GENETIC ALGORITHM BASED APPROACH FOR FINDING FAULTY MODULES IN OPEN SOURCE SOFTWARE SYSTEMS

Aditi Puri[1] and Harshpreet Singh[2]

[1]Department of Computer Science and Engineering, Lovely Professional University, Punjab, India
[2]Assistant Professor, Department of Computer Science, Lovely Professional University, Punjab, India

ABSTRACT

Computer program produces an incorrect or unexpected result or behaves in haphazard way then there is an error in that computer program. In order to improve the software quality, prediction of faulty modules is necessary. Various Metric suites and techniques are available to predict the modules which are critical and likely to be fault prone. Genetic Algorithm is a problem solving algorithm. It uses genetics as its model of problem solving. It's a search technique to find approximate solutions to optimization and search problems.Genetic algorithm is applied for solving the problem of faulty module prediction and as well as for finding the most important attribute for fault occurrence. In order to perform the analysis, performance validation of the Genetic Algorithm using open source software jEdit is done. The results are measured in terms Accuracy and Error in predicting by calculating probability of detection and probability of false Alarms

KEYWORDS

Genetic Algorithm, software metrics, fault, faulty modules, fault proneness

1. INTRODUCTION

A fault prone module is one in which the number of faults are higher than a selected threshold. Making any application or software 100% defects free is the need of the hour and the greatest challenge being faced by software industry. Fault proneness of software depends on the faults it contains and is directly proportional to its measurable attributes and testing. Detection of fault-prone modules of software helps the experts to concentrate more on the development work. Fault proneness can be predicted by classifying software modules into categories of faulty and non-faulty modules at an early stage of development.

Abundant research has been done on the software fault prediction techniques and software metrics were considered for the classification and evaluation of the data.

Predicting fault prone modules adds up to the software quality assurance. Many algorithms have been used to predict the fault proneness including the hierarchical and k-means clustering algorithms. Some work has also been done using Bayes Network Classification Algorithm and spamfiltering technique for finding fault prone software modules.Support vector machine (SVM) and module dependency graphs (MDGs) have also proved helpful for predicting the fault proneness.

DOI : 10.5121/ijcses.2014.5303

Software metrics has been widely used and has proven to be one of the critical attribute to measure the fault proneness of applications and software. Chidamber & Kemerer metrics suite [1] includes the various product metrics and a quality oriented extension. Martin's metrics [10] and Henderson-Sellers metrics [9] are also there.

In the first section of this paper a brief introduction of the topic is described including the use of metrics and the historic methods used in fault findings. Second section of the paper gives the literature review.

2. RELATED WORK

Parvinder S. Sandhu et al.[2] presented K-means clustering approach to cluster the modules into different categories of faulty and non-faulty modules and thereafter empirically validated them. Chidamber & Kemerer metrics suite [1] was used to divide the data into categories. To find the attributes that are important for predicting Chi-squared ranking filter was applied subsequently to reduce the data. The reduced attributes is then given as input to the K-means clustering algorithm that divides data into two or more clusters depending upon that whether they are fault free or fault prone.

Ritika et al. [3]empirically evaluated the performance of SVM technique in predicting fault-prone classes using open source software. SVM structure was generated from the faulty data in MATLAB environment and then evaluated for the JEdit dataset. The performance of the system was evaluated by calculating the accuracy%, the percentage of the predicted values that matches with the obtained values.

D. Doval et al. [4]used MDGs to represent the structure of complex software systems in understandable form. In MDGs, the system’s modules (e.g., files, classes) are represented as nodes and their relationships (e.g., function calls, inheritance relationships) as directed edges that connect the nodes. Genetic Algorithm is then used to efficiently partition the modules to form MDG and thus clustered and classified the modules as fault prone or fault free.

Dr. Parvinder S. Sandhu [5] evaluated the fault proneness of modules in open source software system JEdit using k-NN clustering algorithm based on Chidamber & Kemerer metrics suite [1] .The DIT, CBO, RFC, NPM and LOC metrics were selected and the predicted model so developed was applied to 274 classes of the dataset. The percentage of Accuracy as 82.84, Probability of Detection, Probability of False alarm was recorded for the fault dataset.

