THE ASIAN BULLETIN OF BIG DATA MANAGMENT
Vol. 4. Issue 1 (2024)
https://doi.org/ 10.62019/abbdm.v4i1.107

Code Smell Detection and Software Refactoring Research: A Systematic Literature Review the Transformative

Israr Ali, Sajjad Hussain Rizvi, Syed Hasan Adil, *Abdul Ahad Abro

Chronicle

Article history

Received: February 12, 2024
Received in the revised format: Feb 24, 2024
Accepted: Feb 27, 2024
Available online: Feb 29, 2024

Israr Ali, Sajjad Hussain Rizvi, Syed Hasan Adil and Abdul Ahad Abro are currently affiliated with the Department of Computer Science Iqra University Karachi, Pakistan. Karachi, Pakistan
Email: Israr.ali@iqra.edu.pk
Email: dr.sajjad@szabist.pk
Email: hasan.adil@iqra.edu.pk
Email: abdul.ahad@iqra.edu.pk

Corresponding Author*

Abstract

Today we are experiencing rapid enhancements in software systems and their development. The software industry's demand for tools and techniques for software development, especially automatic and less time-consuming, is increasing daily. Software refactoring and code smell detection are now expanded from code-level changes to the architecture, model, and requirements restructuring. We are moving from an object-oriented paradigm towards cloud computing, web development and mobile application development and so much more. Therefore, code smell and refactoring techniques are talk of the town in various research communities in their objectives. In this paper we will study the existing tools and techniques, research progress by doing a systematic literature review in the field of code smell detection and software refactoring's. We will also classify the existing research techniques, identify the trends in code smell detection and refactoring and try to highlight the gaps in the area for researchers.

Keywords: code smell detection, software refactoring, literature review.

INTRODUCTION

A predictive model is a black box and when we present a feature set of any record it will automatically register a class label. We can describe a predictive model as a problem related to mathematics of finding out target function F, target function F will map all features of set say S1 in a dataset (DS) to one class label. We can say that target of these kind of models is to discover optimal F. There are four basic parts in a predictive model used in these types of studies.

Datasets datasets are the basic and most important component of predictive model is Dataset. Dataset is very important as they have a large impact of model overall performance. We can use different types of datasets when designing predictive model for software engineering. Source Codes, bug reports, and SRS can be used for defect prediction, bug classification and requirement classification.

Features: Features are an important component in predictive model especially in training phase. A good feature set will be able to allow models for prediction to learn efficiently the patterns (and analyses them efficiently) in datasets.

Model Algorithm: We can use different types of algorithms to construct our predictive model, recently several deep learning-based algorithms are used to develop predictive models, such as CNN and RNN based networks Abro et al. (2020).

Performance Measures of Model: most common performance measures like precision, recall, F score, AUC and accuracy have been used in software engineering studies Abro et al. (2021). Predictive models are now extremely popular in research scholars especially in the domain of SE. They can be built by using different types of datasets of SRE (Software Requirement Engineering), APIs, BRS (Bug Reporting Systems) and run time data of Open-Source Software. The output of

