International Journal of Computer Applications (0975 – 8887)
Volume 177 – No. 35, February 2020

Various Test Case Generation in Code Smell Detection Tools and Testing Methods: Review

Deepika Tewatia
Research Scholar (CSE)
Baba Mastnath University
Rohtak, Haryana

Devender Kumar, PhD
Associate Professor (CSE)
Baba Mastnath University
Rohtak, Haryana

ABSTRACT

Test case generation in terms of code smell refers to the features of the software that recognizes a code and design issues which make software hard to realize, evolve and preserve code. Generally, the maintenance and detection of the software applications become more difficult due to the presence of the smell. Programmers are unable to identify the source code applications and face issues to understand the source code of the project. Simultaneously, a code smell face problem to refractors and developers for upgrading and maintenance of the source code. Present, research is active in the automated detection and testing of the bad code smell. Without the knowledge of the code smell with diverse refactoring, and efficient tool make the detection of the code difficult. Particularly, the code smell in software is based on the programming of the source code, that may lead to difficulty in detection of bad code smell. This paper analysis the detection tools, method of code smells and methods for the detection of bad test code smell. The categories of different test code smells are described which includes applications, classes and different method-level code smells. Moreover, detail definition of the bad smell in source code and its types in source code is also elaborated. In addition to that, bad code smell detection is described which includes automated detection and machine learning methods for identifying the bad code smell. Additionally, the automated tools which are given as, a check style, décor, infusion, deodorant and iplasma. The detection methods are decision tree (DT), Learning group rules, Multilayer perceptron (MLP), Naïve Bayes (NB), Support Vector Machine (SVM), Radial basis system (RBS) network.

Keywords

Code smell, Test case generation, Machine learning, Bad smell detection, Automated tools.

1. INTRODUCTION

Testing plays an essential role in the whole software advancement effort. During the development of the software scheme, test suites and repairing of the software requires updating software. Software testing is measured in manual or in an automated way [1] [2]. An automated test suite helps in the maintenance of the activities like as code command and regression testing [3]. A STCE (software test code engineering) is referred as a systematic approach to develop, verify and maintain high quality of the test code [4] [5]. If the program has s/w evolution and maintenance issue, it is referred as bad smell of test code. At present there are various detection methods that helped in the evaluation of the bad smell [6]. The various categories of code smell are given as, application smell, class level smell and method level smell [7][8].

i) *Application smell:* It provides duplicate information which is similar to the actual data and Its architecture is more complex.

ii) *Class level smell:* It contains a large number of classes, variable with no literal and inappropriate data.

iii) *Method level smell*: The different parameter is read at the same time. The complex structure of coding is created.

The numerical measure of the software is S/W metrics. The below table 1 main focused on the metrics of the source code.

Table 1: Software Representations [18]

Representation	Label	Level
NOM	It is a number of methods	Class
LOC	Lines of code	Class
PAR	Number of parameters	Method
MLOC	Method, lines of code	Method

The developers resolve the issue of the complex design and maintenance through refactoring method in an automated way.

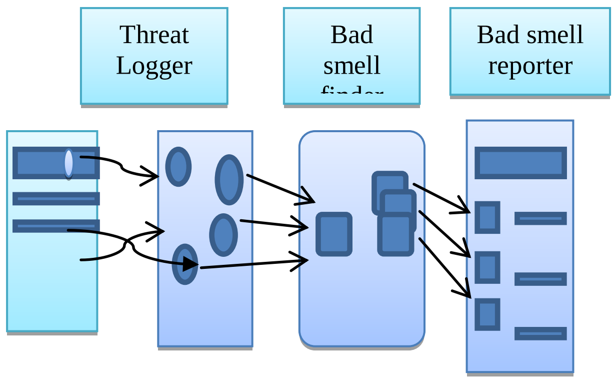

Fig .1: Schematic process of test case generation (Code smell)[11]

It is also called as the reusability tests and can be done in an automated way [9].The detection and testing of the bad smell can be done through refactoring technique. An automated approach can be done in three stages [10]. The threat logger collects the interaction events from the real users. The high level interaction events are determined through reusability events. The bad smell is searched though the bad smell finder. After that, it is stored in bad smell reporter [11].

