International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 4 Issue: 4
ISSN: 2454-4248
425 – 430

Code Smell Detection Techniques and Process: A Review

Pratiksha Sharma (Author)
Department of Computer Science & Engineering
Chandigarh University, Gharuan Punjab, India
e-mail: pratikshasharma21192@gmail.com

Er. Arshpreet Kaur *(Author)*
Department of Computer Science & Engineering
Chandigarh University, Gharuan Punjab, India
e-mail: arshpreet.cse@cumail.in

***Abstract*—** A code smell is a hint that something has turned out badly some place in your code. The idea of code smells was introduced to characterize various different types of design shortcomings in code. Code and design smells are poor solutions to recurring implementation and design problems. They may hinder the evolution of a system by making it hard for software engineers to carry out changes. In this paper, we reviewed code smell detection tool like: Décor, InFusion, JDeodorant, PMD, Stench Blossom, etc. Furthermore, we discussed various code smells detecting techniques. Code clones are indistinguishable fragment of source code which may be embedded deliberately or inadvertently. Reusing code pieces through reordering with or without minor adjustments is general undertaking in programming advancement. We've examined several papers to explore various tools and techniques used for code smell. In addition, we reviewed the process of code smell detection.

Keywords- *Code Smell, Detection tool (i.e, Infusion and Deodorant), fragment of source code.*

*****

I. INTRODUCTION *(HEADING 1)*

Software maintenance is last and big-budgeted period of SDLC[1]. The major concern behind product support is the change of current programming framework by including new functionalities, to rectify errors in the product framework or because of the new necessities of the association that are not distinguished amid the prerequisite stage. Yet, the most extreme endeavours are required while expanding the current programming by including new functionalities. One of the systems utilized for programming support is Software Re-engineering [2] is the most utilized.

1.1 CODE SMELL

The word "code smell" was presented by Kent Beck to define those structural problems in the source code that can be detected by experienced developers. As written by Kent Beck [3]: "A code smell is a hint that something has gone wrong anywhere in your code".

The uncertain structure may not be causing serious harm (in terms of bugs and failures) at the moment, but it has a negative impact on the overall structure of the system and as on sequence, on its quality factors. Code smells can clutter the design of a system, making it harder to understand and maintain. Moreover, the attendance of code smells can warn about wider development difficulties such as wrong architectural [4] choices or even bad management practices.

Code smells are structural characteristics of software that may specify a code or design problem and can make software hard to evolve and maintain. The concept was introduced by Fowler, who de?ned 22 different kinds of smells [5]. Code smells are strictly related to the practice of refactoring software to enhance its internal quality [6]. As engineers recognize terrible stenches in code, they ought to assess whether their event indications at some pertinent corruption in the structure of the code, and if positive, choose which refactoring ought to be practical. Using a symbol, smells are like the symptoms of possible diseases, and Refactoring. Operations may heal the linked illnesses and remove their symptoms.

1.1.1 Bad Smells in Code

Code smell is any manifestation that demonstrating something incorrectly. It for the most part demonstrates that the code ought to be refectories or the total outline ought to be reevaluated. The term appears to have been coined by Kent Beck[4] .Usage of the term enlarged after it was contained in Refactoring. Bad code exhibits certain characteristics that can be rectified using Refactoring. These are called Bad Smells[3].

- Long Method: when method is too long means more number of lines of code.
- Large Class: Modules that have large numbers of instance variables and large number of lines of code. Occasionally they are only used infrequently large classes can also suffer from code duplication.
- Long Parameter List: Long constraint lists are hard to recognise. Long parameter list means that a method takes too many parameters.
- Comments: If the explanations are present in the code more than the lines of code.
- Switch Statements: Switch statements may harvest duplication. You can find comparable switch statements scattered in the program in several places. . Maybe classes and polymorphism would be more proper.
- Lazy Class: Courses that are not doing much work and number of method is null.
- Temporary Field: when some of the illustration variables in a class are only used occasionally [5].

