East African Scholars Journal of Engineering and Computer Sciences
Abbreviated Key Title: East African Scholars J Eng Comput Sci
ISSN: 2617-4480 (Print) & ISSN: 2663-0346 (Online)
Published By East African Scholars Publisher, Kenya
Volume-3 | Issue-10 | Dec-2020 | DOI: 10.36349/easjecs.2020.v03i10.004

OPEN ACCESS

Review Article

Influence of Anti-Patterns Detection Techniques on Software Maintenance

Dr. Manjeet Singh[1*], Er. Abhinash Singla[1], Er. Amreen Kaur[1], Er. Beant Kaur[1]
[1]Assistant Professors, Department of CSE, BGIET Sangrur, Punjab, India

Article History
Received: 21.11.2020
Accepted: 25.12.2020
Published: 30.12.2020

Journal homepage:
https://www.easpublisher.com

Quick Response Code

Abstract: Anti-patters are the defects which affect the system quality negatively. An indication of the existence of anti- patterns, in the software is known as ―Code Smell‖ which leads to the refactoring of system. Thus the maintenance becomes difficult to manage. More the number of smells more refactoring is needed. Different approaches have been identified for the detection of anti-patterns in the system. The paper aimed at investigating the impact of anti-patterns on classes and what are the certain kinds of anti-patterns that have a higher impact than others finally the results have been concluded for the future studies in open source systems. The paper is divided in to four sections in which the introduction is followed by the types of anti-patterns. Furthermore the related work has been examined carefully with a brief conclusion. Thus the paper reveals different approaches for the identification code smells in the software system. Hence the detection of smells will be helpful in providing more reliability during testing and maintenance phases by predicting anti-patterns and faults before the delivery of the product. Moreover the identification of anti-patterns will be of usage to the community of software engineers and managers for improving the software development maintenance activities.

Keywords: Anti-patterns, Code Smells, Refactoring and Maintenance.

1. INTRODUCTION

Software systems require continuous maintenance and intelligent development for achieving good quality. Quality can be improved by various quality assurance activities like formal technical reviews, testing and enforcement of standards. Every time when new software is to be developed; already present capital (source code documents, design templates) is used in the development process. But in most of the cases the reusability of software components degrades the performance and quality of the software. Some common examples of software reusability are: Software Libraries, Design Patterns, Frameworks, Systematic Software Reuse. In software designing when a problem is occurring time and again than a typical solution is needed and a way is provided by the software developers by introducing the term – Design Patterns. But in majority of cases the design patterns start performing like ―ANTI-PATTERNS‖. Such design defects are called Code Smells or Anti-Patterns. In case code smells are present maintenance becomes the need of the system. The code smell is referred as an indication of the presence of anti-patterns in the system. More the number of smells more maintenance is needed. Anti-Patterns are defined as a classical style which provide a solution of the trouble which could generates negative results. Hence, anti-patterns are treated to be the negative solutions that yield more problems than they locate. Therefore the system starts demanding Refactoring [6].

The purpose of the paper is to disclose the commonly occurring anti-patterns in system which is generally specified in [5] and [6]. The work done will be helpful to the community of software engineers and managers for improving the software development maintenance activities.

2. ANTI-PATTERNS IN SOFTWARE SYSTEM

The software quality is characterized by good design. The design crumbles with the passage of time as changes are made in the structure due to changing user requirements. The defects of the software disclose themselves in the form of ―Anti-patterns‖ or ―Code Smells‖ [6]. There is a very fine line between the anti-pattern and code smell. Anti-patterns are considered to be a bad programming practice but not an error. Due to lack of experience and relevant knowledge of the software developers anti-patterns are introduced to the literature, for solving a specific problem. Code Smell is a manifestation that indicates the problem of software system [5]. It is an implication that admits anti-patterns.