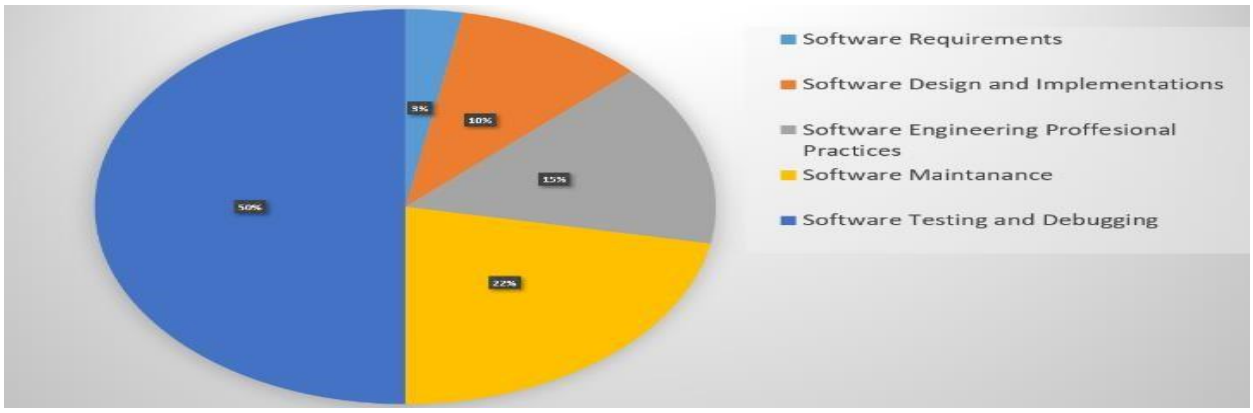

Figure 1.
Research domains in the field of software engineering

These predictive models are the unique features found in the data. Different predictive models have been developed including Code smell detection Abro et al. (2023), Issues in APIs and their classification Zhang et al. (2016) and prediction of defects in software's Alomar et al. (2019). We have made studies to find out primary studies done in Research domains in the field of software engineering. The results are shown in Figure 1.

As we can see the major publication are in the domain of Software Testing and Debugging. The reason could be this domain has more predictive model applicable tasks like software defect prediction, software code smell detection and software bug report maintenance. We also studied the specific topics where most of the predictive model related papers are found. Table 1 describes the ranking of these specific topics.

Table 1.
Predictive models

Ranking	Software Engineering Task Name	Research Domain
1	Defect Prediction	Software Testing and Debugging
2	Bug Report Management	Software Testing and Debugging
3	Software Quality Management	Software Testing and Debugging
4	Software Cost Prediction	Software Design and Implementation
5	Developer behavior Analysis	Software Engineering Professional Practices
6	Code Clone Management	Software Maintenance
7	Software Change Management	Software Maintenance

Table 1: Ranking of SE topics in last decade automated by using Predictive models Lin et al. (2019). Software Refactoring is a technique or set of techniques that is used by experienced Software Engineers to change the internal structure of code to improve the overall quality aspect of the Software. The Main factor is that the changes does not affect the External Structure of the Software. This Process very much depends on developer

expertise and experience in this regard. Automated Refactoring can be extremely helpful for Software Developers especially new ones with low experience. Machine Learning has been used in last decade to accomplish this task and recently researchers are trying to achieve this task by using different deep learning methods. In this research we will first see the weather deep learning can be used effectively to automate this process and then we will propose a novel deep learning based predictive model to achieve this goal. Software Refactoring task are presented here in Table 2, Table 3 and Table 4 as Class, Methods and Variable Levels Tantithamthavorn et al. (2016).

Table 2.
Class level Software Code Refactoring

Refactoring	Problem and Solution
Extract Method	Related statements that can be grouped together. Extract them to a new method.
Inline Method	Statements unnecessarily inside a method. Replace any calls to the method with the method's content.
Move Method	A method does not belong to that class. Move the method to its rightful place.
Pull Up Method	Sub-classes have methods that perform similar work. Move them to the super class
Push Down Method	The behavior of a super-class is used in few sub-classes. Move it to the sub-classes.
Rename Method	The name of a method does not explain the method's purpose. Rename the method.
Extract And Move Method	The two aforementioned refactorings together.

Table 3.
Method level Software Code Refactoring

Refactoring	Problem and Solution
Extract Variable	Hard-to-understand/long expression. Divide the expression into separate variables.
Inline Variable	Hard-to-understand/long expression. Divide the expression into separate variables.
Parameterize Variable	Variable should be a parameter of the method. Transform variable into a method parameter.
Rename Parameter	The name of a method parameter does not explain its purpose. Rename the parameter.
Rename Variable	The name of a variable does not explain the variable's purpose. Rename the variable.
Replace Variable w/ Attribute	Variable is used in more than a single method. Transform the variable to a class attribute.

Table 4.
Variable Level Refactoring

Refactoring	Problem and Solution
Extract Class	A class performs the work of two or more classes. Create a class and move the fields and methods to it.
Extract Subclass	A class owns features that are used only in certain scenarios. Create a subclass.
Extract Super-class	Two classes own common fields and methods. Create a super class and move the fields and methods.
Extract Interface	A set of clients use the same part of a class interface. Move the shared part to its own interface.
Extract Interface	A class is in a package with non-related classes. Move the class to a more relevant package.
Rename Class	The class' name is not expressive enough. Rename the class.

It is long been a fact accepted that Software refactoring is an essential ingredient in good quality software developer tools, but how to do Refactoring and what steps are needed and where to apply these techniques is a problem specially for new developers and Software development environments that want to automate this process.

