International Journal of Computer Sciences and Engineering
Vol.**12**, Issue.**4**, pp.**47-54**, April **2024**
ISSN: 2347-2693 (Online)
Available online at: www.ijcseonline.org

ISSN: 2347-2693 (E)

Research Article

Functional Decomposition as an Anomaly in Object-Oriented Software Design

Brahmaleen K. Sidhu[1]

[1]Dept. of Computer Science and Engineering, Punjabi University, Patiala, India

Corresponding Author:brahmaleen.sidhu@gmail.com

Received: 04/Mar/2024; **Accepted**: 06/Apr/2024; **Published**: 30/Apr/2024. **DOI**: https://doi.org/10.26438/ijcse/v12i4.4754

Abstract: Software must evolve continuously and accept the changes imposed by the environment in order to stay relevant. As the software undergoes updates, the quality of its design degrades. Poor design further deteriorates the quality of software. In the traditional software development processes, quality is often measured at code level using metrics-based approaches. However, quality assessment at model level has various advantages over code level. UML models provide a higher level of abstraction allowing isolation of the core design problem from irrelevant coincidental problems, which typically interfere at code level. Problems uncovered at the design level can be improved directly in the model. Early quality assessment reduces maintenance costs and manages requirement volatility. This paper presents a design flaw detection approach based on machine learning for UML models of object-oriented software. It advances the proposition of a concise quality assurance procedure wherein the root cause of design defects is identified instead of a localized flaw detection and correction approach. The notion of functional decomposition is advanced as an anomalous design tendency as object-oriented software architecture based on functional decomposition compromises on major quality goals like comprehensibility, changeability and semantic consistency. A semi-supervised machine learning technique is used in an unsupervised mode to detect functional decomposition as an anomaly. The precision and recall of the proposed approach were found to be 0.8 each.

Keywords: Functional Decomposition, Machine Learning, Model Refactoring, Object-Oriented Design, Software Quality, UML Class Diagram

1. Introduction

Software is the real driver of innovation and growth of societies across the world. Software lies at the centre of the sophisticated digital technologies, tools and methods that are deployed by organizations for digitizing their operating models. Such transformations from traditional business practices have resulted in higher success rates for enterprises, as per the McKinsey Global Institute report [1] that measures the digitization of United States' economy. Trends like the digital ecosystem approach and replacement of legacy applications by SaaS (Software as a Service) have built momentum in the growth of software industry. Global research and advisory firm, Gartner Inc. has anticipated vigorous growth for the software industry in the forthcoming years, as per their report [2].

The growing dependency of nearly all the segments of society on technology has unlocked software's immense potential for social and economic benefit as well as impairment. A malfunction of industrial-strength software system can have huge impact in terms of financial or business loss, inconvenience to users or loss of property and life. Thus, software systems need to be of high quality with respect to properties like – functionality, reliability, efficiency, maintainability, portability, reusability, flexibility, and interoperability. The shift from project-centric to product-centric delivery in software industry has radically overhauled development organizations.

In their report titled "Application Development and Platforms Primer for 2019" [3], Gartner analysts Wong and Mann assert that software development organizations must shift to a continuous quality mind-set in order to shorten cycles and improve delivery outcomes. Evolving application architecture is said to drive better business outcomes. There is a realization that software quality assurance needs to match up the precision and decorousness of hardware quality processes. Good quality needs to be assured at design stage of development process ahead of construction stage.

In model-driven engineering, models depicting various perspectives of users form the basis of implementation. Such practices along with automatic code generation accommodate frequent changes imposed by environment and higher user acceptance, which is an important measure of product quality. Model refactoring is a form of quality assurance process that continually transforms software models in order to increase their understandability and modifiability. The practice aims to produce better quality and facilitate evolution of the software.

Thus, high quality is an important goal of software development process. In model driven development, high software (code) quality can be reached only if the design quality, i.e. the quality of involved models is high. Quality assessment at model level has other advantages also. Models provide a higher level of abstraction allowing isolation of the core design problem from irrelevant coincidental problems, which typically interfere at code level. Problems uncovered at the design level can be improved directly in the model. Early quality assessment reduces maintenance costs and manages requirement volatility.

Present study focuses on models' quality in object-oriented software as object-oriented development paradigms continue to dominate, as reported by the 2024 TIOBE Index, 2023 IEEE Spectrum Ranking, and 2022 GitHub's Octoverse. The main reason is that the technique of object-orientation models a system analogous to human perception of the real world. The software system is fabricated out of mutually interacting objects that encapsulate behaviour and information associated with the corresponding real-world entities. Also, considerable number of software development paradigms and methods follow object-oriented principles as they are inherently immune to bad design.

This paper is organized as follows, Section 1 contains the introduction of software quality and discusses its significance. The context of the presented study is discussed. Section 2 contain the related work of software design flaw detection. It lists relevant research works carried out in the area of code smell detection and model smell detection. Section 3 introduces the notion of functional decomposition as an anomalous design choice in object-oriented software. Section 4 contain the methodology of implementation of proposed algorithm. Section 5 discusses the results and section 6 concludes research work with future directions.