***Corresponding Author:** Dr. Manjeet Singh

However code smells are technically not wrong but they indicate delicacy in design. It may lead to the failure of system and risk of bugs in future. Hence system demands ―Refactoring‖– changing the existing software code without affecting the external behavior. Nearly 20 code smells are specified in [6] which are embodied in the source code where refactoring of code is needed. The following types of anti-patterns/code smells are suggested by the following author.

Table 1: Types of Anti-Patterns

Author Name	Anti-pattern/Code Smells
M. Fowler [6]	Lazy Class, Large Class, Long Method, Long Parameter List, Message Chain, Duplicate Code, Divergent Change , Shot Gun Surgery, Feature Envy, Data Clumps, Primitive Obsession, Switch Statements, Parallel Inheritance Hierarchies, Middle Man, Speculative Generality, Inappropriate Intimacy, Temporary Field
B.F. Webster [1] and W. J. Brown *et al*., [5]	Blob, Spaghetti Code, Conditional Complexity, Anti-Singleton, Class Data Should Be Private(CDSBP), Refused Parent Bequest (RPB), Swiss Army Knife

3. RELATED WORK

Anti-patterns are derived from work on patterns. These are considered to be a poor design choices but not an error. Code Smells is an evidence of the presence of anti-patterns. The term anti-pattern was originated by Andrew Koenig, who introduces the perception of patterns in the software engineering [3]. Both the terms anti-patterns and code smells are used interchangeably.

Several authors studied the influence of anti-patterns and code smells in the software systems. The work explores the anti-patterns and code smells, in context to software engineering activities. The first book on smells was written by Webster [1] who includes risks of quality assurance, coding, etc. According to [5] attention is needed towards object oriented systems. More than 35 smells were specified in [5] which include the well-known design smell ―BLOB‖. The concept of refactoring was presented in [6], due to the presence of nearly 20 code smells. The author provided a detailed insight of term ―Refactoring‖ which is a process of altering the software system without changing its external behavior. The author revealed different types of code smells like Lazy Class, Long Method, Long Parameter List, Shotgun Surgery, etc. The above mentioned authors provide a detailed knowledge on code smells and anti-patterns. However the approach followed by the authors for the identification of anti-patterns was completely manual. It was a time consuming and error prone job for large projects. Thus some researchers proposed the automatic, visualization based and statistical detection techniques.

3.1 Traditional Detection Techniques

In the traditional detection techniques the researchers introduced the different methods using which anti-patterns and code smells can be identified in the system manually. This was the first step towards the future of semi-automatic and automatic anti-pattern detection techniques. It varies from software reading method to metric based and template driven approach. Some of the approaches are discussed here:

The manual detection strategy [31] was suggested by Travassos and F. Shull [7]. It was a software reading technique which provided assistance in detection of smells in Object Oriented Systems. Different types of Reading Techniques had been included for the purpose of detection i.e. Defect Based Reading, Perspective Based Reading, Use Based Reading. A project was developed by the team of students which was reviewed on two factors: Horizontal Review and Vertical Review. The approach was completely manual and time consuming.

Connie U. Smith and Lloyd G. William [8] investigated the affect of anti-pattern God Class on the system and show how to solve it. They also proposed three new performance anti-patterns that often occur within the software systems R. Marinescu [14] introduced a metric based approach for detection of anti-patterns. The technique was realized on tool plasma More than 8 anti-patterns were detected with nearly same number of techniques. The threshold values were compared against values of metrics combined with set operators.

M. J. Munro [16] suggested a template driven model to detect anti-patterns. The template consists of three components: Name of smell, Description of the properties of code smell in text format, Heuristics for the detection of smells. He explored the product metrics for the apperception of ―bad smells‖ in java source software. The paper aimed at using the metrics to identify the peculiarity of code smell. Interpretation rules had been applied to calculate the metrics results which are applied to Java source-code. Hence based on

the calculated results, location of bad smells in the java code could be easily identified. For the implementation a prototype tool had been used on two case studies.