LITERATURE REVIEW

Code Smell detection and code refactoring has always been done in informal ways by software developers and it hugely depends on their experience and expertise. The term 'refactoring' was first used in 1990 Mens et al. (2004) Palomba et al. (2018). In Poon et al. (2006), published their survey paper about software refactoring in 2004 and emphasized the importance of formalism and process methods with tools to apply refactoring in better ways. In Mens et al. (2004), published a survey in 2004 about code smell detection and concluded that impact of code smell detection remains far from fully understandings and needs attention and knowledge of some techniques are still not understood properly. In Zhang et al. (2019) performed a machine learning and search-based case study on software refactoring and concluded that automated approach can outperform manual refactoring techniques.

Additionally, they used ANN and GA to choose the refactoring solution. The systematic literature review and provided an overview of code smell detection approaches. They concluded that Extract Class and Move method are two most common refactoring technique researchers have automated. In this research they classified the tools and techniques of refactoring based on their detection method. Abro et al. (2021) selected four code smells (Long Method, Feature Envy, Large Class, and Data Class) with 16 different ML algorithms. Their result suggested that J48 and random forest were best in terms of performance and SVM was poor. It performed a review on code smell detection tools and type of smell they were able to identify. They identified that ML is not very much used in detection of code smell and other approaches are more frequently used. The research on refactoring focusing on object-oriented programming. Authors build a predictive model over OOP systems and used machine learning classifiers and achieved 90% accuracy. The search-based algorithms applied on code refactoring's and

concluded that genetic algorithms are most used in this regard. Singh highlighted the tools and datasets used in code smell and refactoring's and concluded that God Class and feature envy are most used by researchers in automated detection. Literature review on impact of code smell detection on SDLC and identified that Human based code smell detection is not reliable and developer expertise significantly impact SDLC while applying code smell detection techniques. The literature review and focused on prevention rather than fixing code after code writing (inspection). The Machine learning based literature review in 2019 and suggested that source code base approaches are more popular rather than modular based approaches.

ML based approaches and his results shows that God classes and Functional Decomposition are most used code smells and Decision Tree and SVM are mostly used by researchers. It used 10 different ML algorithm on 5 different datasets to investigate the algorithms performance and proposed a new refactoring approach and proved that his approach outperformed other approaches. In literature review, 16 different ML algorithms for code smell detection and concluded that J48 and Random Forest obtained the best result while SVM produced the worst performance. Further, studied 6 different ML algorithms for predicting refactoring and suggested that Random Forest are best for prediction and process and ownership matrices are best suitable for creation for better model.

Moreover, worked on feature envy detection using deep learning and ANN and proposed a new approach for smell detection. Lastly the proposed a deep learning base approach to detect code smells this time for four different code smells. Deep Learning is getting popular in code smell detection techniques suggestions and it can be verified that the interest in this direction is increasing.

Table 5.
Number of results against code smell on https://scholar.google.com/

Keyword	Year	Results Count
"code smell", "deep learning"	2015	1 result
"code smell", "deep learning"	2016	3 results
"code smell", "deep learning"	2017	5 results
"code smell", "deep learning"	2018	21 results
"code smell", "deep learning"	2019	70 results
"code smell", "deep learning"	2020	87 results
"code smell", "deep learning"	2021	180 results

Heuristics based approaches

An approach of solving any problem that used shortcuts to produce solution and that solution may or may not be optimal is called a heuristics-based approach. Research Studies in the domain of code smell detection are done by using heuristics-based approaches as given in table 6. These are tools easily available on internet to download install as plugins.

Table 6.
Heuristics based approaches

Jdeodorant	Tsantalis and Chatzigeorgiou 2009 [28]	Feature Envy, Long Method, Type/State Checking, and God Class, and make appropriate refactoring suggestions to resolve them
JMove	Terra 2018 [29]	Move Method refactoring based on similarity between dependency sets.
method movements tool	Hui Liu, Yuting Wu, Wenmei Liu, Qiurong Liu, and Chao Li. 2016[30]	Whenever a method is moved, the tool checks methods within the same class and suggests to move methods that have a strong relationship with the moved method
a refactoring recommendation tool	Naoya Ujihara, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2017 [31]	The authors capture semantic similarity between a method and a class by computing cosine similarity between TF-IDF vectors
Relational Topic Models (RTM)	Bavota 2013 [32]	The approach considers both structural (method calls) and textual (identifiers and words in comments) information extracted from the source code.

ML / DL -based approaches

Machine learning based tools for code smell detection are also available on internet to be used as plugin in IDE's. Some of them are given in the table 7.