2. Related Work

The identification of deviations from good design principles and heuristics is known as smell detection. The term "bad smells" was coined by Fowler et al. in the book *Refactoring: Improving the Design of Existing Code* [4] to refer to structures in the code that are potential candidates for improvement. Extensive research has been carried out in code smell detection [5] [6]. Similar to code smells, model smells are defined as elements within the model that symptomize design defects or bad alternatives to recurring design problems (anti-patterns) in object-oriented design. Redundancies, ambiguities, inconsistencies, incompleteness, non-adherence to design conventions or standards, abuse of the modelling notation are typical model smells [7].

UML is a widely accepted modelling language in the field of software engineering. It allows for the visual representation of systems, software architectures, and designs in a standardized way. Various strategies have been used in the literature for analysing the design defects, i.e. smells in UML models. Studies employing pattern-based smell detection [8], [9], [10], [11], [12], [13], [14] identify areas of a design that would benefit from the implementation of design patterns. These design patterns are pre-established templates that provide expert knowledge-based solutions for common design issues. They offer a tried-and-tested approach to implementing relationships and interactions between classes or objects.

Anti-patterns are the recurring and counterproductive design practices. Studies employing metrics-based smell detection [15], [16], [17], [18], [19], [20], [21] use threshold values of quality metrics to mark refactoring opportunities. The selection of threshold is subjective and thus cannot be applied universally. Human judgement plays an important role in using values of quality metrics as indications of smell.

Thirdly, there are research works using rule-based smell detection approaches [22], [23], [24]. These identify both model smells and anti-patterns using a declarative rule definition. These rules are manually defined to identify the symptoms that characterize the smell. Suitable refactoring operations are selected to fix the identified model smells.

In a PhD thesis titled "Development of refactoring technique for architecture-based evolution of object-oriented software systems" [25], it is argued that the conventional methods of model smell detection suffer from various drawbacks. For instance, the selection of threshold value in metric-based approaches is subjective and thus cannot be applied universally. Smells uncovered by deviant values of individual metrics are consequences of sub-optimal realization of design principles, thus, are superficial in nature. In the context of refactoring class diagrams, localized iterative refactoring operations are used to fix architectural defects. Such design transformations introduce cascaded flaws in the design and trigger a vicious cycle of re-evaluation, re-refactoring and synchronization among various parts of the model. This resultant ravel of problems signals the need for a different perspective on bad design in order to derive a more effective refactoring technique. Also, none of the prior works was found to be using machine learning techniques.

3. Functional Decomposition in Object-Oriented Design

Software design methods involve three fundamental decisions [26]. The first and foremost is the criteria for partitioning the software into components. This process of decomposition, also known as factoring, aids the understanding of software to be built, makes the design process easy and leads to an effective design. The decomposition paradigm also lays the foundation for the other two design decisions: regarding various representations of software and design quality. The design phase of the software development process translates the requirements models representing the problem domain into design models representing the solution domain, to be used as blueprints for construction of software.

A function-oriented design approach views the software solution as one big process and partitions it into smaller, simpler processes (tasks) that need to be performed. On the other hand, object-oriented design approach partitions the system into classes or objects that interact among themselves to achieve the software solution.

While cataloguing model smells for UML class diagrams in a study [27], it was discovered that most of the design flaws are caused when object-oriented software is designed with procedural design instincts, i.e. when object-oriented software is contrived on functional decomposition. Such software systems exhibit poor design and face severe drawbacks during the inevitable process of evolution. In the work presented in this paper, functional decomposition is cast as an anomaly in object-oriented design.

Object-oriented software architecture based on functional decomposition compromises on major quality goals like comprehensibility, changeability and semantic consistency and thus calls for a big refactoring (a long series of repeated operations). Following functional decomposition, objects are constructed around tasks (functions) instead of data. The class diagram in **Error! Reference source not found.** shows an example of an object-oriented design exhibiting functional decomposition.