Test code smell has a different group of smells that is related to the test cases and interrelated to each other. The test classes can be modified by the group of the test cases. The test code can be categorised as[12][13],

1.1 Smell 1: Test Track War

This testing will run as long as one of the testing fails when different programmers are running the code. The temporary resources are allocated. It is difficult to identify the use of the documentation when various test methods are selected [14].

1.2 Smell 2: Lazy Test

It happens when various test techniques select the same method using similar variables. The tests are used when same instance variables are combined to form a structure through inline model approach [15].

1.3 Smell 3: Unintended Testing

The test class is made up to test the complement in the creation of the code. The test class starts to smell when the testing is performed on other objects [16]. The test class can acheived through the extraction and move method. When smell takes place there may be issue in hiding data during the creation of the source code. Its testing process is not easy because the object is segmented in to maximum level.

1.4 Smell 4: For Testers:

The test methods can be used for creating the class. Moreover, code can be produced through the functionality of the test methods [17] [18]. The extract class can be used to replace methods from one class to another class. The developer cannot test the class without adding methods to the class.

1.5 Smell 5: Test Code Doubling:

The undesirable doubling is required for the test code. Particularly, the part of the test code is required to set up the fixed data to resolve the issue of the doubling. The output is similar to the normal code doubling. Generally, the doubling of the test code in similar test class can be eliminated by extraction technique.

2. LITERATURE SURVEY

Liu, H et al., 2019 [19] proposed research on deep learning technique for the detection of the code smell. An automatic selection of the source code for the detection of code smell was done using deep neural network (DNN) and improved deep learning methods. Moreover, an automatic detection was built for complex mapping among such features and calculations. A major challenge was that deep learning needs maximum amount of labels trained information, whereas current database for code smell was small. However, they proposed an automated method for creating labelled trained information for neural network based classifiers that may not require any social intervention. In proposed work, they applied for common and recognized code smells such as feature envy, long technique, maximum class and inappropriate class. The simulation results determined that the open source presentations recommend that planned method was essentially improved the proposed state of art. **Hasanain, W et al., 2018 [20]** performed an industrialized case study that aimed at providing an appropriate understanding like as duplicate test segments which was called as a clone. In this research, the whole test clones were up to 49 % of the LOC. The proposed output includes clone frequencies, kinds, segments and size disseminations and amount of line differences in clone test cases. It was challenging to store the reliable and eliminate un-required clones at the time of the whole testing method of large commercial software. The clone similarity was studied to perform another analysis. They detected that the type -3 reliable and type -3 blind clone segments store the maximum resemblance between 70 to 75%. The amount of the segment existences reduces where similarity of whole type -3 increased. It was analyzed that the differences in the similarity index were 30% that occurred when compared versions of code segments. **Tahmid, A et al.,2016[21]** aimed to identify the development of the code smell in S/W by observing the behavior of the clusters like as size, quantity and connectivity. The clusters and its features were examined. The detection of code smell clusters was done in three stages , identification of code smells through detection equipment, elimination of relation by observing the source code structure, creating graphs from recognised smells and its relations and then discloses the clusters. The observation was computed on JUnit as a case study and behaviour of four clusters was described in this research. The amount of the interrelated code smell clusters in code smell increases gradually with the time. In the case study, the average improvement in cluster count was 3.78 clusters per release. The mega clusters achieved 69.49% of smelly nodes as an average. The simulation results of this research provide appropriate code smell develop in software. **Guggulothu, T et al.,2019 [22]** studied multilevel classification model to identify if the given code component was influenced with multiple code smell or not. They had measured two code smell database and transformed them into a multilevel database. In this research, the two multilevel classification techniques include CC and LC. It was observed that CC technique provides better outcomes as compared to LC method. The proposed method provides better output as compared to current research. The experimental provides accuracy up 91%, whereas it was 73% in current research. **Rubin, J et al.,2019 [23]** presented FAKIE , an automated tool based method to create android code smell detection regulations for performing static analysis of the android applications. The proposed method was a supplement to current code smell detection code. The main objective of the research was the definition of the detection equipment. They used FP –GROWTH connection regulation algorithm to analyse the information created by static analysis. They validated the 48 open source applications from F-Droid. The planned method was automatic identification rules with an average of F-measure of 0.95. Moreover, the empirical study of the 2993 applications of the android applications was done after downloading from Androzoo. The outcome of this research demonstrates the ability to eliminate the data from database of code smell. **Sae-Lim et al., 2017 [24]** examined by expert developers to identify the aspects they used for selecting and arranging the code smells. In this research, they showed an experiment to identify an expert developer to select and organize code smells. The comparative analysis was done in which experts provide maximum priority to code smell linked to specific background. Firstly, task applicability was the major approach considered during code smell selection, followed by Smell severity, and both were measured at the time of the selection method. The other, the essential approach was the main factor in code smell arrangement process, followed by task application. In addition, other aspects like as task application price and collocated smell were measured in both the methods. In this research, the expert developers used code smell filtration and prioritization identification. **Di Nucci, D. et al., 2018[25]**