H. Alikacem and H. Sahraoui [20] provided a language which identifies the overlook of quality factors and provides s methodology for identification of smells in object oriented systems. The terminology provided the guidelines with the help of fuzzy logic, metrics, association and inheritance. However it was not validated on any real world project. Other related detection approaches are discussed by R. Allen and D. Garlan [2, 4], E.M. Dashofy and A. vander Hoek *et al.*, [17].

3.2 Visualization Based Detection Techniques

In visualization based detection techniques the researchers used different approaches i.e. Metric Based Visualization Technique, Visualized design defect Detection Strategy, Domain Specific Language etc. Some of the approaches are discussed here.

Simon *et al.,* [10] suggested a powerful technique to inspect the internal quality of the software using a metric based visualization approach. Four types of source code refactoring had been analyzed: move function, move attribute, extract class and inline class. An enhanced metric tool- Crocodile had been used. The approach enabled the software engineers to identify the ―code smells‖ with a click of mouse and following the visualization rules.

Langelier *et al.,* [19] specified a visualization approach for the quality analysis of large scale systems. A framework had been provided which was implemented on open source systems. Geometrical 3D Box was used for the representation of classes. Analysis had been done on the values of Metrics i.e. For Coupling, Cohesion, Inheritance and Size Complexity CBO, LCOM5, DIT.WMC metrics had been used.

Dhambri, K. *et al.,* [25] introduced a design defect detection visualization based strategy. The approach is validated for three types of anomalies- Blob, Functional Decomposition and Divergent Change. The study is further extending to automatic detection based approach in the near future.

Cedric Bouhours *et al.,* [29] worked on the investigation of bad smells in the designing process. The spoiled patterns had been targeted for the identification of bad smells. Spoiled Patterns are defined as the patterns which did not provide proper functionality to the system for which it had been designed. A comparison had been made between design patterns and spoiled patterns

Naouel Moha *et al.,* [31] specified a domain specific language based on DECOR, for unmasking anti-patterns. It is a mechanism which provides a track for description of anti-patterns, by going through the sequence of steps: Description analysis, Specification, Processing, Detection, and Validation. It casts a detection system, DETEX which plays role of reference instantiation of DECOR. More than 15 types of code smells had been identified on 11 open source systems.

3.3 Automatic Detection Techniques

In the automatic detection strategies fully automatic detection tools had been used. Different types of anti- patterns are identified and few of these approaches are validated on the real world systems. Some of the approaches are discussed here:

Yann-Gael Gueheneuc *et al.,* [9] classified three types of design defects i.e. Intra-Class (within class), Inter- Class (among classes) and Semantic Nature. A Meta model had been used to describe design patterns. Inter- class design defects could be resolved easily with the help of Ptidej Tool. Eva van Emden and Leon Moonen [11] specified an approach by which the java source code software's quality can be improved. The concluded results can also be used in the tool for automatic software inspection. jCOSMO code smell browser have been developed to disclose the smells in java source code. The tool was validated for CHARTOON system. Jagdish Bansiya and Carl G. Davis [12] introduced a hierarchical prototype for the evaluation of quality attributes (reusability, flexibility, understandability etc.) in object oriented designs. Architectural and Detectable equities of classes and their objects is calculated using design metricsDAM, DCC, CAM, etc. The model provided approach to implement it on real world projects easily.

Yann-Gael Gueheneuc [15] introduced a tool suite ―Ptidej‖ which have the capability to reverse-engineer different programming languages to UML class diagrams accurately. PTIDEJ generates the UML class diagrams which help in identification of code smells with a higher level of abstraction. The author provided a brief outline of different reverse engineering tools like Rational Rose, ArgoUML version 0.14.1, Chava Fujaba version 4.0.1 IDEA, Borland Together, and Womble recover.

S. Counsell and Y. Hassoun [21] described the refactoring of seven open source Java systems- MegaMek, JasperReports, Antlr, Tyrant, PDFBox, Velocity and HSQLDB. The results demonstrated that the most common re- engineering components of open source systems are- Renaming and Moving fields/methods among the code.