Table 7.
ML / DL -based approaches

word2vec technique	H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou, and L. Zhang. 2019. [33]	deep learning-based approach to detect Feature Envy, Long Method, Large Class, and Misplaced Class code smells.
coding criterion proposed	Mouna Hadj-Kacem and Nadia Bouassida. 2019 [34]	an approach to detect code smells (Feature Envy, Blob, and Long Method) using deep learning techniques
Designite	Tushar Sharma, Vasiliki Efstathiou, Panos Louridas, and Diomidis Spinellis. 2019. [35]	The results show that a Recurrent Neural Network (RNN) performs better than a CNN for some code smells. The authors also investigated the effectiveness of the transfer-learning technique by training a deep learning model on samples of C# code and then using this model to predict code smells in Java

REVIEW METHODOLOGY

In this section we will discuss the process we will follow for conducting this review. The review approach and process followed by us is given in figure 2 below. This process is basically divided into three phases. First planning and then conducting and in the end reporting the result. In the first stage the datasets were identified from where we can find papers, we also formed the research questions (RQs). In the next phase we shortlisted the papers and irreverent papers were excluded from our research. At the end in the final phase all result found were documented and research questions were answered.

Planning Phase: Planning phase was divided into two phase first is paper searching and second is Research Question preparation.

Paper Searching: We started our search by visiting and exploring different online databases, journals, conferences and internet links. This search was divided into three steps. In the first step all online databases like google scholars, IEEE Explorer, ACM Digital

Library and Springer were used. The search string for this research with search terms was as follows

Figure 2.
Review Methodology

(Code refactoring or refactoring techniques) and (code smell or code smell detection) and (software Maintenance matrices) or (refactoring tools)

The second step was to filter only well-known journals and conference papers and selection of most recent year's publications. Research Questions

While conduction of paper search and literature review so many research questions were raised but some of them were short listed based on review. Table 8 below provides the research questions for the current study

Table 8.
Research Questions

RQ1	What are the different code smells that are already detected by studies?
RQ2	What are the different code smells that are not yet detected by studies?
RQ3	What are the different refactoring techniques that are already used by studies?
RQ4	What are the different refactoring techniques that are not used by studies?
RQ5	Which refactoring technique should be used for a specific code smell?
RQ6	What are the different software metrics used in the detection studies?

Conducting Phase: In the next phase we are discussing about the conducting process of review. First, we selected the paper by using our inclusion criteria and then filter out the papers by using our exclusion criteria.

Inclusion Criteria: All duplicate and not relevant papers were not included, we focused on our RQs and then set our inclusion criteria.

- Papers in which in refactoring technique was identified or discussed.
- Papers in which any smell was identified or detected.
- Papers in which software maintenance quality matrices were used to show the impact of refactoring.
- Papers where refactoring was suggested using machine learning.
- Papers where refactoring was suggested using deep learning.

Exclusion Criteria: we focused on our RQs and then set our exclusion criteria.

- Papers in which code smell and refactoring was not discussed.

- Papers in which our RQs are not relevant.
- Papers which are before 2001
- Papers not providing and experimenting results.

Distribution of paper

We divided our collected research papers into three different categories. The sources of our papers were well known journals, recognized conferences and book chapters. The category wise distribution of research papers is depicted in figure 3

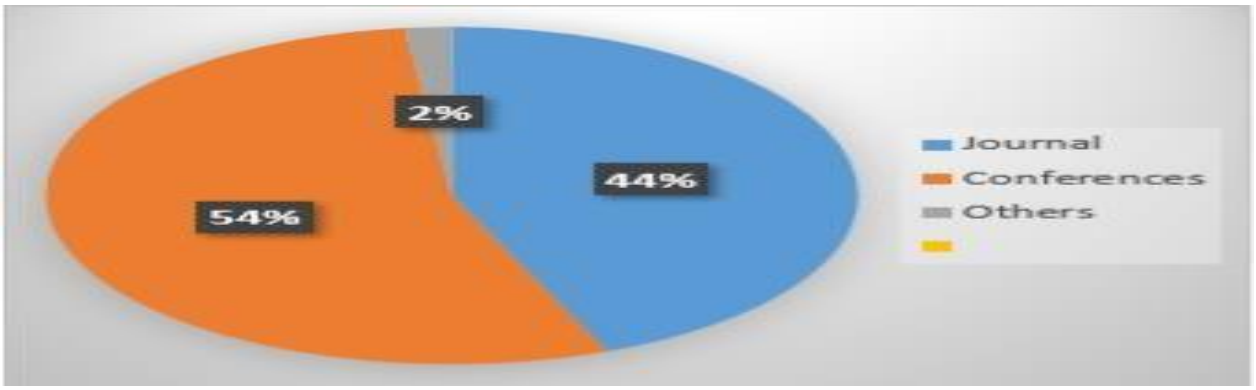

Figure 3.
Paper distribution as per publications

RESULTS

The selected research papers were analyzed based on all RQs and results are given below