proposed research on machine learning methods for the detection of code smell , probably resolving the problem of equipment subjectivity giving to learner the capability to distinguish among smelly and non-source code components. The proposed work provided a new opportunity for code smell detection, the single kind of the smell was contained in every database to train and test the machine learners. In this research, they replicate the study with various database configurations which contained the examples of more than kind of the smell.

Table 2, gives the comparative analysis of the various methods, advantages and problems that have been investigated after the survey of different papers.

Table 2: Comparative analysis of methods, merits and issues of various surveyed code smell testing and detection papers

Author	Methods	Advantages	Issues
Liu, H et al., 2019 [19]	Deep learning method	Detecting small code	In detection of misplace class.
Hasanain, W et al., 2018 [20]	Clone analysis method	Analyse industrial test code	Clone fragments
Tahmid, A et al.,2016[21]	Smelly cluster and type checking method	Identify code smell clusters.	Complex structure
Guggulothu, T et al.,2019 [22]	Multilevel classification method	Differentiate smelly and non-smelly code elements.	Identification and maintenance of code smell are difficult.
Rubin, J et al.,2019 [23]	Rule mining based method.	Identify code smell applications	Over using of H/W resources.
Sae-Lim et al., 2017 [24]	Code smell prioritization method	Filter code smell	High evaluation costs
Di Nucci, D. et al., 2018[25]	Machine learning	Detecting small smell	Different code elements

In Table 3 , analysed different parameters after the survey of various papers.

Table 3: Comparative analysis of different parameters analysed from various surveyed papers

Authors/Parameters	Accuracy	Precision	Recall	F-measure	sensitivity
Liu, H et al., 2019 [19]	x	✓	✓	✓	x
Hasanain, W et al., 2018 [20]	✓	x	x	x	x
Tahmid, A et al.,2016[21]	x	x	x	x	✓
Guggulothu, T et al.,2019 [22]	✓	x	x	✓	x
Rubin, J et a.,2019 [23]	✓	✓	✓	x	x
Sae-Lim et al., 2017 [24]	✓	x	x	x	✓
Di Nucci, D. et al., 2018[25]	✓	x	x	✓	x

3. BAD SMELL AND ITS TYPES

Bad smell is also termed as the code smell that is related to the problem in the software design and development of source code [26]. Bad code smell is the poor structure and arrangement during the implementation of the code. The code smell detection tools have been implemented to detect the problems of the program code. The types of the bad smell include;

3.1 Data Class

The complex methods are not present in the data class. The attributes may be public or unprotected through assessor techniques.

3.2 God Class

It is capable to access different attributes from different classes. The attributes present in one class can be used for another class. It contains complicated class methods and large number of the access attributes.

3.3 Feature Envy

It accesses external attributes, the attributes that is present in another class. The feature envy method access large number of the external attributes as compared to local attributes. The less number of the external class is used by the largest number of the attributes.

3.4 Long Method

These are more complex methods that contain a large number of attributes declared in class. The largest of variables used by the attributes are accessed by the accessory. It contains numerous lines of the source code with different number of parameters.