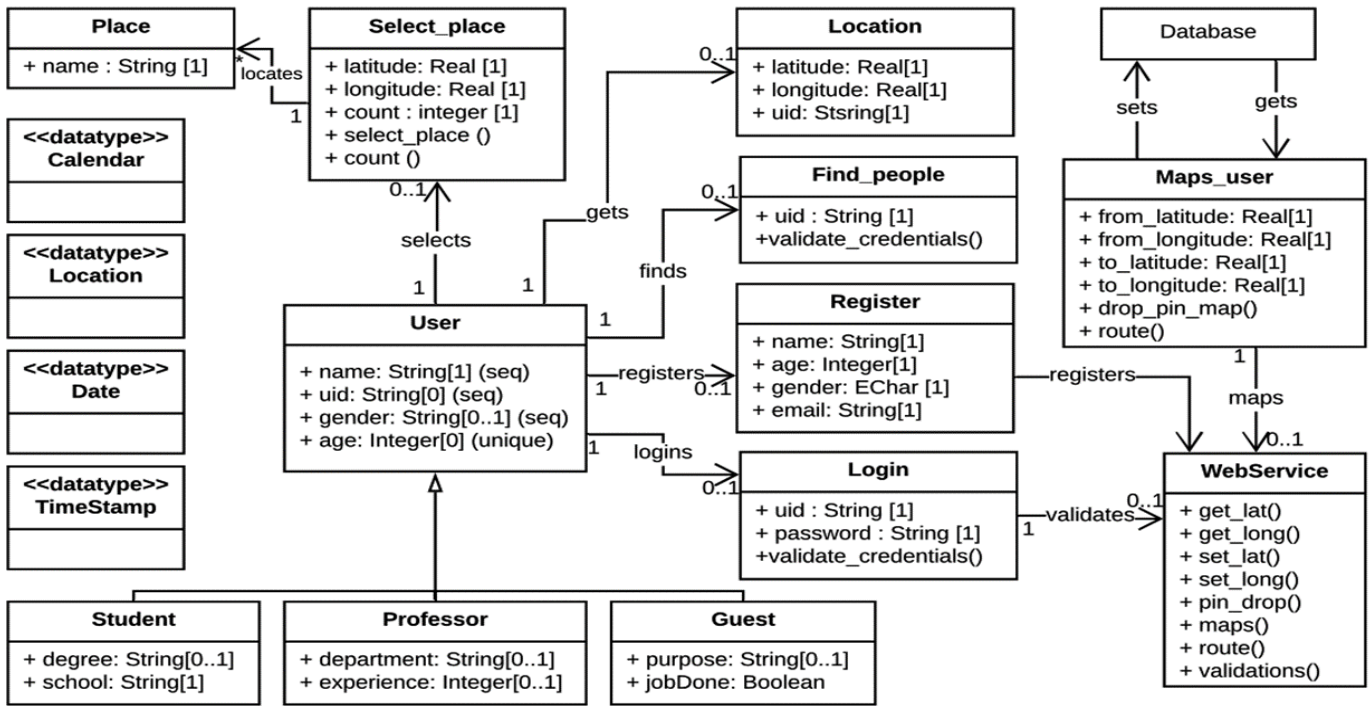

Figure 1. Anomalous UML Class Diagram Detected by Proposed Algorithm

The design depicted in the diagram has improper encapsulation, suboptimal use of inheritance, high coupling and unwieldy size. The class diagram is a part of a navigation system's design. 7 out of the 13 classes, namely, *Student*, *Professor*, *Guest*, *users*, *Location*, *Register* and *Place* contain only data members and do not specify any operations for the instantiated objects. This is in violation with the object-oriented principle of encapsulated objects. Objects must have a well-defined interface that specifies the stimuli to which the object responds. Classes *Find_people*, *Login*, *Select_place* and *Maps_user* represent isolated functions instead of encapsulated objects.

Most of the operations performed in the example system have been clubbed into the class *WebService*. This class does not contain any data and is heavily associated to other classes that avail its services. Thus, the parts of the system are strongly coupled. Proper encapsulation of attributes and operations in the class *users* would reduce redundancy in the classes *Login*, *Find_people* and *Register*, achieve effective inheritance and cut down the unwieldy size of the model. It would also cut down the unnecessary associations and coupling in the model. In the approach proposed here, class models which have lower leverage of the essential object-oriented principles like extensible classes, inheritance, reusability, and polymorphism are identified as anomalies.

4. Methodology of Proposed Approach

"No set of metrics rivals informed human intuition", these words of Beck and Fowler (Fowler et al., 1999) accentuate the role of human perspicacity in performing the refactoring process. Thus, artificial intelligence is an instinctive choice for automating the process. Machine learning is a form of artificial intelligence which enables a computer program to learn from experience (seen data) and subsequently apply the acquired knowledge to make predictions for unseen cases. It is proposed that machine learning has huge application potential in model refactoring because it would provide the capability to discover the subtle relationships among elements of a graphical model that indicate flawed design but are inconspicuous to human eye.

Another motivation to apply machine learning is the snowballing growth of the software engineering world with advances in development paradigms, design methodologies, programming techniques and automated testing. The rapidly changing business rules also have an intriguing effect on quality notions. The current software yield is an assortment of varied sizes and designs. Even within the object-oriented domain, a rigid definition of good design exists no more. This diversity has made manual decisions nearly impossible. Smell detection rules used in the refactoring process must continuously adapt to the changing perceptions of good and bad design. Here, a machine learning algorithm trained by data can outperform human judgement.

Anomaly detection has been widely researched as a problem of machine learning and data mining since Edgeworth's paper *On Discordant Observations* [28]. It refers to the problem of identifying patterns in data that do not conform to expected behaviour. In other words, anomalies are outliers or exceptions that deviate significantly from the majority of the data.