 ISSN: 2454-4248
425 – 430

- Duplicate Code: The equivalent code arrangement in two or more places is a good sign that the code need to be refectories.
- Dead Code: It's a segment in source code of a program which is executed but output is never used in any other multiplication. The completing of dead code wastes computation time and memory.

II. LITERATURE REVIEW

ThanisPaiva, et al., (2017) [7] evaluated and compared four code notice identification apparatuses, to be specific inFusion, JDeodorant, PMD, and JSpIRIT. Code smells allude to any side effect in the source code of a program that conceivably shows a more profound issue, ruining programming support and advancement. Identification of code smells is trying for engineers and their casual definition prompts the usage of various recognition systems and apparatuses. These apparatuses were connected to various adaptations of similar programming frameworks, specifically Mobile Media and Health Watcher, to figure the exactness and understanding of code notice discovery instruments. They ascertained the exactness of each device in the recognition of three code smells: God Class, God Method, and Feature Envy. Understanding was ascertained among devices and between sets of apparatuses. One of our primary discoveries is that the assessed devices display distinctive levels of precision in various settings. For MobileMedia, with respect to assention, we found that the general understanding between instruments differs from 83 to 98% among all devices and from 67 to 100% between sets of devices. We additionally directed an auxiliary investigation of the advancement of code smells in both target frameworks and found that, by and large, code smells are available from the snapshot of formation of a class or strategy in 74.4% of the instances of MobileMedia and 87.5% of Health Watcher.

E. Kodhai, et al (2016) [8] presented an incremental clone detection along with hybrid approach to locate clones in multiple alterations of program. This hybrid technique is a merger of metrics computation and textual analysis. In period of last ten years, considerable research effort was made for detection and expulsion of clones from software framework. However, some practical tools are available for programming languages. Majority of techniques used for clone detection are limited one alteration of program. Both techniques of clone detection and modification functionalities are united with Clone Manager, is a tool for Java and C programs. This incremental technique is an improved feature to Clone Manager tool. They examined the improved Clone Manager tool with parameters recall ratio and precision for 6 open source projects.

Dongjin Yu, et al., (2017) [9] proposed a novel technique of code clone detection based on Java bytecode. Code clones are commonly believed as unwanted for many reasons, despite of ease provided to developers. Identification of code clones improvise the quality of source code via software re-engineering. Several methods were proposed in Java source code while just few concerned to its bytecode. The Java bytecode displays semantic nature of code. Using the block-level code fragments extracted from bytecode, and simultaneously identify code clones at both method level and block level. During code clone detection process the similarities of instruction sequences and call sequences are calculated to enhance accuracy and performance. The results prove that proposed method is more effective than existing methods.

Yingnong Dang, et al., (2017) [10] described the encounter of shifting XIAO, a code-clone detection and analysis approach and supporting tool, to wide industrial practices i.e., (1) shipped in Visual Studio 2012, a broadly used industrial IDE; (2) deployed and intensively used at the Microsoft Security Response Centre. Amid programming improvement, code clones are normally delivered, as some of the same or comparative code pieces spreading inside one or numerous expansive code bases. Various research ventures have been done on experimental investigations or apparatus bolster for distinguishing or dissecting code clones. Nonetheless, practically speaking, couple of such research ventures have brought about generous industry adoption. According to our encounters, innovation exchange is a fairly confounded excursion that requirements huge endeavours from both the specialized viewpoint and social perspective. From the specialized perspective, huge endeavours are expected to adjust an examination model to an item quality device that tends to the requirements of genuine situations, to be coordinated into a standard item or advancement process. From the social viewpoint, there are solid needs to cooperate with professionals to recognize executioner situations in mechanical settings, make sense of the hole between an examination model and an apparatus fitting the necessities of genuine situations, to comprehend the prerequisites of discharging with a standard item, being coordinated into an improvement procedure, understanding their discharge rhythm, and so forth.

