International Journal of Advanced Science and Technology
Vol. 29, No. 12s, (2020), pp. 1169-1183

Antipatterns: A Search Of Negative For Positive The Analysis Of Published Taxonomy

Kapil Kumar[1], Anil Kumar Solanki[2], Sharvan Kumar Garg[*]
[1]*Research Scholar, Department of Computer Science & Engineering, Bhagwant University, Ajmer, Rajasthan, India*
[2]*Professor & Head, Bundelkhand Institute of Engineering & Technology, Jhansi, Uttar Pradesh, India*
[*]*Professor and Head, Subharti Institute of Technology & Engineering, Subharti University, Meerut, Uttar Pradesh, India*
[*]*shravan_garg23@rediffmail.com*

Abstract

Performance is an important quality attribute among all other attributes. It is important to analyze the performance of any software system and on the other hand is difficult to understand. If the performance of any software is not up to the mark, a number of negativities for example the breakage of strong relationship with the customer, downing of business, wastage of money and time, reimplementation of software or application etc. may occur. If the problem of poor performance occurs it may create delays in the implementation, restructuring and even the system may be implemented again which automatically increase the cost of the overall project. Antipatterns are solutions to the recurring design problems occurs due to poor practices and poor smells. The identification of antipattern has become the topic of interest of the scientist and the software or application developers. A number of approaches of detection of antipatterns have also been proposed by the researchers. Software Developers and Modeling Experts should be aware of all these types of antipatterns. Although a number of research papers have already been written to explore the importance of antipatterns but a structured bibliography of antipatterns which shows its necessity is missing. To overcome this difference, this paper presents a bibliography of presented and proposed antipatters observed in established systems. Along with the analysis of existing antipattern, some issues for future discussion will also be proposed. This paper provides a support to research scholars and the software developers those who are interested in analysis of already proposed taxonomy related to the finding of antipatterns during the development of any software.
Keywords: *Antipattern, Poor Smells, Poor Practices, Process Models, Taxonomy*

1. Introduction

The performance of any system depends on the overall process, implemented in System Development Life Cycle. There are a lot of evidences that demonstrates design problems i.e. antipatterns, which degrade the quality of software deployment and its maintenance. From last many years, finding of antipatterns has become the topic of study with the aim of analyzing its effect on software deployment and its maintenance. A number of studies have shown that the source code was effected with poor practice that is containing the antipatterns is required to convert in the clean code [31] and for this an exercise is required. The source code with antipatterns has a great chance of faultiness in comparison to rest code in the system [31][37]. The presence of antipatterns, results in the overall performance degradation [1][21].

These issues gives proper directions that poor practices need to be identifies very carefully and required refactoring operations should be planned accordingly to overcome the problem of antipatterns. It can be suggested in the following manner.

(i) first of all identification of the antipatterns and then
(ii) Providing the refactoring solutions to deal with the same.

The use of patterns in the field of software engineering goes back to the seminal work of [54]. In this work, a pattern is a structured description of a reusable solution for software design. It is an important property of a pattern that its description follows a certain structure, which includes the pattern name, a description of the problem and a solution and consequences.

Antipatterns, coined in 1995 by [7], are solutions that are known to have deficiencies. In the domain of business processes, such patterns are also known as weakness patterns [8]. They document commonly reinvented bad solutions to recurring problems [9]. Antipatterns are often recognized by the appearance of failures, which are identified during execution or system implementation [10]. Although it is known that the use of antipatterns has disadvantages, they can frequently be found in practice as long as modelers are not trained to avoid those [11]. Time pressure, failed design decisions and unforeseen changes might be reasons for the (unintentional) use of antipatterns [12]. In the traditional sense of the word, an antipattern "should not only be a bad solution, but also explains why this solution appears problem-solving and why it actually turns out to be a poor practice" [13]. For the purpose of this work, we use the term "antipattern" less strictly, which is in line with the use of this term by [14]. In order to get a broad overview on the topic, we also include patterns that should rather be called "bug patterns". Those are patterns describing frequent errors and violations in process models.

.

2. Representation of Antipatterns

To understand how antipatterns are represented, we further analyzed a number of publications with respect to the dimensions antipattern Number, naming of antipatterns, Definition, Description and contribution.

Antipattern #: This dimension refers to the proposed number of antipattern in the research paper such as AN1, AN2, AN3, …….. etc.

Naming: This dimension refers to the name of the antipatterns.

Definition: This dimension refers that due to what poor practice it comes to the existence i.e. cause.

Description: This dimension refers to the possible solution in the antipattern text. However, in the case of antipatterns, the problem and its consequence are heavily intertwined. In some cases, the negative consequence of an antipattern even seems to be more descriptive and recognizable than a detailed description of the problem. So we have opted to combine "problem" and "consequence" to single description.