The presented attempt to identify design flaws as anomalies at the modelling stage of object-oriented software development is a first, to the best of our knowledge. The proposed method uses semi-supervised learning technique in an unsupervised mode in order to detect use of functional decomposition in object-oriented design. A semi-supervised learning method [29] uses training data with labelled instances only for the normal classes and an unsupervised method works sans training data. Proposed method is a parametric statistical technique that uses an unlabelled dataset as training data, assuming that the test data contains few anomalies and the model learned during training is robust to these few anomalies.

The proposed approach uses data science methods to tag the outlier UML class diagrams that follow process-based decomposition and do not conform to the data-based object-oriented manifesto of ease of development, low complexity, reusability and easy maintainability (understandability, modifiability, fault detection, testability). Following sections give the details of the approach.

4.1 Feature Set

The algorithm uses an unlabelled training set $T = \{x^1, x^2, \ldots, x^m\}$ of m number of UML class diagrams selected randomly. It is assumed that the number of normal examples outnumber the number of anomalous examples in the training set. Each element of T is a feature vector x of size n, representing n features of a class diagram that are indicative of anomalous object-oriented design. The feature space (R^n) comprises n metrics measuring design properties viz. inheritance, coupling and size to identify the outlier software models (c.f. Table 1).

Table 1. Features Used to Detect Anomalous Class Diagrams

Feature Name	Description
Inheritance Features (reusability, modifiability, testability, probability of fault detection)	
Number of generalizations	Number of parent-child pairs in generalization relationship in the diagram.
Maximum depth of inheritance	The maximum among the DIT [30] values of classes in the diagram. The DIT value for a class within a generalization hierarchy is the length of the longest path from the class to the root of the hierarchy.
Coupling Features (complexity, understandability, maintainability)	
Number of associations	Number of associations in the diagram; includes aggregation and composition. There is an association from class C to class D if C has an attribute of type D.
Total coupling	Sum of Direct Class Coupling (DCC) values of all classes in the diagram. The DCC value for a class is a count of the different number of classes that the class is directly related to [31]. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods. Bidirectional associations are counted twice, because C knows D and vice versa.
Maximum coupling	The maximum among DCC values of classes in the diagram.
Number of dependencies	Number of dependencies in the diagram. There is a dependency from class C to class D if C has an operation with a parameter of type D.
Size Features (reusability, complexity, development effort, maintainability)	
Number of classes	Number of classes in the diagram.
Number of attributes	Total number of attributes in the classes of the diagram.
Number of operations	Total number of operations in the classes of the diagram.

Thus, the dataset is an $m \times n$ matrix where the columns represent the features and each row is a vector representing one class diagram example. The proposed approach requires feature vectors to have normally distributed values. Scaling, logarithmic transformations and square root transformations were applied to features with skewed distributions. Skewness measures asymmetry in given dataset. It represents the manner in which the data are clustered around the average. In a skewed distribution data falls to a side of the mean value. Kurtosis is used as a measure of skewness as follows (1):

$$S_k = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2\right)^2} \qquad (1)$$

Here, S_k computes the kurtosis for vector x with size n and mean μ. As per the given function, kurtosis of the normal distribution is 3. In the calculations, 3 was subtracted from the computed values of S_k, so that the normal distribution has kurtosis of 0. The resultant histograms showing the normally distributed feature vectors are plotted in Figure 2.

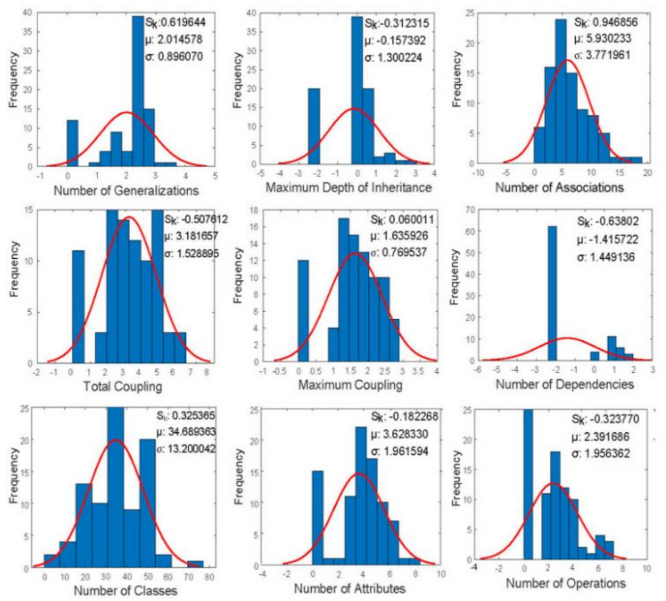

Figure 2. Histograms of Feature Vectors

4.2 Probability Density Estimation

In this step, a statistical model is fit to the training data. A probability density test is applied to determine if an unseen instance of class diagram belongs to this model or not. The training data is modelled as a multivariate Gaussian distribution, a generalization of the one-dimensional normal distribution to higher dimensions. A multivariate Gaussian distribution captures the correlations between class diagram features and is a better choice than univariate distribution when the number of training examples is much greater than the number of features.