ShrutiJadon, (2016) [11] proposed to create a feature set by analysing C program for fragments of code and matching similarities. Code clones characterized as succession of source code that happen more than once in a similar program or crosswise over various projects are unfortunate as they increment the span of program and makes the issues of excess. Settling of bugs recognized in one clone require discovery of all clones. Henceforth, it is basic to recognize and evacuate all code clones in a program. The concentrate of past research chip away at the code clone location was to discover indistinguishable clones, or clones that are indistinguishable up to identifiers and strict esteems. Be that as it may, identification of comparable clones is regularly essential. Based on highlight sets the grouping of calculation is being performed by utilizing the Support Vector Machine (SVM) as a machine learning

 ISSN: 2454-4248
425 – 430

apparatus. The yield of the machine device would be the closeness proportion with which the two C programs are identified with each other and furthermore the class in which they would happen. It was watched that the test consequences of the instrument execution indicate identification of code clones in the program and its exactness increments with the expansion in number of occurrences.

Abdullah Sheneamer et al., (2015) [12] presented a hybrid technique which utilised a coarse grain method to break down the clones efficiently to enhance precision. In the event that two parts of source code are indistinguishable to each other, they are called code clones. Code clones present challenges in programming upkeep and cause bug engendering. Coarse-grained clone indicators have higher accuracy than fine-grained, yet fine-grained identifiers have higher review than coarse-grained. In this manner, we utilize a fine-grained identifier to get extra data about the clones and to enhance review. Our technique distinguishes Type-I and Type-2 clones utilizing hash esteems for pieces, and gapped code clones (Type-3) utilizing square discovery and resulting examination between them utilizing Levenshtein separation and Cosine measures with changing limits.

III. CODE SMELL DETECTION TOOLS

- Check style [13] Checkstyle2 has been industrialised to help computer operator to write Java code that adheres to a coding standard. It is able to perceive the Large Class, Long Method, Long Parameter List, and Duplicated Code smells.
- Décor [14] defined an approach that allows the specification and automatic detection of code and design smells (also named anti patterns). They quantified six code smells by using a custom language, automatically generated their detection algorithms using patterns, and authorized the algorithms in terms of precision and recall. Decor platform for software analysis is an application to the Decor Tool. In the following, with the name Decor we mean the component developed for code smell detection.
- InFusion It is current, commercial evolution of I Plasma [5]. InFusion is able to sense more than 20 design errors and code smells, like Duplicated Code, classes that break encapsulation, i.e. Data Class and God Class, approaches and classes that are seriously coupled, or ill-designed class hierarchies.
- I Plasma This tool [13] is a combined platform for quality calculation of object-oriented systems that includes support for all the necessary phases of examination, from model abstraction, up to high-level metrics based analysis. IPlasma5 is able to detect what the authors define as code disharmonies, classified into identity disharmonies, collaboration disharmonies, and classification disharmonies. Code smells like Repeated Code (named Important Duplication), God Class, Feature Envy, and Refused Parent Bequest, etc are considered as disharmonies.
- JDeodorant [15] is an Eclipse plugin that automatically identifies the Feature Envy, God Class, Long Method and Switch Statement (in its Type Checking variant) code smells in Java programs[16]. The tool assists the user in determining an appropriate arrangement of refactoring requests by determining the possible refactoring transformations that solve the identified difficulties, ranking them bestowing to their impact on the design, presenting them to the developer, and automatically applying the one selected by the developer.
- PMD [17] scans Java source code and looks for potential problems or possible bugs like dead code, empty try/catch/finally/switch statements, unused local variables or parameters, and duplicated cipher. PMD is competent to detect Large Class, Long Method, Long Parameter List, and Duplicated Code smells, and allows the user to set the inceptions values for the oppressed metrics.
- Stench Blossom [18] is smell detector supplies an interactive visualization framework designed to give programmers a quick and high level overview of the scents in their code, and of their derivation. The device is a module for the Eclipse condition that gives the developer three disparate perspectives, which progressively offer more data about the odors in the code being imagined. The reaction is synthetic and visual, and has the shape of a set of petals close to a code element in the IDE editor. The size of a petal is directly relative to the “strength” of the smell of the code element it refers. The only possible technique to find code smells is to physically browse the source code, looking for a petal whose size is big enough to make the user supposing that there is a code smell. The tool is able to detect 8 smells.