In table 4, the different types of the bad smells are defined such as double code, long method, large class and feature envy, etc. In table 5, some categories of the detectors of the code smell are given.

Table 4: Types of bad smell and its descriptions [31]

Bad smell types	Explanation
Double code	Double display of the code structure
Long method	Complex method
Large class	It contains variable, instance and methods
Large list of the	A complex list of parameters in

parameters	function
Feature envy	Closely coupled class
Clumps of information	Information exist together
Lazy class	The class which may not perform and need elimination
Conflicting variation	Variation in different classes
Artificial design	Complex design
Complicated conditions	Selects unrelated circumstances.
Switch statement	Run time classes are used.
Temporary fields	Variables used rarely
Primitive class	Primitive classes are used

Table 5: Detector related to code smell [27]

A code smell	**Detectors**
God class	iplasma, PMD
Data class	iplasma,fluid tool
Feature envy	iplasma,fluid tool
Long method	iplasma,fluid tool

4. CODE SMELL DETECTION

Smells are the definite structure of the code that determines the destruction of the fundamental design codes and its quality. A code smells are inaccurate design at the time of the development and maintenance of the program but it does not contain bugs. However, the problem in design structure increases the chance of the bugs in lines of the code [28]. Bad smell generally takes place during the generation of code instead of testing code. The refactoring process before the testing escapes the code from bad smells. The testing code is essential when complicated refactoring is done. Code refactoring is the method of reconstructing the computer code that alters the factoring without modifying the exterior behavior. The re-factoring method decreases the complexity and improves the readability of code. An automated unit testing should be arranged before the process of refactoring. The detection, unit testing and refactoring plays an essential role during maintenance and arrangement of the source code. The detection of bad smell is related to sign of problem during the execution of the source code, maintenance and evolution of software. Code smell detection is challenging approach for developers and designing of code smell detection tools and equipment.

4.1 Programmed/Automated Tools of Bad Smell Test Case Generation

Different tools has been established to enhance the quality of the code at the time of the software development and other tools maintains industrialized and maintenance actions. Major tools are refactored in an automated way, but recent IDE is capable of performing the refactoring in an automated way. Some of the tools are categorized as[29],

4.1.1 Check style

It had been established to help programmers, for writing the code of JAVA. It is capable to identify the maximum number of classes, longer methods in class, parameters and double code list.

4.1.2Décor

It is defined as the method which permits the identification and automated recognition of the code and the design of the bad code smell. The custom language automatically creates the detection algorithms through templates, precision and recall. The whole process is done in a Décor platform for analysis of the s/w design.

4.1.3 Infusion

It is the advanced version of the iplasma infusion which is capable to identify more than twenty a design and code smells such a double code, segmented sub-class and god class methods.

4.1.4 iplasma

This tool combines a framework for quality assessment of object oriented classifications which includes provision for essential stages of s/w analysis. This tool is capable to detect the code disagreements, which worked as code smell, feature envy, god class method.

4.1.5 Jdeodorant

It automatically detects the feature envy, god class, long method and type checking in source code of JAVA. It contributes the operator in recognising a suitable series of the refactoring applications that resolve the detected issues, arranges according to structure and relating to the one selected y programmer.

4.2 Machine Learning Detection Methods of Bad Smell

Machine learning techniques can be categorized in to supervised and unsupervised learning. Supervised techniques are used for the detection of the code smell. Some of the methods are [30],

4.2.1 Decision Tree

It is used for the classification of the feature sets. Every node in the decision tree demonstrates the feature in classification and representation of the node. The classified root node is based on the feature values.

4.2.2 Learning Group on Rules

Decision tree may be interpreted into group of rules by generating a separate rule for every path from root to leaf in the tree. Hence, rules can be encouraged from the training information using a variety of rule based algorithm.

4.2.3 One layer Perceptron

The linear separation can be classified through single layer perceptron.

4.2.4 Multilayered Perceptron

The non-linear classification issues may not be resolved by one layer. In addition, more than neuron is associated to form a combined pattern.

4.2.5 Radial Basis Function (RBF) Network

It is three layer network, where every hidden component develops a radial function and every output component developed weigh sum of hidden components.