RQ1: What are the different code smells that are already detected by studies?

Code smells are the indicators that there is a refactoring opportunity in code. Some of the most detected code smells are listed in table 9 with the number of research papers count.

Table 9.

No	Code smell	Count	No	Code smell	Count
1	Feature envy	22	16	Speculative generality	5
2	Data class	19	17	Schizophrenic class	4
3	Intensive coupling	17	18	Divergent change	4
4	Lazy class	11	19	Long method	4
5	Long parameter list	11	20	Duplicate code	3
6	Message chain	10	21	Inappropriate intimacy	3
7	Brain method	10	22	Complex	3
8	Large class	9	23	Type checking	3
9	Refused bequest	8	24	Switch statements	3
10	Spaghetti Code	7	25	Class data private	3
11	Blob	6	26	Shotgun survey	2
12	Parallel inheritance	6	27	Middle man	2

13	Temporary field	5	28	Brain class	2
14	Data clumps	5	29	Class hierarchy problem	2
15	Functional decomposition	5	30	Nested try statements	2

Table 9 code smell detected

RQ2: What are the different code smells that are not yet detected by studies?

By going through stablished code smells we found some of the smells that are not yet detected by any tool or technique as per our understandings. All these smells are listed in table 10.

Table 10.

Code Smell Name
Primitive Obsession
Alternative Classes with Different Interfaces
Comments
Dead Code
Incomplete Library Class
Middleman
Cyclomatic complexity
Down casting
God Line
God Class

Table 10 code smell not detected

RQ3: What are the different refactoring techniques that were investigated by studies?

Refactoring is code changes that improves code or product quality but behavior remains same. It the solution of code smells and it is a different technique then rewriting the code from scratch. Some of the most used refactoring techniques that are used in research papers are listed here in table 10.

Table 11.
Refactoring techniques investigated

No	**Refactoring technique**	**Count**	**No**	**Refactoring technique**	**Count**
1	Extract class	9	19	Move method	2
2	Extract method	8	20	Consolidate conditional expression	2
3	Form template method	5	21	state/strategy	2
4	Introduce parameter object	5	22	Collapse hierarchy	2
5	Extract superclass	4	23	Introduce null object	2
6	Extract subclass	4	24	Remove parameter	2
7	Move field	3	25	Inline method	2
8	Inline temp	3	26	Replace delegation with Inheritance	2
9	Introduce assertion	3	27	Push down field	2
10	Encapsulate collection	3	28	Remove setting method	1
11	Push down method	3	29	Replace temp with query	1
12	Pull up field	3	30	Replace method with method object	1
13	Pull up method	3	31	Encapsulate field	1

14	Preserve whole object	3	32	Add parameter	1
15	Duplicate observed data	3	33	Replace type code with	1
16	Hide delegate	2	34	Inline class	1
17	Separate query from modifier	2	35	Extract interface	1
18	Replace data value with object	2	36	Rename method	1

RQ4. What are the different refactoring techniques that are not used by studies?

By going through stablished refactoring techniques we found some of the techniques that are not yet solved or discussed by any tool or technique as per our understandings. All these techniques are listed in table 12.

Table 12.
Refactoring techniques not investigated

No	Refactoring technique	No	Refactoring technique
1	Bi Association to Uni directional	15	Replace Array with Object
2	Change Reference to Value	16	Conditional with Polymorphism
3	Change Unidirectional Association to Bidirectional	17	Replace Constructor with Factory Method
4	Change Value to Reference	18	Replace Error Code with Exception
5	Duplicate Conditional Fragments	19	Replace Exception with Test
6	Decompose Conditional	20	Replace Magic Number with Constant
7	Hide Method	21	Replace Nested Conditional
8	Introduce Foreign Method	22	Replace Parameter with Methods
9	Introduce Local Extension	23	Replace Parameter with Method Call
10	Parameterize Method	24	Replace Subclass with Fields
11	Preserve Whole Object	25	Self-Encapsulate Field
12	Pull Up Constructor Body	26	Separate Query from Modifier
13	Remove Control Flag	27	Substitute Algorithm
14	Remove Middle Man		

RQ5: Which refactoring technique should be used for a specific code smell?