Simranjit Kaur et al. [6] investigated the accuracy of the fault prediction of the software modules using Hierarchical based clustering. Structural code and design attributes of the software systems was found and suitable metric values were selected for representing the dataset. Further the metrics values were refined and normalized. Hierarchical Clustering algorithm was then used to classify the software components into two categories of faulty and fault-free systems. Both agglomerative as well as divisive approach was used for the predicting the accuracy of 85.12 and finally confusion metrics was used to predict the outcome.

Aarti Mahajan et al. [7] predicted the software quality by investigating the capabilities of Bayes Network Classification Algorithm using open source software JEdit. Structural code and design attributes of the software systems was found and suitable metric values were selected for representing the dataset. Further the metrics values were refined and normalized. The dataset so obtained was given as input to the Bayes Network Classification Approach.Mean Absolute Error and Root Mean-Squared Error both were calculated. Finally confusion metrics with an accuracy of 70.8 was used to predict the outcome.

Osamu Mizuno et al. [8] applied the spam filtering technique for finding fault prone software modules. Each software module was considered as an e-mail message and either of fault-prone (FP) category or not-fault prone (NFP) category. New modules were classified as FP or NFP by studying the characteristics of existing FP and NFP modules using spam filter. Instead of metric measure, source code was used and the text classification is made on the basis of the past history of the development. FPFinder, prototype tool was used to track bugs. The result of the experiment concluded that spam filter approach can classify more than 75% of the modules correctly.

Table 1. Overview of the techniques/methods followed in literature review

Sno.	**Technique Name**	**Methodology followed**	**Accuracy percentage**	**Limitations**
1.	K-means clustering approach [2]	1. Categorized the dataset into metrics. 2. Used chi-squared to filter out the important metrics. 3. Made two clusters of faulty & non-faulty modules using k-means	62.4	Most important attributes for fault prediction was not found.
2.	Support vector machine (SVM) technique [3]	1. SVM structure was generated from the faulty data in MATLAB 7.4 environment **2.** Structure evaluated for the JEdit dataset.	78.1022	Most important attributes for fault prediction was not found.
3.	Module dependency graphs (MDGs) [4]	1. MDGs were used to represent the structure of complex software systems in understandable form. 2. Genetic Algorithm is then used to efficiently partition the modules to form MDG	Not calculated	Does not provide a mechanism to integrate a designer’s knowledge of a system into the automatic clustering process
4.	k-NN clustering algorithm [5]	1. Metrics was selected to form a model. 2. Predicted model so developed was applied to 274 classes of the dataset.	82.84	Similar studies were not carried out to confirm the acceptability of the prediction.
5.	Hierarchical based clustering [6]	1. Hierarchical based clustering is used. 2. Components were divided into two categories of faulty and fault-free.	85.12	Most important attribute for fault prediction was not found.

6.	Bayes Network classification algorithm [7]	1. Suitable metric values were selected for representing the dataset. 2. Bayes network classification is used. 3. Mean absolute error and root mean-squared error were calculated.	70.8	Most important attribute for fault prediction was not found.
7.	Spam filtering technique [8]	1. Each module considered as an e-mail message 2. New modules were classified as FP and NFP on basis of past history 3. FPFinder tool was used to track bugs.	Classified more than 75% of modules correctly.	1. Some modules were misclassified. 2. No practical advantage.

3. PROPOSED APPROACH

Step 1: Raw data is collected in the form of structural codes, source code of open source system and design attributes.

Step 2: Evaluation of the data obtained in step 1 is done on Chidamber & Kemerer metrics suite [1] and out of those some metrics are chosen to illustrate the principal design. They are-

- **Coupling between Objects (CBO)**, CBO for a class is the count of the total number of other classes to which it is coupled and vice versa.
- **Lack of Cohesion (LCOM)**, it is the measurement of the dissimilarity of methods in a class. It is measured by looking at the instance variable or attributes used by methods in a class.
- **Number of Children (NOC)**, The NOC is the number of immediate subclasses of a class in a hierarchy.
- **Depth of inheritance (DOI)**, the depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes
- **Weighted Methods per Class (WMC)**, it is the count of sum of complexities of all methods in a class. Consider a class P1, with Methods M1…… Mr that are defined in the class. Let. C1, C2....Cr be the complexity of the methods
  WMC= ΣC_i where i=1 to r
  If all the methods complexities are considered to be unity, then WMC = r the number of methods in the class.
- **Response for a class (RFC)**, it is defined as set of methods that can be potentially executed in response to a message received by an object of that class.
  It is given by RFC= |RS|, where RS, the response set of the class
  RS = M_i U all j{R_{ij}}
- **Number of Public Methods(NPM)**, it is the count of number of Public methods in a class
- **Lines of Code (LOC)**, It is the count of the total number of lines in the text of the source code excluding comment lines.