Yann Gael Gueheneuc and Giuliano Antoniol [26] presented Design Motif Identification Multilayered

Approach (DeMIMA) for the detection of micro architectures (complementary to design motifs). It was a three layered architecture in which the first two layers provided a miniature of the source code, and the third layer identified design patterns. The approach provided 100 % recall on the open source and industrial systems as well, using explanation-based constraint programming.

Stephane Vaucher *et al.,* [28] examined carefully the God Classes to detect the occurrence of bad smells in the software. Xerces and Eclipse JDT (open-source systems) - had been studied for the investigation of God Classes.

Salima Hassaine *et al.,* [32] introduced IDS (Immune based Detection Strategy) - a machine learning process which was energized from the immune system of the human body. System could be easily identified for the presence of code smells and anti-patterns. Gantt Project v1.10.2 and Xerces v2.7.0 were manually-checked for the existence of smells. Foutse Khomh *et al.,* [34] proposed Bayesian Detection Expert, a Goal Question Metric (GQM) based approach to construct Bayesian Belief Networks (BBNs) from the descriptions of anti-patterns. BBN examined that identify whether a class is an anti-pattern or not. BDTEX is validated for three anti-patterns: Blob, Functional Decomposition, and Spaghetti code including two open source systems Gantt Project and Xerces. The approach is also applied to two industrial projects Eclipse & JHotDraw.

Satwinder Singh and K. S Kahlon [35] investigated the importance of software metrics and encapsulation for revealing the code smells. A software metric model had been introduced that provided the categorization of smells in the code. Firefox open source system had been investigated for the validation of results.

Satwinder Singh and K.S Kahlon [36] introduced a metric model for investigating the smelly classes in the system. The paper revealed that the results obtained from the metrics could be helpful in determining the code smells and faulty classes. Francesca Arcelli Fontana *et al.,* [37] proposed that various software analysis code smell detection tools are available in the market but the accuracy of their judgment is still not very much clear. Therefore six versions of Gantt Project had been explored for the detection of four types of code smell, using more than six tools.

Daniele Romano, Paulius Raila *et al.,* [38] studied the system by considering source code changes (SCC) obtaining from 16 Java open source systems. Three anti-patterns Complex Class, Spaghetti Code, and Swiss Army Knife have been identified. It had been detected that the number of code changes in anti-patterns classes is greater than the number of changes with no anti-pattern.

Foutse Khomh *et al.,* [40] investigated the affect of antipatterns on classes. More than 50 releases of four systems Argo UML, Eclipse, Mylyn, and Rhino had been considered. 13 types of anti-patterns have been identified .The relation between the habitation of anti-patterns with the change tendency and fault-tendency is investigated. It had been detected that classes participating in anti-patterns are faultier than others.

Hui Liu *et al.,* [41] aimed at detecting bad smells in the code. A detection strategy had been introduced that reduces the efforts of detecting bad smells by a factor of 17 to 20 %.

Abdou Maiga, Nasir Ali *et al.,* [42] described ―SMURF‖ which is an Anti-pattern Detection Approach. More than 290 experiments have been conducted on three systems i.e Argo UML, Xerces, Azureus. Four types of anti-patterns Blob, Spaghetti Code, Functional Decomposition, and Swiss Army Knife have been identified. Author revealed that the accuracy rate of SMURF is greater than that of DETEX and BDTEX for detection of anti-patterns in the system.

Kwankamol Nongpong [43] carried out the research by combining the code smells with the tools needed for refactoring. A tool had been generated called JcodeCanine which could easily identify the code smells and provided with the information where the refactoring was needed.

Fehmi Jaafar, Yann Gael Gueheneuc *et al.,* [44] provided a relationship between anti-patterns and design patterns. Three open source systems Argo UML, JFreeChart and XercesJ had been considered for the evaluation of relationship. It had been concluded that relationship exists between anti-patterns and design patterns but on temporary basis. The classes had more error tendency which is present in such anti-patterns.