The training set of m examples, $\{x^1, x^2,\ldots, x^m\}$, such that $x^i \in R^n$, is modelled using multivariate Gaussian distribution with parameters μ and Σ i.e. $x \sim \mathcal{N}(\mu, \Sigma)$,such that $\mu = \{\mu_1, \mu_2,\ldots, \mu_n\}^T$ is a vector consisting of mean values of n features ($\mu \in R^n$) and Σ is the covariance matrix measuring the variability of the features ($\Sigma \in R^{n\times n}$). μ is computed as in (2).

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^i \tag{2}$$

The covariance matrix Σ is the matrix whose (i, j)th entry is the covariance Σ_{ij} computed as in (3) where the operator E denotes the mean value of the argument.

$$\Sigma_{ij} = \mathrm{cov}(x_i,x_j) = E\left[\left(x_i-\mu_i\right)\left(x_j-\mu_j\right)\right] = E[x_ix_j]-\mu_i\mu_j \tag{3}$$

Given a new example $xtest = \{x^1,x^2,\ldots,x^n\}^T$, a feature vector representing a class diagram under observation, the probability of $xtest$, represented as $p(xtest)$, quantifies the proximity of model to object-oriented design. $p(xtest)$ is computed as in (4).

$$p(xtest) = \frac{\exp\left(-\frac{1}{2}(xtest-\mu)^T\Sigma^{-1}(xtest-\mu)\right)}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \tag{4}$$

Test instances that have lower probability of being generated from the learned model are declared as anomalies, i.e. the class model under observation, represented by *xtest*, is tagged as an anomaly if $p(xtest) < \varepsilon$, where ε is the threshold value. $p(xtest)$ is the probability of *xtest* conforming to the object-oriented behaviour depicted by training set.

5. Results and Discussion

In the training phase of the algorithm, UML class diagrams were serialized using OMG's standard XMI[1] format for creating textual representations of MOF-compliant models. The design quality measurement tool SDMetrics[2] (version 2.35) was used to calculate the metrics(features listed in Table 1) for the XMI representations. 86 UML class diagrams selected randomly from repository of UML class models offered by [32] were used. Hence, the training dataset was sized 86×9. The anomaly detection algorithm was implemented using MATLAB[3] R2017b (version 9.3).

The anomaly detection model learned from the training dataset was validated using validation dataset comprising feature vectors from 21 UML class models. The algorithm output a probability score for each example in dataset. The threshold value (ε) decides the classification of the examples as being normal or anomalous. It is the value above which an example is marked as positive.

Since the problem at hand is an imbalanced binary classification problem (the number of anomalous examples is usually few), the *F*-measure was used to find the best value of ε to use for identifying outliers based on the results from the validation set and the ground truth. Ground truth was established by manual inspection. *F*-measure, the harmonic mean of precision *P* (5) and recall *R* (6) is computed as in (7).

$$P = \frac{\text{truePositives}}{\text{truePositives} + \text{falsePositives}} \tag{5}$$

$$R = \frac{\text{truePositives}}{\text{truePositives} + \text{falseNegatives}} \tag{6}$$

$$F = \frac{2PR}{P+R} \tag{7}$$

The metric precision used here measures the ability of the model to mark only the relevant class diagrams. Precision is defined as the number of true positives per the sum of number of true positives and number of false positives. True positives (*truePositives*) are correctly identified class diagrams that show presence of functional decomposition, and false positives (*falsePositives*) are the diagrams the model labels as positive for depicting functional decomposition but actually do not (cf. Table 2).

Recall measures the ability of the model to find all the relevant cases in the validation dataset. Recall is defined as the number of true positives per the sum of number of true positives and number of false negatives. False negatives (*falseNegatives*) are the class diagrams the model identifies as negative that actually are positive.

Table 2. Confusion Matrix

		PREDICTED	
		Positive	Negative
ACTUAL	Positive	truePositive	falseNegative
	Negative	falsePositive	trueNegative

The threshold selection algorithm iterates over the probability scores of validation set and computes *F*-measure for the different values of ε. The best value of *F*-measure gives the best value for threshold. Using the said validation dataset best threshold value of 6.5865e-09 was obtained at the best *F1* measure of 0.6667.

The Receiver Operating Characteristic (ROC) curve was used to visualize the performance of the model (cf. **Figure 3)** The ROC curve plots the *True Positive Rate* (8) versus the *False Positive Rate* (9).

$$\text{True Positive Rate} = \frac{\text{truePositives}}{\text{truePositives} + \text{falseNegatives}} \quad (8)$$

$$\text{False Positive Rate} = \frac{\text{falsePositives}}{\text{falsePositives} + \text{trueNegatives}} \quad (9)$$

Two anomalous class models were detected in validation set. The vectors representing class diagrams of validation dataset are plotted as in Figure 4. Dashed lines in red represent anomalous examples (labelled 1 by the algorithm).