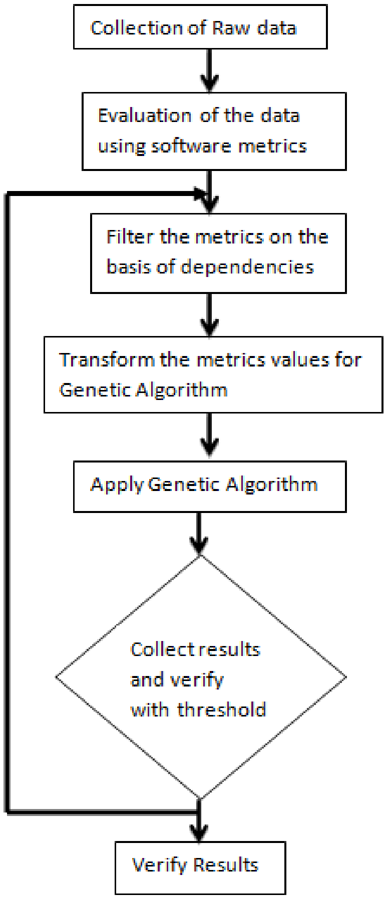

Collection of Raw data
Evaluation of the data using software metrics
Filter the metrics on the basis of dependencies
Transform the metrics values for Genetic Algorithm
Apply Genetic Algorithm
Collect results and verify with threshold
Verify Results

Figure 1: Flowchart of methodology followed

Step 3: The data collected in step 2 is now filteredto find the most important metrics for the fault prediction.

Step 4: Further the reduced or selected attributes are given as input to Genetic Algorithm.

Step5: The performance of the algorithm is evaluated on the basis of confusion matrix. This allows us to have more detailed analysis than mere having correct guesses. The following sets of attributes are being used to refine our results:

• **Probability of Detection (PD)**, it is defined as the probability of correct classification of a module that contains a fault.

PD = TP / (TP + FN)

• **Probability of False Alarms (PF)** is defined as the ratio of false positives to all non-defect modules. PF = FP / (FP + TN)

- **Recall or true positive rate (TP)** it is the proportion of faulty modules that are correctly identified as faulty.
- **False positive rate (FP)**it is the proportion of non-faulty modules that are incorrectly classified as faulty.
- **True negative rate (TN)**it is defined as the proportion of non-faulty cases that are classified correctly as non-faulty.
- **False negative rate (FN)** is the proportion of faulty modules that are incorrectly classified as non-faulty.

Table 2. Confusion matrix of prediction outcomes

Predicted	Real Data		
		Fault	No fault
	Fault	TP	FP
	No Fault	FN	TN

It can be clearly concluded as probability of detection or PD should be maximum and probability of false alarm or PF should be minimum.

The accuracy (AC) is the proportion of the total number of predictions that were correct. It is determined using the equation:

$$AC = TP+TN /TP+FP+FN+TN$$

The error in the calculation of the values of PD and PF is calculated as:

$$Error= FN+FP/ TP+FP+FN+TN$$

4. RESULTS AND FINDINGS

Step 1: The data for the research is taken as the source code of the open source jEdit project. The source code of the two versions of jEdit is collected and analyzed and fed as input into JArchitect, static analysis tool for JAVA code. This tool supports a large number of software code metrics. It also allows visualization of dependencies using directed graphs and dependency matrix. Code base snapshots comparison, validation of architectural and quality rules are also performed by this tool.