Harshpreet Kaur Saberwal *et al.,* [45] explored the open source systems for the identification of code smells in the classes. An empirical model had been designed for the detection of smells in the system. The work carried out is validated on the versions of real world project – JfreeChart.

Pandiyavathi and Manochandar [47] suggested the methods for the revealing the code smells in the system. An overview of refactoring technique had been proposed which would be time saving. Algorithm had been proposed to implement the refactoring methods.

Francis Palma *et al.,* [48] specified the detection of anti-patterns in business processes. The rule-based approach has been detected for improving

the quality of BPEL (Business Process Execution Language) processes to detect BP anti-patterns. Seven BP anti-patterns have been specified and four have been detected with three example BPEL processes. Francis Palma *et al.* [49] proposed that the quality of service based systems get affected by the use of anti- patterns. Based on the data collected from the SBS FraSCAti, it was shown that the services suspicious of anti-patterns require more maintenance that non-patterns services.

Satwinder Singh and K. S. Kahlon [50] revealed the importance of metrics and the threshold values in software quality assurance. Analysis of risk in software system was explored against the threshold values for the detection of bad smells. Hence based on threshold values faulty classes could be easily identified. The study is validated by the three versions of open source systems of Mozilla Firefox.

Jiang Dexun, Ma Peijun *et al.,* [51] suggested that the classes which were functionally not related could generate problems in software maintenance. Hence the detection and refactoring of such classes is needed. A bad smell was proposed by the author named - Functional over related classes (FRC). A detection strategy was suggested to indentify the bad smell. The work was validated on four open source systems- HSQLDB, Tyrant, ArgoUML and JfreeChart.

3.4 Empirical Detection Techniques

The following are the empirical detection techniques which explore the work done on code smells and anti- patters**.** Different types of anti-patterns have been considered by different authors. Some of the proposed work is given below: Mika Mantyla *et al.*, [13] presented the research work done on the bad code smells. The paper provided taxonomy for making the smells more understandable. Author revealed different types of classes for bad smells like Bloaters, Encapsulators, Dispensables, Couplers, etc. A Survey was performed at a Finnish software company, which provided a correlation between the smells.

Foutse Khomh *et al.,* [24] introduced the concept of software quality maintenance by avoiding the use of harmful antipatterns. The paper revealed that the quality of the software is affected by use of anti-patterns.

S. Olbrich *et al.,* [27] considered the historical data of Lucene and Xerces. It had been identified that classes with the antipatterns Blob and Shotgun Surgery have a higher change frequency than non-anti-patterns classes.

Min Zhang, Tracy Hall *et al.,* [33] provided a deep insight of literature by going through more than 300 papers on code bad smells since 2000. The paper disclosed that research work is needed on the percussion of code smells. It had been concluded that the smell-Duplicated Code is studied more than other code smells.

Rabia Bashir [39] identified the impact of anti-patterns on open source software development. The paper revealed that the anti-patterns, which are available in open source software development and the solutions to avoid them.

Harvinder Kaur and Puneet Jai Kaur [46] examined various types of anti-pattern detection techniques i.e. Manual (metric based approach, metric-based heuristics, ad hoc domain specific language) Semi-Automated (DCPP matrix) and SVM based anti-pattern detection techniques (DTEX, BTEX, SMURF).