A test dataset comprising feature vectors from randomly selected 39 UML class models and the threshold value selected during validation was used to test the accuracy of proposed approach. Precision of 0.8 and recall of 0.8 were obtained. The items of test set are plotted in Figure 5 with the anomalous marked in red colour

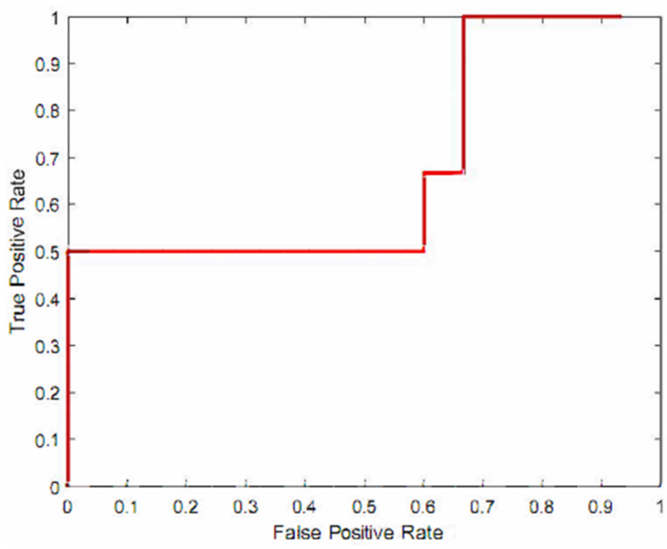

Figure 3. ROC Curve

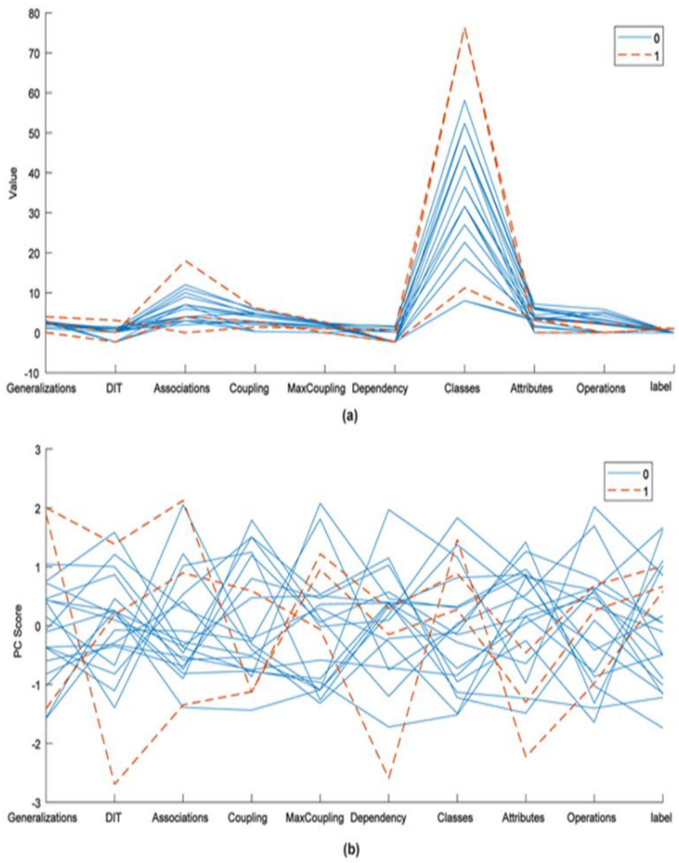

Figure 4. Validation Dataset Plot (a) Values of Features (b) Standardized Principal Component Scores of Features

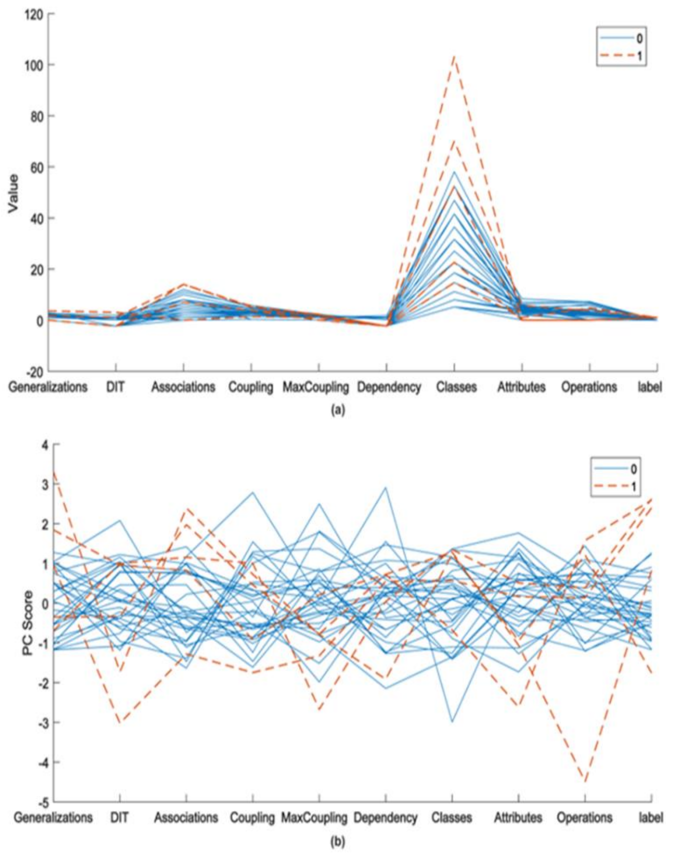

Figure 5. Test Dataset Plot (a) Values of Features (b) Standardized Principal Component Scores of Features