4.2.6 Naïve Bayes

The Bayesian network contains graphs with one unnoticed node and sequence of identifying as a node with the notion of state free between root and child node. It depends on guessing unidentified node that depends on identified nodes.

4.2.7 Support Vector Machine

It depends on the perception of boundary on either side of hyper plane distinguishing two features. Its goal is to develop the boundary and generate maximum distance between the features in hyper plane. The amount of the features is not affected by the complexity structures.

5. CONCLUSION AND FUTURE SCOPE

In conclusion, surveyed on different aspects of testing and detection of the bad smell in various test case generation process. Test case generation for bad smells are mainly referred as the poor designed tests and its main impact on the generation of the source code and quality of the test suites. Test code smell is the main area of the considerations, among the researchers in smell detection, and prevention. The software representations can be used to recognize the bad smell in the code which may lead to regular failures of the code. In case the bad smell is unidentified, it may consume maximum resources in term of management costs, testing and so forth. Code smell is related to the complex issue in designing of the source code which lead to problem in maintenance and evolution of the software. This paper proposes a study on the detection of the bad smell in the software that is also called as code smell. The object oriented software representations have been used to identify the bad smell in source code. The identified categories of the bad smell are large class,long method, lazy class, switch statement, parameter list, and primitive fields. Generally, bad code smell detection is challenging approach for researchers and programmers. However, this paper explained detection tools and methods along with comparison tables. Some of the detection tools are given as Infusion, deodorant, check style and Décor. Moreover, a comparison table of the detectors and types of the bad code smell is also given. Besides, some of the detection methods of machine learning methods are explained.

In Future Scope, the programmers must be focused on the modifications of the classes and concerned about the errors in program code. However, various detection methods for detecting bad code smell has been recommended by various researchers. Despite, there has not been developing automated tools for testing and detection of the bad code smell. Hence, there must be work done on testing and detection framework of test code smell.

6. REFERENCES

[1] Sharma, P and Kaur, E. A , "Design of testing framework for code smell detection (OOPS) using BFO algorithm," International Journal of Engineering & Technology, vol 7(2.27), pp 161-166, 2018.

[2] Garousi, V and Küçük, B , "Smells in software test code: A survey of knowledge in industry and academia," Journal of systems and software, 138, pp 52-81, 2018.

[3] Tufano, M., Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A and Poshyvanyk, D, " Towards Automated Tools for Detecting Test Smells: An Empirical Investigation into the Nature of Test Smells,".

[4] Khomh, F., Di Penta, M. and Gueheneuc, Y. G , "An exploratory study of the impact of code smells on software change-proneness," In 2009 16th Working Conference on Reverse Engineering (pp. 75-84), IEEE, 2009.

[5] Neukirchen, H and Bisanz, M. , "Utilising code smells to detect quality problems in TTCN-3 test suites," In Testing of Software and Communicating Systems ,pp. 228-243, Springer, Berlin, Heidelberg, 2007.

[6] Danphitsanuphan, P and Suwantada, T , "Code smell detecting tool and code smell-structure bug relationship," In 2012 Spring Congress on Engineering and Technology (pp. 1-5). IEEE, 2012.

[7] Chatzigeorgiou, A.and Manakos, A. , "Investigating the evolution of bad smells in object-oriented code," In 2010 Seventh International Conference on the Quality of Information and Communications Technology (pp. 106-115), IEEE, 2010.

[8] Chatzigeorgiou, A and Manakos, A , "Investigating the evolution of code smells in object-oriented systems," Innovations in Systems and Software Engineering, 10(1), 3-18, 2014.

[9] Grigera, J., Garrido, A and Rivero, J. M. , "A tool for detecting bad usability smells in an automatic way," In International Conference on Web Engineering (pp. 490-493). Springer, Cham, 2014.

[10] Van Rompaey, B., Du Bois, B., Demeyer, S and Rieger, M , "On the detection of test smells: A metrics-based approach for general fixture and eager test," IEEE Transactions on Software Engineering, 33(12), 800-817, 2007.