By going through stablished refactoring techniques we found some code smells and some of the techniques that can be used to solve code smells. All these techniques are listed in table 12.

Table 13.

No Bloaters	Code Smells	Refactoring techniques Name
1	Long Method	Extract Method Replace Method with Method Object. Replace Temp with Query, Introduce Parameter Object or Preserve Whole Object Decompose Conditional
2	Large Class	Extract Class Extract Subclass Extract Interface Duplicate Observed Data

3	Primitive Obsession	Replace Data Value with Object. Preserve Whole Object. Introduce Parameter Object Replace Type Code with State/Strategy. Replace Type Code with Subclasses Replace Type Code with Class Replace Array with Object.
4	Long Parameter List	Replace Parameter with Method Call. Preserve Whole Object. Introduce Parameter Object.
5	Data Clumps	Extract Class Introduce Parameter Object Preserve Whole Object

Table 14.
Code smell with suitable refactoring techniques

Object-Orientation Abusers		
6	Alternative Classes with Different Interfaces	Extract Superclass Add Parameter Parameterize Method Move Method Rename Methods
7	Refused Bequest	Replace Inheritance with Delegation. Extract Superclass
8	Switch Statements	Introduce Null Object. Replace Parameter with Explicit Methods Replace Conditional with Polymorphism. Replace Type Code with State/Strategy. Replace Type Code with Subclasses Move Method. Extract Method
9	Temporary Field	Introduce Null Object Replace Method with Method Object. Extract Class
Change Preventers		
10	Divergent Change	Extract Class. Extract Superclass and Extract Subclass
11	Parallel Inheritance Hierarchies	Move Field Move Method
12	Shotgun Surgery	Move Field Move Method Inline Class
Dispensable		
13	Comments	Introduce Assertion. Rename Method Extract Method. Extract Variable
14	Duplicate Code	Extract Method Pull Up Field Pull Up Constructor Body. Form Template Method.

		Substitute Algorithm.
		Extract Superclass
		Extract Class
15	Data Class	Encapsulate Field
		Encapsulate Collection
		Move Method and Extract Method
		Remove Setting Method and Hide Method
16	Dead Code	Remove Parameter.
		Inline Class or Collapse Hierarchy
	Lazy Class	Collapse Hierarchy
		Inline Class
17	Speculative Generality	Inline Method
		Remove Parameter.
		Collapse Hierarchy.
Couplers		
18	Feature Envy	Move Method.
		Extract Method
19	Inappropriate Intimacy	Move Method and Move Field
		Extract Class and Hide Delegate
		Change Bidirectional Association to Unidirectional.
		Replace Delegation with Inheritance.
20	Incomplete	Introduce Local Extension.
	Library Class	Introduce Foreign Method.
21	Message Chains	Hide Delegate.
		Move Method.
		Extract Method
22	Middle Man	Remove Middle Man

Software metrics helps us understand the software code properties from different angles. Figure 4 given below displays the matrices used in different research paper especially for code smell and refactoring impact on code.

CONCLUSIONS

In this review study we have investigated different code smells and refactoring techniques and software metrics. Total 80 papers were selected for this purpose, 6 RQs were established and answered. Findings of this study are given below. For refactoring techniques, Move method, Extract Class/ Method are automated by most researchers. For code smell detection Long method code, Feature Envy are detected by most researchers. Cohesion, Coupling and Complexity software metrics are mostly used by researchers to prove their technique impact.