Step 2: Using JArchitect_4.0.0.8041 analysis of jEdit version 4.5pre 1 is done and the following values are calculated-

- Public methods on classes give the total number of public methods in the classes of the software.
- Cyclomatic Complexity (CC) is calculated for calculating the Weighted Methods per Class (WMC) which is the count of sum of complexities of all methods in a class.
- Lines of code (LOC) is calculated as the count of number of lines in the text of the source code of jEdit version 4.5pre 1 and jEdit version 5.0.1excluding comment lines.
- Efferent Coupling is calculated both at the method level and at the package level. Efferent Coupling for a particular method is the number of methods that method directly depends on. Efferent Coupling for a particular package is the number of packages it directly depends on. It is used to calculate Coupling between Objects (CBO) metric.
- Relational Cohesion (H)gives the average number of internal relationships per type. Let R be the number of type relationships that are internal to the project and N be the number of types within the project then
  H=(R+1)/N Calculating H gives the value of Lack of Cohesion (LCOM) metric.
- The Depth of Inheritance Tree for a class or a structure is the number of its base classes (including the System.Object class thus DIT >= 1)

Table 3. Number of occurrences of the property in jEdit version 4.5pre 1

Property	**Occurrences**
Public methods on classes	1123 classes
Cyclomatic complexity(cc)	8114 methods
LOC	54270
Efferent Coupling	459
Relational Cohesion	4.47
Properties on interfaces	63 interfaces
Methods on interfaces	63 interfaces
Arguments on methods on interfaces	177 methods
Public properties on classes	1123 classes
Arguments on public methods on classes	6594 methods
BC instructions in non-abstract methods	8114 methods

Table 4. Maximum value of the metrics that exists in jEdit version 4.5pre 1

Property	**Max**
Public methods on classes	241 public methods
Cyclomatic complexity(cc)	309 cyclomatic complexity
LOC	NA
Efferent Coupling	NA
Relational Cohesion	NA
Properties on interfaces	0 properties
Methods on interfaces	17 methods
Arguments on methods on interfaces	8 arguments
Public properties on classes	0 public properties
Arguments on public methods on classes	12 arguments
BC instructions in non-abstract methods	1445 BC instructions

Table 5. Average values of the metrics in jEdit version 4.5pre 1

Property	**Avg**
Public methods on classes	5.87
Cyclomatic complexity(cc)	3.1
LOC	NA
Efferent Coupling	NA
Relational Cohesion	NA
Properties on interfaces	0
Methods on interfaces	2.81
Arguments on methods on interfaces	1.34
Public properties on classes	0
Arguments on public methods on classes	1.14
BC instructions in non-abstract methods	29.37

Table 6. Standard deviation of the metrics values in jEdit version 4.5pre 1

Property	StdDev
Public methods on classes	11.64
Cyclomatic complexity(cc)	7.56
LOC	NA
Efferent Coupling	NA
Relational Cohesion	NA
Properties on interfaces	0
Methods on interfaces	2.56
Arguments on methods on interfaces	1.53
Public properties on classes	0
Arguments on public methods on classes	1.26
BC instructions in non-abstract methods	56.78

Using JArchitect_4.0.0.8041 analysis of jEdit version 5.0.1 is done and the following values are calculated-

Table 7. Number of occurrences of the property in jEdit version 5.0.1

Property	Occurrences
Public methods on classes	46 classes
Cyclomatic complexity(cc)	299 methods
LOC	2774
Efferent Coupling	82
Relational Cohesion	2.51
Properties on interfaces	3 interfaces
Methods on interfaces	3 interfaces
Arguments on methods on interfaces	9 methods
Public properties on classes	46 classes
Arguments on public methods on classes	214 methods
BC instructions in non-abstract methods	299 methods

Table 8. Maximum value of the metrics that exists in jEdit version 5.0.1

Property	Max
Public methods on classes	32 public methods
Cyclomatic complexity(cc)	64 cyclomatic complexity
LOC	NA
Efferent Coupling	NA
Relational Cohesion	NA
Properties on interfaces	0 properties
Methods on interfaces	5 methods
Arguments on methods on interfaces	1 arguments
Public properties on classes	0 public properties
Arguments on public methods on classes	6 arguments
BC instructions in non-abstract methods	2052 BC instructions

Table 9. Average values of the metrics in jEdit version 5.0.1

Property	**Avg**
Public methods on classes	4.65
Cyclomatic complexity(cc)	2.95
LOC	NA
Efferent Coupling	NA
Relational Cohesion	NA
Properties on interfaces	0
Methods on interfaces	3
Arguments on methods on interfaces	0.78
Public properties on classes	0
Arguments on public methods on classes	1
BC instructions in non-abstract methods	51.21

Table 10. Standard deviation of the metrics values in jEdit version 5.0.1

Property	**StdDev**
Public methods on classes	5.56
Cyclomatic complexity(cc)	5.41
LOC	NA
Efferent Coupling	NA
Relational Cohesion	NA
Properties on interfaces	0
Methods on interfaces	1.63
Arguments on methods on interfaces	0.42
Public properties on classes	0
Arguments on public methods on classes	0.98
BC instructions in non-abstract methods	155.74

Step 3*:* The values of the metrics collected in the previous step is converted into binary form. This conversion is done on the basis of the threshold value of the metrics available.