[11] Usha, K., Poonguzhali, N and Kavitha, E , "A quantitative approach for evaluating the effectiveness of refactoring in software development process," In 2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS) (pp. 1-7). IEEE, 2009.

[12] Van Deursen, A., Moonen, L., Van Den Bergh, A and Kok, G , " Refactoring test code," In Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001) ,pp. 92-95, 2001.

[13] Guerra, E. M. and Fernandes, C. T , "Refactoring test code safely," In International Conference on Software Engineering Advances (ICSEA 2007) (pp. 44-44), IEEE, 2007.

[14] Atkins, F. J and Coe, P. J , "An ARDL bounds test of the long-run Fisher effect in the United States and Canada," Journal of Macroeconomics, 24(2), 255-266, 2002.

[15] Chen, W. K. and Wang, J. C , "Bad smells and refactoring methods for gui test scripts," In 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing , pp. 289-294, IEEE, . 2012.

[16] Palomba, F., Di Nucci, D., Panichella, A., Oliveto, R and De Lucia, A , "On the diffusion of test smells in automatically generated test code: An empirical study," In Proceedings of the 9th international workshop on search-based software testing (pp. 5-14),ACM, 2016.

[17] Fontana, F. A., Dietrich, J., Walter, B., Yamashita, A and Zanoni, M , "Antipattern and code smell false positives: Preliminary conceptualization and classification," In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) ,Vol. 1, pp. 609-613, IEEE, 2016.

[18] Danphitsanuphan, P and Suwantada, T., "Code smell detecting tool and code smell-structure bug relationship," In 2012 Spring Congress on Engineering and Technology ,pp. 1-5, IEEE, 2012.

[19] Liu, H., Jin, J., Xu, Z., Bu, Y., Zou, Y and Zhang, L , "Deep learning based code smell detection," *IEEE Transactions on Software Engineering,* 2019.

[20] Hasanain, W., Labiche, Y and Eldh, S, "An analysis of complex industrial test code using clone analysis," In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 482-489). IEEE, 2018.

[21] Tahmid, A., Nahar, N and Sakib, K, "Understanding the evolution of code smells by observing code smell clusters," In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* ,Vol. 4, pp. 8-11,IEEE, 2016.

[22] Guggulothu, T , "Code Smell Detection using Multilabel Classification Approach," *arXiv preprint arXiv:1902.03222*, 2019.

[23] Rubin, J., Henniche, A. N., Moha, N., Bouguessa, M. and Bousbia, N , " Sniffing Android Code Smells: An Association Rules Mining-Based Approach," In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pp. 123-127,IEEE, 2019.

[24] Sae-Lim, N., Hayashi, S and Saeki, M , "How do developers select and prioritize code smells? a preliminary study," In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)* ,pp. 484-488), IEEE, 2017.

[25] Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A and De Lucia, A , "Detecting code smells using machine learning techniques: are we there yet?," In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* ,pp. 612-621, IEEE, 2018.

[26] Fernandes, E., Oliveira, J., Vale, G., Paiva, T and Figueiredo, E, "A review-based comparative study of bad smell detection tool,". In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (p. 18). ACM, 2016.

[27] Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A. and De Lucia, A , "Detecting code smells using machine learning techniques: are we there yet?," In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* , pp. 612-621, IEEE, 2018.

[28] Fontana, F. A., Braione, P and Zanoni, M , " Automatic detection of bad smells in code: An experimental assessment," *Journal of Object Technology*, *11*(2), 5-1, 2012.

[29] Mathur, N and Reddy, Y. R , "Correctness of Semantic Code Smell Detection Tools," In *QuASoQ/WAWSE/CMCE@ APSEC* (pp. 17-22), 2015.

[30] Caram, F. L., Rodrigues, B. R. D. O., Campanelli, A. S and Parreiras, F. S , "Machine Learning Techniques for Code Smells Detection: A Systematic Mapping Study," *International Journal of Software Engineering and Knowledge Engineering*, *29*(02), 285-316, 2019.

[31] Umesh, I. M and Srinivasan," G. N , "A study on bad code smell," International Journal of Latest Technology in Engineering, Management & Applied Science,vol 4(5), 2015.