6. Conclusion and Future Scope

This paper presents a novel perspective on flawed software design. A wider perspective is proposed in contrast to the traditional approaches which use individual model smells to mark flawed design. It is suggested that a quality assurance framework working at the roots of design flaws would be incisive. Proposed approach targets object-oriented software at model-level and aims to detect the presence of functional decomposition, a dominant cause of design smells.

Functional decomposition basically represents lack of object-oriented approach. Following function-oriented approach, a software system is viewed a process. During the design activity, the system is partitioned into a series of subfunctions. On the other hand, in object-oriented approach the system is viewed as a set of objects interacting with each other in order to achieve the common goal.

The proposed algorithm uses semi-supervised anomaly detection technique in an unsupervised mode in order to detect use of functional decomposition in object-oriented design. UML class models were transformed into a feature set that trained the proposed system to spot anomalous design. This approach draws attention to the potential of machine learning methods in model-level software quality assurance.

Some practical problems were confronted during the execution of the proposed algorithm. It was observed that defining a feature space that totally typifies object-oriented design is a challenge. Discerning such features and then quantifying them in order to make them fit for application of statistical models is major obstacle especially when the observations are graphical models of software. Notational inconsistencies were found in the UML class diagrams retrieved for training and testing the proposed model. These were handled during the transformation of diagrams into metrics by manual inspection and intervention. Lack of standardized measures and tools that interact with graphical models also restrict the research work.

The perspective of software quality is constantly changing and has many aspects comprising structural quality, conformance to requirements, fitness for user’s needs and satisfaction, compliance with standards, and aesthetic quality. Further work is certainly required to develop and validate definitions of flawed design that are more extensive than model smells and evolve with the concept of quality.

Conflict of Interest
Author declares that there is no conflict of interest.

Funding Source
None.

Acknowledgements
The author would like to express sincere gratitude to Dr. Kawaljeet Singh and Dr. Neeraj Sharma for their invaluable guidance, encouragement, and support throughout the course of this research work. They provided valuable insights, constructive feedback, and unwavering support at every stage of the research process, contributing significantly to the success of this work.

References

[1] J. Manyika, S. Ramaswamy, S. Khanna, A. Yaffe, H. Sarrazin, G. Pinkus and G. Sethupathy, "Digital America: A tale of the haves and have-mores," McKinsey & Company, December **2015.**

[2] N. Gupta, H. Swinehart, J. Poulter and B. Abbabatulla, "Forecast Analysis: Enterprise Application Software, Worldwide, 4Q18 Update," **2019.**

[3] J. Wong and K. Mann, "Application Development and Platforms Primer for 2019," **2019.**

[4] M. Fowler, K. Beck, J. Brant, W. Opdyke and d. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, **1999.**

[5] M. . Y. Mhawish and M. Gupta, "Generating Code-Smell Prediction Rules Using Decision Tree Algorithm and Software Metrics," *International Journal of Computer Sciences and Engineering,* Vol.**7**, No.**5**, pp.**41-48**, **2019**.

[6] M. Kaur and D. Kaur, "Improve the accuracy and time complexity of code smell detection using SVM and Decision Tree with Multi-label Classification," *International Journal of Computer Sciences and Engineering,* Vol.**8**, No.**12**, pp.**66-69**, **2020**.

[7] T. Mens, G. Taentzer and D. Müller, "Challenges in Model Refactoring," in *Proceedings of 1st Workshop on Refactoring Tools*, University of Berlin, July, 2007.

[8] C. Bouhours, H. Leblanc and C. Percebois , "Bad smells in design and design patterns," *Journal of Object Technology,* May-June, Vol.**8**, No.**3**, pp.**43-63**, **2009**.

[9] G. E. Boussaidi and H. Mili, "Understanding design patterns - what is the problem?," *Software: Practice and Experience,* December, Vol.**42**, No.**12**, pp.**1495–1529**, **2012**.

[10] M. El-Sharqwi, H. Mahdi and I. El-Madah , "Pattern-Based Model Refactoring," in *Proceedings of The 2010 International Conference on Computer Engineering and Systems (ICCES),* Cairo, Egypt, **2010.**

[11] R. France, S. Ghosh, E. Song and D.-K. Kim, "A metamodeling approach to pattern-based model refactoring," *IEEE Software,Special Issue on Model Driven Development,* vol. 20, no. 5, pp. 52-58, September/October **2003.**

[12] S. R. Judson, R. B. France and D. L. Carver, "Supporting Rigorous Evolution of UML Models," in *Proceedings of Ninth IEEE International Conference on Engineering Complex Computer Systems, 2004.*, **2004.**

[13] D.-K. Kim, "Design pattern based model transformation with tool support," *Software: Practice and Experience,* April, Vol.**45**, No.**4**, pp.**473-499**, **2015**.