Contribution: This dimension refers to that who has contributed in the analyzing and observing the antipattern.

3. Taxonomy of proposed Antipattern

Antipattern #: AN1

Naming: Blob.

Definition: It occurs when a single class either executes the complete task of the system or contains the total data of the system.

Description: Uniform distribution of assignments and data through refactoring of the Architectural Design.

Contribution: Smith and Williams.

Antipattern #: AN2

Naming: Concurrent Processing System (CPS).

Definition: It occurs when maximum utilization of processing elements does not take place.

Description: Use of appropriate Scheduling Algorithms for implementation of concurrent processing in the system.

Contribution: Smith and Williams.

Antipattern #:	AN3
Naming:	Pipe and Filter Architectures (PFA).
Definition:	Pipe and Filter Architecture problem is observed due to slowest filter which causes system to have unacceptable throughput.
Description:	Splitting of big filters into small modules to overcome the overhead.
Contribution:	Smith and Williams.
Antipattern #:	AN4
Naming:	Extensive Processing (EXP).
Definition:	It occurs when the response time is increased due to extensive processing.
Description:	Reduction of extensive processing to remove high traffic.
Contribution:	Smith and Williams.
Antipattern #:	AN5
Naming:	Circuitous Treasure Hunt (CTH).
Definition:	It occurs when maximum number of locations is required to search by an object for getting appropriate or required data or information.
Description:	Change in design to provide separate paths for processing.
Contribution:	Smith and Williams.
Antipattern #:	AN6
Naming:	Empty Semi Trucks (EST).
Definition:	It occurs when a number of executions of request take place to execute a single task.
Description:	Solutions to provide maximum utilization of available bandwidth.
Contribution:	Smith and Williams.
Antipattern #:	AN7
Naming:	Tower of Babel (TOB).
Definition:	It occurs when the large scale conversion of internal data into general format takes place.
Description:	Conversion of data into required format or minimize the conversion rate.
Contribution:	Smith and Williams.
Antipattern #:	AN8
Naming:	One lane Bridge.
Definition:	It occurs when any process do not get chance of its execution.
Description:	Use of scheduling algorithms to avoid or minimize conflicts.
Contribution:	Smith and Williams.
Antipattern #:	AN9
Naming:	Excessive Dynamic Allocation (EDA).
Definition:	It occurs due to creation and deletion of objects without any requirement.
Description:	Reuse of available objects.

Contribution:	Smith and Williams.
Antipattern #:	AN10
Naming:	Traffic Jam.
Definition:	It occurs due to backlog of processes that create inconsistency in response time.
Description:	Removal of cause of backlog of jobs or implementation of load balancing.
Contribution:	Smith and Williams.
Antipattern #:	AN11
Naming:	The Ramp.
Definition:	It occurs when the processing time increases as per the usage of system.
Description:	Use of Dynamic Data Structures in place of static.
Contribution:	Smith and Williams.
Antipattern #:	AN12
Naming:	More is less.
Definition:	It occurs due to more Thrashing.
Description:	Measurement of required time Thrashing.
Contribution:	Smith and Williams.
Antipattern #:	AN13
Naming:	Falling Dominoes.
Definition:	It occurs when the failure causes performance failures in other components.
Description:	Degrade software performance, poor reliability and occurs fault tolerance issues. The module due to which the performance is degraded must be separated until not improved or corrected.
Contribution:	Smith and Williams.
Antipattern #:	AN14
Naming:	Unnecessary Processing.
Definition:	It occurs due to use of irrelevant code.
Description:	Removal of extra or irrelevant code or steps.
Contribution:	Smith and Williams.
Antipattern #:	AN15
Naming:	Code Bad Smell.
Definition:	During Software Development Life Cycle, software passes through different development stages. It is possible that changes at different stages made by the software developers in a hurry because of other constraints regarding time etc. As a result, the source code during changes at different stages the quality may be compromised that is introduction of antipattern in the Code with Bad Smell [24].
Description:	As the overhead of responsibilities increases, the complexity of a class increases and automatically the its introduces poor quality. These classes with responsibility overhead are known as Large Class [24]. The designing of separate classes for a group of tasks ir required.