 ISSN: 2454-4248
425 – 430

Table 1: Code Smell Detection Tools

Tool	Version (Year)	Type	Analysed Language	Refactoring
Checkstyle	5.4.1 (2011)	Eclipse Plugin, Standalone	Java	No
Décor	1.0 (2009)	Standalone	Java	No
iPlasma	6.1 (2009)	Standalone	C++, Java	No
inFusion	7.2.11 (2010)	Standalone	C, C++, Java	No
JDeodrant	4.0.4 (2010)	Eclipse Plugin	Java	Yes
PMD	4.2.5 (2009)	Eclipse Plugin, Standalone	Java	No
Stench Blossom	1.0.4 (2009)	Eclipse Plugin	Java	No

IV. Code Smell Detection Process

Each smell is a manifestation that shows the infringement of programming plan standards, for example, seclusion, reflection, epitome, chain of command, and modifiability [19]. Programmed identification of code smells from source code is a key to programming refactoring, upkeep, and quality confirmation of source code. Diverse systems and apparatuses apply distinctive techniques for the location of code smells. The bland code notice recognition process may take after advances (not really all) as delineated in Figure 1. Code notice recognition strategies take source code or broke down source code in various portrayals and details of code smells as info. Code notice details are coordinated with analysed source code by utilizing programming measurements, or diverse different procedures and occasions of various odours are recuperated. A few procedures additionally imagine recouped smells and furthermore bolster refactoring of recuperated smells. The vast majority of the code notice discovery procedures utilize existing item arranged source code measurements that are separated from other programming instruments. The exactness of measurements construct methods is needy with respect to the correct choice of source code measurements and their understanding. Haralambiev et al. [20] additionally understood that measurements construct systems need direction in light of the understanding of measurements. A few systems register specifically to source code measurements by performing static investigation on the source code and afterward utilize these measurements for recognizing code smells. In any case, not all code scents can be identified with static examination of source code [21]. A few procedures apply a blend of static and dynamic examination techniques on source code and process source code measurements that are utilized for the location of code smells. Besides, it is additionally clear from the writing that not all code scents can be identified with just the investigation of source code, for example, parallel legacy or shotgun surgery. The forming data is required to distinguish such smells.

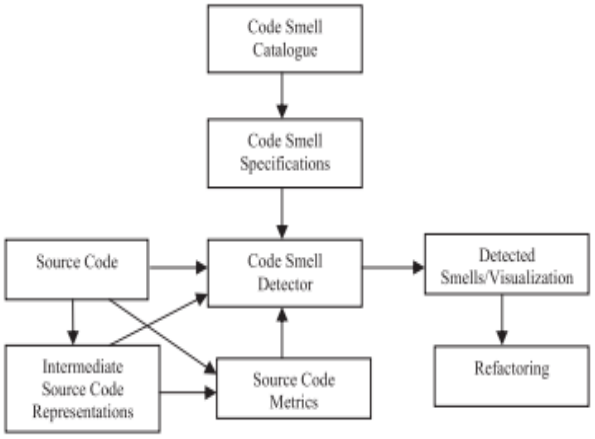

Figure 1: Generic Code Smell Detection Process [22]

V. Code Smell Detection Technique

A key element for correlation of code smells identification systems is the utilization of one ormore recognition procedures. We quickly talk about code smelldetection procedures in this sub-section. Earlier manual techniques were used to detect code smell design principles. Such procedures are manual, prone to error and time consuming and less effective in identification of code smell in bigger systems [22]. Numerous code smell discovery strategies and devices apply source code measurements for recuperation of code smells from the source code are discussed below:

- **Metrics based techniques** are restricted just to the location of code notices that are relatively simple to recognize. These methods are comparable in ideas as they rely upon source code measurements, however they contrast by the way they apply measurements and what kinds of code smells they centre around. The precision of measurements based code notice location procedures is reliant on the correct determination of limit esteems, which are generally observational and untrustworthy.
- **Symptoms based techniques** utilize diverse manifestations/documentations that are converted into recognition calculations. The transformation of side effects into recognition rules requires examination and translation push to choose

 ISSN: 2454-4248
425 – 430

appropriate limit esteems. The impediment of these strategies is that there is still no accord on characterizing standard side effects with similar elucidations. The exactness of these systems is low a result of the distinctive translations of similar side effects.

- **Visualization based code smell identification techniques** utilize the semi-robotized process for the discovery of code smells. These procedures incorporate the ability of human mastery with the mechanized identification process. The impediment of these procedures is human exertion, and they have versatility issues for vast frameworks. These systems are additionally blunder inclined as a result of wrong human judgment.
- **Search based code smell detection technique** apply diverse calculations for the identification of code smells specifically from source code. Most strategies in this class apply machine learning calculations. These procedures gain from standard outline and coding rehearses and looks at how code digresses from these practices. The accomplishment of these methods relies upon quality informational collections and their preparation. These strategies have impediments for managing obscure and changing meanings of code smells.
- **Co-operative based techniques** have the inspiration to perform distinctive exercises helpfully to improve execution of exercises. The helpful based methods are moderately new, and they enhance exactness and execution for code smell detection. The first calculation produces identification, and the second calculation creates finders. The two calculations depend on hereditary programming. Speculation of approach for the location of different kinds of code smells is sketchy.
- **Probabilistic-based code smell detection technique** apply fuzzy rationale decides that incorporate quantitative properties and connections among classes. These methods rank hopeful odours utilizing fluffy rationale induction guidelines and handle vulnerability in the code notice identification process. [23] Introduced a factual examination based method to distinguish five code smells.

VI. CONCLUSION

A code smell is an insight that something has turned out seriously some place in your code. Code smells was acquainted with portray different distinctive kinds of outline inadequacies in code. Code and configuration smells are poor answers for repeating usage and plan issues. They may thwart the development of a framework by making it hard for programming architects to complete changes. In this paper, we checked on code notice identification instrument like: InFusion, JDeodorant, PMD, Stench Blossom, and so on. Besides, we talked about different code smells distinguishing strategies. Code clones are vague piece of source code which might be inserted intentionally or unintentionally. Reusing code pieces through reordering with or without minor alterations is general endeavour in programming progression. We've inspected a few papers to investigate different devices and procedures utilized for code smell. Also, we assessed the procedure of code notice identification.

REFERENCES

[1] R. Koschke, Survey of research on software clones, in: Duplication, Redundancy, and Similarity in Software, Dagstuhl Seminar Proceedings, 2007, p. 24.

[2] C.K. Roy, J.R. Cordy, A Survey on Software Clone Detection Research, Technical Report 2007-541, Queen's University at Kingston Ontario, Canada, 2007, p. 115.

[3] Nongpong, K. (2012). Integrating" code smells" detection with refactoring tool support (Doctoral dissertation, The University of Wisconsin-Milwaukee).

[4] Ito, Y., Hazeyama, A., Morimoto, Y., Kaminaga, H., Nakamura, S., &Miyadera, Y. (2014, August). A Method for Detecting Bad Smells and ITS Application to Software Engineering Education. In Advanced Applied Informatics (IIAIAAI), 2014 IIAI 3rd International Conference on (pp. 670-675). IEEE.

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

[6] Du Bois, B., Demeyer, S., &Verelst, J. (2004, November). Refactoring-improving coupling and cohesion of existing code. In Reverse Engineering, 2004. Proceedings. 11th Working Conference on (pp. 144-151). IEEE.