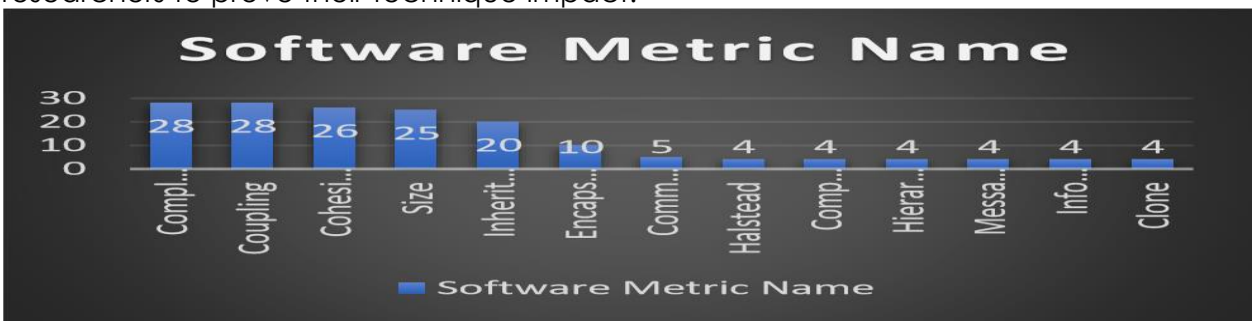

Figure 4.
Software Metrics used in research papers

FUTURE WORK

We think that following are the areas where we can further work andprogress.s

Consequently, a thorough examination can be carried out to extrapolate the outcomes to all projects utilizing object-oriented languages, such as C, C++, C#, etc.

To lessen the effort of the maintenance phase, research can be done on class prioritization and determining the best refactoring order.

The effects of restructuring on various software qualities, such as internal and external quality features, can be examined through a systematic review process.

It is possible to create a deep learning-based, free, open-source solution for developers that addresses code smell and software refactoring.

DECLARATIONS

Acknowledgement: We appreciate the generous support from all the supervisors and their different affiliations.
Funding: No funding body in the public, private, or nonprofit sectors provided a particular grant for this research.
Availability of data and material: In the approach, the data sources for the variables are stated.
Authors' contributions: Each author participated equally to the creation of this work.
Conflicts of Interests: The authors declare no conflict of interest.
Consent to Participate: Yes
Consent for publication and Ethical approval: Because this study does not include human or animal data, ethical approval is not required for publication. All authors have given their consent.

REFERENCES

Abro, A. A., Khan, A. A., Talpur, M. S. H., Kayijuka, I., & Yaşar, E. (2021). Machine learning classifiers: a brief primer. University of Sindh Journal of Information and Communication Technology, 5(2), 63-68.

Abro, A. A., Siddique, W. A., Talpur, M. S. H., Jumani, A. K., & Yaşar, E. (2023). A combined approach of base and meta learners for hybrid system. Turkish Journal of Engineering, 7(1), 25-32.

Alomar, E. A., Mkaouer, M. W., Ouni, A., & Kessentini, M. (2019). Do design metrics capture developers perception of quality and empirical study on self-affirmed refactoring activities. arXiv preprint arXiv:1907.04797.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016, May). Automated parameter optimization of classification techniques for defect prediction models. In Proceedings of the 38th international conference on software engineering (pp. 321-332).

Abro, A. A. (2020). Identifying the machine learning techniques for classification of target datasets. Computer Science-Artificial Intelligence.

Zhang, J., Wang, Z., Zhang, L., Hao, D., Zang, L., Cheng, S., & Zhang, L. (2016, July). Predictive mutation testing. In Proceedings of the 25th International Symposium on Software Testing and Analysis (pp. 342-353).

Lin, B., Zampetti, F., Bavota, G., Di Penta, M., & Lanza, M. (2019, May). Pattern-based mining of opinions in Q&A websites. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (pp. 548-559). IEEE.

Mens, T., & Tourwé, T. (2004). A survey of software refactoring. IEEE Transactions on software engineering, 30(2), 126-139.

Palomba, F., Tamburri, D. A., Fontana, F. A., Oliveto, R., Zaidman, A., & Serebrenik, A. (2018). Beyond technical aspects: How do community smells influence the intensity of code smells?. IEEE transactions on software engineering, 47(1), 108-129.

Poon, C. S., & Chan, D. (2006). Feasible use of recycled concrete aggregates and crushed clay brick as unbound road sub-base. Construction and building materials, 20(8), 578-585.

Mens, T., & Tourwé, T. (2004). A survey of software refactoring. IEEE Transactions on software engineering, 30(2), 126-139.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., ... & Zhang, D. (2019, August). Robust log-based anomaly detection on unstable log data. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 807-817).

Abro, A. A., Talpur, M. S. H., Jumani, A. K., Siddique, W. A., & Yaşar, E. (2021). Voting combinations-based ensemble: A hybrid approach. Celal Bayar University Journal of Science, 18(3), 257-263.