- A good range for Relational Cohesion is 1.5 to 4.0. Projects where Relational Cohesion < 1.5 or Relational Cohesion > 4.0 might be problematic.
- Methods where CC is higher than 15 are hard to understand and maintain. Methods where CC is higher than 30 are extremely complex and should be split in smaller methods (except if they are automatically generated by a tool.)
- Types where Ce > 50 are the types that depends on too many other types. They are complex and have more than one responsibility.
- Methods where BC Instructions is higher than 100 are hard to understand and maintain. Methods where BC Instructions is higher than 200 are extremely complex
- Number of Methods > 20 might be hard to understand and maintain but there might be cases where it is relevant to have a high value for Number of Methods.

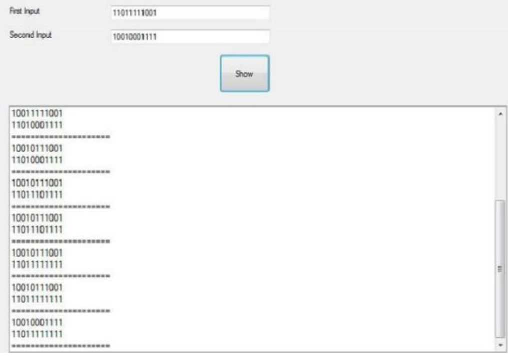
Table 11. Transformed values for jEdit version 4.5pre 1

Property	**Transformed value**
Public methods on classes	1
Cyclomatic complexity(cc)	1
LOC	0
Efferent Coupling	1
Relational Cohesion	1
Properties on interfaces	1
Methods on interfaces	1
Arguments on methods on interfaces	1
Public properties on classes	0
Arguments on public methods on classes	0
BC instructions in non-abstract methods	1

Table 12. Transformed values for jEdit version 5.0.1

Property	**Transformed value**
Public methods on classes	1
Cyclomatic complexity(cc)	0
LOC	0
Efferent Coupling	1
Relational Cohesion	0
Properties on interfaces	0
Methods on interfaces	0
Arguments on methods on interfaces	1
Public properties on classes	1
Arguments on public methods on classes	1
BC instructions in non-abstract methods	1

Step 4. Genetic Algorithm is applied on the above transformed values. Input to the Genetic Algorithm is the string of the transformed metric values. The result so obtained after the application of the algorithm is thus further analyzed for calculating the most important attribute for the fault prediction.



```
First Input     11011111001
Second Input    10010001111

10011111001
11010001111
====================
10010111001
11010001111
====================
10010111001
11011101111
====================
10010111001
11011101111
====================
10010111001
11011111111
====================
10010111001
11011111111
====================
10010001111
11011111111
====================
```

Figure 2: Output of the Genetic Algorithm

Genetic Algorithm applied on the metric values gives the public methods on Classes, Efferent Coupling, Arguments on methods and interfaces, public properties on classes, arguments on public methods on classes and BC instructions in non-abstract methods as the most important attributes in the fault prediction.

Step 5: The performance of the algorithm is evaluated using Confusion Matrix. The following sets of attributes are calculated as:

Table 13. Recorded Confusion matrix

		Real Data	
Predicted		**Faulty**	**Non-Faulty**
	Faulty	7	4
	Non-Faulty	1	5

The values of Probability of Detection (PD) and Probability of False Alarms (PF) are 0.875 and 0.44 respectively.The error and accuracy values are calculated and recorded as 0.294 and 0.705 respectively.

5. CONCLUSION

In this paper Genetic Algorithm is used to find critical classes and metrics that are fault prone.The proposed Genetic Algorithm technique shows high value of Probability of Detection (PD) i.e. 0.875 and low value of Probability of False Alarms (PF) i.e. 0.44.The error and accuracy values are calculated and recorded as 0.294 and 0.705 respectively. It is therefore concluded that Genetic algorithm can be used for object-oriented systems and is useful in predicting the fault prone classes.