4. CONCLUSION

In this paper a vast literature survey have been done to limelight the affect of anti-patterns on the source code. Our study reveals different approaches for the detection of anti-patters and code smells. It has been concluded that the research community have analyzed their results on the basis of within company projects only. The need has been identified to examine the results for different company projects. Further it has been analyzed that not much researchers have done work on large projects to identify the anti-patterns. Only few have done work on large number of anti-patterns to disclose the impact of these smells. Therefore the need has been generated for the identification anti-patterns and the kinds of anti-patterns with their impact on classes in the object oriented open source projects. Thus the future work will also be possible for the identification of the commonly occurring anti-patterns in the open source systems and to investigate the impact of anti-patterns using the software metric. Hence the results obtained will be beneficial to the software industry to improve the quality of the software system by predicting the faulty classes, during the testing phase. It helps in providing more reliability during testing and maintenance phases by predicting anti-patterns and faults before delivery of the product. Thus the results produced will also be of interests to engineers, as they can predict which classes are to be tested more precisely. The study will be valuable for the software engineers and managers for improving their maintenance activities by eliciting the code smells.

REFERENCES

1. Webster, B. F. (1995). Pitfalls of Object Oriented Development. 1st Ed. M & T Books.
2. Garlan, D., Allen, R., & Ockerbloom, J. (1995). Architectural Mismatch: Why Reuse is so Hard. *IEEE Software*, 12(6), 17-26.
3. Koenig & Andrew, (1995). Patterns and Antipatterns. *Journal of Object-Oriented Programming*, 8, 46– 48.
4. Allen, R., & Garlan, D. (1997). A formal basis for architectural connection. *ACM Transactions on*