[7] Paiva, Thanis, Amanda Damasceno, Eduardo Figueiredo, and CláudioSant'Anna. "On the evaluation of code smells and detection tools." Journal of Software Engineering Research and Development 5, no. 1 (2017): 7.

[8] Kodhai, Egambaram, and SelvaduraiKanmani. "Method-level incremental code clone detection using hybrid approach." International Journal of Computer Applications in Technology 54, no. 4 (2016): 279-289.

[9] Yu, Dongjin, Jie Wang, Qing Wu, Jiazha Yang, Jiaojiao Wang, Wei Yang, and Wei Yan. "Detecting Java Code Clones with Multi-granularities Based on Bytecode." In Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual, vol. 1, pp. 317-326. IEEE, 2017.

 ISSN: 2454-4248
425 – 430

[10] Dang, Yingnong, Dongmei Zhang, Song Ge, Ray Huang, Chengyun Chu, and Tao Xie. "Transferring code-clone detection and analysis to practice." In Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track, pp. 53-62. IEEE Press, 2017.

[11] Jadon, Shruti. "Code clones detection using machine learning technique: Support vector machine." In Computing, Communication and Automation (ICCCA), 2016 International Conference on, pp. 399-303. IEEE, 2016.

[12] Sheneamer, Abdullah, and JugalKalita. "Code clone detection using coarse and fine-grained hybrid approaches." In Intelligent Computing and Information Systems (ICICIS), 2015 IEEE Seventh International Conference on, pp. 472-480. IEEE, 2015.

[13] Fontana, Francesca Arcelli, PietroBraione, and Marco Zanoni. "Automatic detection of bad smells in code: An experimental assessment." Journal of Object Technology 11, no. 2 (2012): 5-1.

[14] NaouelMoha, Yann-GaëlGuéhéneuc, Laurence Duchien, and AnneFrançoise Le Meur. DECOR: A method for the specification and detection of code and design smells. IEEE Transactions on Software Engineering, 36(1):20–36, January–February 2010. doi:10.1109/TSE. 2009.50.

[15] Abdelmoez, W., Kosba, E., &Iesa, A. F. (2014, January). Risk-based code smells detection tool. In The International Conference on Computing Technology and Information Management (ICCTIM) (p. 148). Society of Digital Information and Wireless Communication.

[16] Sreenu, K., & Rao, D. J. Performance-Detection of Bad Smells In Code for Refactoring Methods

[17] Pessoa, T., Monteiro, M. P., & Bryton, S. (2012). An eclipse plugin to support code smells detection. arXiv preprint arXiv:1204.6492.

[18] Emerson Murphy-Hill and Andrew P. Black. An interactive ambient visualization for code smells. In Proceedings of the 5th international symposium on Software visualization, SOFTVIS '10, pages 5–14, Salt Lake City, Utah, USA, 2010. ACM. doi:10.1145/1879211.1879216.

[19] Peters R, Zaidman A. Evaluating the lifespan of code smells using software repository mining. In proceedings of 16th European conference on Software Maintenance and Reengineering(ICSMR), pp. 41–416, 2012.

[20] Haralambiev H, Boychev S, Lilov D, Kraichev K. Applying source code analysis techniques: a case study for a large mission-critical software system. In Proceedings of International Conference on Computer as a Tool, pp. 2–3, 2011.

[21] Ligu E, Chatzigeorgiou A, Chaikalis T, Ygeionomakis N. Identification of refused bequest code smells. In Proceedings of IEEE International Conference on Software Maintenance(ICSM), pp.392-395, 2013.

[22] Rasool, Ghulam, and Zeeshan Arshad. "A review of code smell mining techniques." Journal of Software: Evolution and Process 27, no. 11 (2015): 867-895.

[23] Mathur, Nitin. "JAVA SMELL DETECTOR." (2011).