[14] X.-B. Wang, Q.-Y. Wu, H.-M. Wang and D.-X. Shi, "Research and Implementation of Design Pattern-Oriented Model Transformation," in *Proceedings of International Multi-Conference on Computing in the Global Information Technology (ICCGI 2007),* Guadeloupe City, **2007.**

[15] T. v. Enckevort, "Refactoring UML models: using openarchitectureware to measure uml model quality and perform pattern matching on UML models with OCL queries," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and application (OOPSLA)*, Orlando, Florida, USA, **2009.**

[16] M. V. Kempen, D. Kourie, M. Chaudron and A. Boake, "Towards Proving Preservation of Behaviour of Refactoring of UML Models," in *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information*

technologists on IT research in developing countries SAICSIT '05, **2005.**

[17] U. Mansoor, M. Kessentini, M. Wimmer and K. Deb, "Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm," *Software Quality Journal,* Vol.**25**, pp.**473–501**, **2017** .

[18] M. Mohamed, M. Romdhani and K. Ghedira, "M-REFACTOR: A New Approach and Tool for Model Refactoring," *ARPN Journal of Systems and Software,* July, Vol.**1**, No.**4**, pp.**117-122**, **2011**.

[19] T. Ruhroth, H. Voigt and H. Wehrheim, "Measure, Diagnose, Refactor: A Formal Quality Cycle for Software Models," in *Proceedings of 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Patras, Greece, **2009.**

[20] T. Arendt and G. Taentzer, "Implementation Details of Smells and Refactorings for UML Models within the Eclipse Modeling Framework," Philipps Universität Marburg, Marburg, November 4, **2011.**

[21] A. C. Jensen and B. H. Cheng, "On the use of genetic programming for automated refactoring and the introduction of design patterns," in *Proceedings of the 12th annual conference on Genetic and Evolutionary Computation (GECCO)*, Portland, Oregon, USA, **2010.**

[22] M. Akiyama, S. Hayashi, T. Kobayashi and M. Saeki, "Supporting Design Model Refactoring for Improving Class Responsibility Assignment," in *Model Driven Engineering Languages and Systems (Proceedings of 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011)*, vol. 6981 of Lecture Notes in Computer Science, J. Whittle, T. Clark and T. Kühne , Eds., Springer Berlin Heidelberg, pp.**455-469, 2011.**

[23] Ł. Dobrzanski, "UML Model Refactoring- Support for Maintenance of Executable UML Models," Sweden, July **2005.**

[24] M. Stolc and I. Polasek, "A visual based framework for the model refactoring techniques," in *Proceedings of IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Herlany, Slovakia, **2010.**

[25] B. K. Sidhu, "Development of refactoring technique for architecture based evolution of object oriented software systems," Punjabi University, Patiala, **2019.**

[26] R. S. Pressman, Software Engineering, A Practitioner's Approach, 7th ed., New York: McGraw-Hill, **2010.**

[27] B. K. Sidhu, K. Singh and N. Sharma, "A Catalogue of Model Smells and Refactoring Operations for Object-Oriented Software," in *Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, Coimbatore, **2018.**

[28] F. Y. Edgeworth, "On discordant observations," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science,* Vol.**23**, No.**143**, pp.**364-375**, **1887**.

[29] O. Chapelle, B. Scholkopf and A. Zien, Semi-Supervised Learning, The MIT Press, **2006.**

[30] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering,* June, Vol.**20**, No.**6**, pp.**476-493**, **1994**.

[31] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering,* January, Vol.**28**, No.**1**, pp.**4-17**, **2002**.

[32] B. Karasneh and M. R. Chaudron, "Img2UML: A System for Extracting UML Models from Images," in *39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander, Spain, **2013.**

AUTHOR'S PROFILE

Brahmaleen K. Sidhu earned her Ph.D. degree in Faculty of Engineering and Technology from Punjabi University, Punjab, India, M.Tech. degree in Computer Science and Engineering from the Punjab Technical University, Punjab, India, and B.Tech. degree in Computer Science and Engineering from Punjabi University. She is currently working as Assistant Professor in the Department of Computer Science and Engineering, Punjabi University and has around 18 years of teaching experience. Her research interests include software architecture, software evolution, software quality, refactoring, model-driven development, data science and machine learning. She has around 80 research papers in reputed international journals including Thomson Reuters (SCI & Web of Science) and conferences including IEEE, and a book titled "A Handbook of Reinforcement Learning" published in 2023. She has been awarded the "International Innovative Educator Award 2021" and is listed in "100 Eminent Academicians of 2021" by International Institute of Organized Research.