Software Engineering and Methodology (TOSEM), *6*(3), 213-249.
5. Brown, W. J., Malveau, R. C., Brown, W. H., McCormick III, H. W., & Mowbray, T. J. (1998). Anti-patterns: refactoring software, architectures, and projects in crisis‖. 1st Ed. Wiley, New York.
6. Fowler, M. (1999). Refactoring—improving the design of existing code‖ 1st Ed. Addison-Wesley.
7. Travassos, G., Shull, F., Michael, F., & Victor, R. B. (1999). Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality.‖ In *Proceedings of 14th Conf. Object-Oriented Programming, Systems, Languages, and Applications*, pp. 47-56.
8. Connie, U., Smith, & Lloyd, G. W. (2000). Software Performance Anti-patterns.‖ *ACM Soft. Engg. Research*, pp. 127-136.
9. Yann Gael, G., Albin-Amiot, H., & Ecole des Mines de, N. (2001). Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-class Design Defects.‖ Paper accepted at TOOLS USA.
10. Simon, F., Steinbruckner, F., & Lewerentz, C. (2001). Metrics Based Refactoring,‖ In *Proceedings of Fifth European Conf. Software Maintenance and Re-eng.*, p.30
11. Eva van, E., & Leon, M. (2002). Java Quality Assurance by Detecting Code Smells,‖ In *Proceedings of Ninth Working Conference on Reverse Engg. IEEE.*
12. Jagdish, B., & Carl, G. D. (2002). A Hierarchical Model for Object Oriented Design Quality Assessment.‖ *IEEE Trans. on Software Eng.*, 28(1), 4-17.
13. Mika, M., Jari, V., & Casper, L. (2003). A Taxonomy and an Initial Empirical Study of Bad Smells in Code.‖ In *Proceedings of the Inter. Conference on Software Maintenance*, IEEE. pp. 381-384.
14. Marinescu, R. (2004). Detection Strategies: Metrics-Based Rules for Detecting Design Flaws.‖ In *Proceedings of 20th Int. Conf. Software Maintenance*, pp. 350-359.
15. Yann Gael, G. (2004). A Systematic Study of UML Class Diagram Constituents for their Abstract and Precise Recovery. 11th Asia-Pacific Conference on Soft. *Engg*, pp. 265-274.
16. Munro, M. J. (2005). Product Metrics for Automatic Identification of ―Bad Smell‖ Design Problems in Java Source-Code.‖ In *Proceedings of1 1th IEEE Int. Software Metrics Symp.*
17. Dashofy, E. M., vander Hoek, A., & Taylor, R. N. (2005). A Comprehensive Approach for the Development of Modular Software Architecture Description Languages.‖ *ACM Trans. Software Eng. and Methodology*, 14(2), 199-245.
18. Yann Gael, G. (2005). Ptidej: Promoting Patterns with Patterns‖ In *Proceedings of 1st ECOOP workshop on Building a System using Patterns (BSUP)*, pp. 1-9 SpringerVerlag.
19. Langelier, G., Sahraoui, H. A., & Pierre, P. (2005). Visualization-Based Analysis of Quality for Large-Scale Software Systems.‖ *ACM Inter. Conf. on Automated Soft. Engg*, pp. 214-223.
20. Alikacem, E. H., & Sahraoui, H. (2006). Generic Metric Extraction Framework.‖ In *Proceedings of 16th Int. Workshop Software Measurement and Metrik Kongress*, pp. 383-390.
21. Counsell, S., & Hassoun, Y. (2006). Common Refactorings, a Dependency Graph and some Code Smells: An Empirical Study of Java OSS.‖ *IEEE Inter. Symposium on Empirical Soft. Engg*. pp. 288-296.
22. Yann-Gael, G. (2007). Ptidej: A Flexible Reverse Engineering Tool Suite.‖ *IEEE Inter. Confer. On Soft. Maintenance*, pp 529-530.
23. Naouel, M., Yann-Gael, G., Anne- Francoise Le, M., & Laurence, D. (2008a). A domain analysis to specify design defects and generate detection algorithms.‖ In *Proceedings of of 11th Int. Conf. on Fundamental Approaches to Soft. Engg*., Springer New York, pp. 276-291.
24. Foutse, K., & Yann-Gael, G. (2008). Do Design Patterns Impact Software Quality Positively?‖ In *Proceedings of 12th Conf. on Soft. Maintenance and Reengineering IEEE* pp. 274-278.
25. Dhambri, K., Sahraoui, H., & Poulin, P. (2008). Visual Detection of Design Anomalies‖ In *Proceedings of 12th European Conf. Software Maintenance and Reng*, pp. 279283.
26. Yann, G., & Giuliano, A. (2008). DeMIMA: A Multilayered Approach for Design Pattern Identification.‖ *IEEE Trans. on Software Eng.*, 34(5), 667-684.
27. Olbrich, S., & Cruzes, D. S. (2009). ―The evolution and impact of code sells: A case study of two open source systems.‖ In *3rd Inter. Symposium on Empirical Soft. Engg. and Measurement*, pp. 390-400.
28. Stephane, V., Foutse, K., Naouel, M., & Yann-Gael, G. (2009). Tracking Design Smells: Lessons from a Study of God Classes‖ In *16th Working Conference on Reverse Engg.*
29. Cedric, B., & Herve, L. (2009). Bad smells in design and design patterns. *Journal of Object Techn.*, 8(3), 43-63.
30. Naouel, M., Yann-Gael, G., Anne-Francoise Le, M., Laurence, D., & Alban, T. (2010). From a Domain Analysis to the Specification and Detection of Code and Design Smells‖ *Springer Verlag (Germany)*, pp.345-361.
31. Naouel, M., Yann-Gael, G., Laurence, D., & Anne-Francoise Le, M. (2010). DECOR: A Method for the Specification and Detection of Code and Design Smells‖ *IEEE Trans. on Software Eng.*, 36(1), 20-36.
32. Salima, H., Foutse, K., Yann-Gael, G., & Sylvie, H. (2010). IDS: An Immune-Inspired Approach for the Detection of Software Design Smells‖ *7th IEEE*

