. al.- Validation and automatic test generation on UML models: the AGATHA approach, Elsevier, Electronics Notes in Theoretical Computer Science 66 No.2 (2002) (URL: htpp://www.elsevier.nl/locate/encts/volume66.html 16 pages)
14. Ziemann P.et.al.- Coherently explaining UML state chart and collaboration diagrams by Graph Transformation, Elsevier, Electronic Notes in Theoretical Computer Science 130 (2005) 263–280
15. Lano K.et.al.- Refinement pattern for UML, Elsevier, Electronic Notes in Theoretical Computer Science 137 (2005) 131–149.
16. Lima V. et.al.- Formal verification and validation of UML sequence diagrams source and destination of messages, Elsevier, Electronic Notes in Theoretical Computer Science 254 (2009) 143–160
17. Namita K. et.al.- Test case generation and optimization using UML models and genetic algorithm, Elsevier, Procedia Computer Science 57 (2015) 996 – 1004.
18. Ranjita KS. et. al.- Test case design using slicing of UML interaction diagram, Elsevier, Procedia Technology 6 (2012) 136 – 144.
19. Zhu LZ. et.al.- Automatic conversion from UML to CPN for software performance evaluation, Elsevier, Procedia Engineering 29 (2012) 2682 – 2686.
20. Yashmina R. et.al.- A framework for modeling and analysis UML Activity diagram using graph transformation, Elsevier, Procedia Computer Science 56 (2015) 612 – 617.
21. Ali hanzala K. et.al. Consistency of UML class, object and state chart diagrams using ontology reasoners, Elsevier, http://dx.doi.org/10.1016/j.jvlc.2014.11.006-1045-926X/& 2014
22. Renu G. et. al.- Foundation UML behavioral specification with java, Elsevier, Procedia Computer Science 46 (2015) 941 – 948.

AUTHOR(S) PROFILE

Saurabh Srivastava, received the B.tech degree in computer science and M.tech degrees in Information and Communication from suresh gyan vihar university in 2014 and 2016 respectively. During 2010-2015