The work can be extended by using other evolutionary algorithms for finding the most important attribute for fault prediction and finding the critical classes and metrics.

REFERENCES

[1] J. Gray. Why do computers stop and what can be done about it? Technical Report 85.7, PN87614, Tandem Computers, Cupertino, 1985.

[2] Grottke, Michael, and Kishor S. Trivedi. "A classification of software faults."Journal of Reliability Engineering Association of Japan 27, no. 7 (2005): 425-438.

[3] WikiWikiWeb. Heisen bug examples. Last modified Jan. 21, 2004, URL = http://c2.com/cgi/wiki?HeisenBugExamples (Link verified on May 26, 2005).

[4] S.Chidamber and C.F.Kemerer, "A metrics Suite for Object-Oriented Design," IEEE Transactions on Software Engineering, vol. SE-20, no.6, 476-493, 1994.

[5] R.Martin, "Oo design quality metrics - an analysis of dependencies," In Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94.

[6] B.Henderson-Sellers, "Object-oriented metrics: measures of complexity," UpperSaddle River, NJ, USA: Prentice-Hall, Inc 1996.

[7] J.Bansiya and C.G Davis, "A hierarchical model for object-oriented design quality assessment," IEEE Transactions on Software Engineering, vol. 28, no. 1, 4–17, 2002.

[8] L.Madeyski and M.Jureczko "Which process metrics improve software defect prediction models? An empirical study," submitted to Information and Software Technology, 2011.

[9] S. N Sivanandam., and S. N. Deepa. Introduction to genetic algorithms. Springer, 2007.

[10] P S. Sandhu, Jagdeep Singh, Vikas Gupta, Mandeep Kaur, Sonia Manhas, Ramandeep Sidhu. "A K-Means Based Clustering Approach for Finding Faulty Modules in Open Source Software Systems." In World Academy of Science, Engineering and Technology 48 2010.

[11] R.Malhotra, Bhupinder Singh and Satinder Pal Ahuja. "Determination of Fault Proneness of Modules in Open Source Software Systems using SVM Clustering Approach." In International Conference on Computer Graphics, Simulation and Modeling (ICGSM'2012).

[12] D. Doval, Diego, Spiros Mancoridis, and Brian S. Mitchell. "Automatic clustering of software systems using a genetic algorithm." In Software Technology and Engineering Practice, 1999. STEP'99. Proceedings, pp. 73-81. IEEE, 1999.

[13] P S Sandhu., Rubinderjit Kaur, and Anamika Sharma. "Evaluation of Fault Proneness of Modules in Open Source Software Systems Using k-NN Clustering."

[14] S. Kaur, Manish Mahajan, and Dr Parvinder S. Sandhu. "Identification of Fault Prone Modules in Open Source Software Systems using Hierarchical based Clustering." ISEMS, Bangkok, July (2011).

[15] Mahajan, Aarti, Vikas Gupta, and Parvinder S. Sandhu. "A Bayes Network Classification Approach For Finding Faulty Modules In Open Source Software Systems."

[16] Mizuno, Osamu, Shiro Ikami, Shuya Nakaichi, and Tohru Kikuno. "Spam filter based approach for finding fault-prone software modules." In Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on, pp. 4-4. IEEE, 2007.

[17] X. Xu, C.-H. Lung, M. Zaman, and A. Srinivasan, "Program restructuring through clustering techniques," in Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM '04), pp. 75–84, IEEE Computer Society, Ottawa, Canada, September 2004.

[18] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: bridging the gap between source and high-level models," in Proceedings of the 1995 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 18–27, October 1995.

[19] Wiggerts, Theo A. "Using clustering algorithms in legacy systems remodularization." In Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on, pp. 33-43. IEEE, 1997.

Authors

Aditi Puri is a post graduate student in the Computer Science and Engineering Department at Lovely Professional University. Her research interests include software engineering, Genetic Algorithm and software evolution.

Harshpreet Singh received the B.tech and M.tech degrees in Computer Science in 2007 and 2009. He is an assistant professor in Computer Science Department at Lovely Professional University and pursuing PhD degree. His research interests include process modelling, CBSD, Software reuse.