Inter. Conference on the Quality of Infor. And Comm. Tech., pp. 343-348.
33. Min, Z., Tracy, H., & Nathan, B. (2011). Code Bad Smells: a review of current knowledge‖ *Journal Software Maintenance Evol. Res. Pract.*, 23, 179–202.
34. Foutse, K., Stephane, V., Yann-Gael, G., & Houari, S. (2011). Bdtex: A gqm-based bayesian approach for the detection of anti-patterns‖ *J. Syst. Softw.*, 84(4), 559–572.
35. Satwinder, S., & Kahlon, K. S. (2011). Effectiveness of Refactoring Metrics Model to Identify Smells and Error Prone Classes in Open Source Software‖ *ACM SIGSOFT Soft. Engg. Notes*, 36(5), 1-11.
36. Satwinder, S., & Kahlon, K. S. (2012). Effectiveness of Encapsulation and Object Oriented Metrics to Refactor Coe and Identity Error Prone Classes using Bad Smells. *ACM SIGSOFT Soft. Engg. Notes*, 37(2), 1-10.
37. Francesca Arcelli, F., Pietro, B., & Marco, Z. (2012). Automatic detection of bad smells in code: An experimental assessment‖ *Journal of Object Technology*, 11(2), 1–38.
38. Daniele, R., & Paulius, R. (2012). Analyzing the Impact of Anti-patterns on Change-Tendency Using Fine-Grained Source Code Changes‖ *Proc of the 19th Working Conference on Reverse Engineering (WCRE), IEEE Computer Society Press.*
39. Rabia, B. (2012). Anti-patterns in Open Source Software Development‖ *Int. Journal of Computer Applications*, 44(3).
40. Foutse, K., Massimiliano Di, P., Yann Gael, G., & Giuliano, A. (2012). An exploratory study of the impact of anti-patterns on class change- and faulttendency‖ *Springer Science Business Media*, LLC, Aug.
41. Hui, L., Zhiyi, M., Weizhong, S., & Zhendong, N. (2012). Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort‖ *IEEE Trans. On Software Engg.*, 38(1).
42. Abdou, M., Nasir, A., Neelesh, B., Aminata, S., Yann-Gael, G., & Esma, A. (2012). SMURF: A SVM-based Incremental Anti-pattern Detection Approach. *Presented at 19th Working Conference on Reverse Engineering*, pp. 466-475.
43. Kwankamol, N. (2012). Integrating \Code Smells Detection with Refactoring Tool Support‖ Ph.D. Dissertation, University of Wisconsin-Milwaukee.
44. Fehmi, J., Yann, G., Sylvie, H., & Foutse, K. (2013). Analysing Anti-patterns Static Relationships with Design Patterns‖ In *Proceedings ofthe First Workshop on Patterns Promotion and Anti-patterns Prevention*, EASST, 59.
45. Harshpreet, K. S., Satwinder, S., & Sarabjit, K. (2013). Empirical Analysis of Open Source System for Predicting Smelly Classes‖ *Inter. Journal of Engineering Research & Technology*, 2(3), 1-6.
46. Harvinder, K., & Puneet Jai, K. (2014). A Study on Detection of Anti-Patterns in Object-Oriented Systems‖ *Inter. Journal of Computer Applications*, 93(5), 25-28.
47. Pandiyavathi & Manochandar. (2014). Detection of Optimal Refactoring Plans for Resolution of Code Smells‖ Inter. *Journal of Advanced Research in Computer and Comm. Engg.*, 3(5), 6-11.
48. Francis, P., Naouel, M., & Yann Gael, G. (2013). Detection of Process Anti-patterns: A BPEL Perspective‖ 17th *IEEE Int. Workshop on Enterprise Distributed Object Computing*, pp. 173-177.
49. Francis, P., & Le, A. (2014). Investigating the Changeproneness of Service Patterns and Anti-patterns‖ *7th Inter. Conf. on Service-Oriented Computing and Applications IEEE* pp. 1-8.
50. Satwinder, S., & Kahlon, K. S. (2014). Object oriented software metrics threshold values at quantitative acceptable risk level‖ *CSI Transactions on ICT*, Springer 2(3), 191-205.
51. Jiang, D., Ma, P., Su, X., & Wang, T. (2014). Functional over-related classes bad smell detection and refactoring suggestions‖ *International Journal of Software Engineering & Applications (IJSEA)*, 5(2), 29-47.