Contribution:	[16][17].
Antipattern #:	AN16
Naming:	Feature Envy.
Definition:	When antipattern effects to the other class. It may be because of dependencies with the other class [24]. The concept of cohesion and coupling may be viewed.
Description:	A proper identification of dependencies among the classes through the technique. The problem of so many references must be removed.
Contribution:	[28], [36].
Antipattern #:	AN17
Naming:	Duplicate Code.
Definition:	**If the same code is written at many places, it is duplicate code antipattern**. The identification of duplicate code antipattern is not an easy task as it may be due to common activities or functionalities. It may increase the overhead of maintainability.
Description:	**A number of techniques have been proposed to find out the duplicate code antipattern.**
Contribution:	[30], [28] proposed the tool and techniques for the identification of duplicate code antipattern.
Antipattern #:	AN18
Naming:	Refused Bequest.
Definition:	It may be due to wrong use of the concept of inheritance in Object Oriented development. If during the development in the object Oriented Environment, if something wrong is observed with any attribute which is inherited by any children class, the change in such attribute needs to be done only in the parent of that class.
Description:	A tracking method of proper use of inheritance is required to implement to lookafter the following characteristics: (i) Proper hierarchical system (ii) correct implementation of overriding.
Contribution:	[25], [39].
Antipattern #:	AN19
Naming:	Divergent Change.
Definition:	The problem of low cohesion in classes.
Description:	The method of getting the internal information of the architecture is required.
Contribution:	[48]
Antipattern #:	AN20
Naming:	Shotgun Surgery.
Definition:	Extra efforts for changes[24].
Description:	A technique of finding of this antipattern is proposed by Rao and Raddy [52] which is based on DCPP matrix implementation. One more technique was proposed in HIST [48].
Contribution:	[52].

Antipattern #:	AN21
Naming:	Parallel Inheritance.
Definition:	It is also based on Shotgun Surgery antipattern where all the time a subclass of one class is created.
Description:	The use of technique to observe the internal structure based on historical information.
Contribution:	[33].
Antipattern #:	AN22
Naming:	Functional Decomposition.
Definition:	The poor use of polymorphism may introduce this antipattern.
Description:	The proper knowledge of functional description is required. To identify this antipattern, a procedure is called with to invoke its function.
Contribution:	[44]
Antipattern #:	AN23
Naming:	Spaghetti Code.
Definition:	The use of procedural programming in complex methods with no arguments.
Description:	**I**mplementation of object oriented environment.
Contribution:	[44]
Antipattern #:	AN24
Naming:	Swiss Army Knife.
Definition:	**T**o represent the various responsibilities, exposure of high complexity. **T**his antipattern occurs when a class has a number of complex methods.
Description:	The information of structure is required for the identification of this antipattern.
Contribution:	[44]
Antipattern #:	AN25
Naming:	Type Checking.
Definition:	The use of **c**omplex conditional statements increase the difficulty in the code and automatically tough maintainence. It may occur bad use of polymorphism.
Description:	The complex conditional statements must be removed.
Contribution:	[58]
Antipattern #:	AN26
Naming:	Does more than it says.
Definition:	This antipattern occurs when there is a misunderstanding between the nomenclature of the method and what the method actually does. The method performs more than that is interpreted with its nomenclature.
Contribution:	[4].
Antipattern #:	AN27
Naming:	Says more than it does.

Definition:	This antipattern occurs when the method does not behave as with nomenclature.
Description:	The catalogue with detail behavior is required.
Contribution:	[33], [40], [71].
Antipattern #:	AN28
Naming:	Does the Opposite.
Definition:	This antipattern is identified when the execution is opposite to the nomenclature [4] and the identification of this antipattern will become a complex task if proper documentation is not available.
Description:	The heuristics can be used for the identification of this antipattern.
Contribution:	[33], [40], [71].
Antipattern #:	AN29
Naming:	Contains more than it says.
Definition:	This antipattern occurs when a variable of a class is related to various objects.
Description:	The Correct declaration of variables in a class.
Contribution:	[33], [40], [71]
Antipattern #:	AN30
Naming:	Says more than it contains.
Definition:	This antipattern is opposite to contains more than it says [4] antipattern.
Description:	The correct use in the declaration of variables in a class.
Contribution:	[33], [40], [71].
Antipattern #:	AN31
Naming:	Contains the Opposite.
Definition:	This antipattern also occurs due to incorrect use of a variable in a class [4].
Description:	**T**his antipattern can be identified by the checking of documentation.
Contribution:	[33], [40], [71].
Antipattern #:	AN32
Naming:	Syntax Errors.
Definition:	This describes antipatterns where the syntax of the process modeling language has been used wrongly and therefore the models are invalid [57].
Description:	Exact use of semantics.
Contribution:	[57], [58]
Antipattern #:	AN33
Naming:	Control-flow Problems.
Definition:	This describes antipatterns with flaws related to the control-flow of the process model. In technical terms, this kind of problems can be subsumed as a violation of the soundness property [43].
Description:	Proper Synchronization, Avoidance of livelock problem, loop structure, Termination Boundaries**.**
Contribution:	[43], [55], [73].

Antipattern #:	AN34
Naming:	Understandability Problems.
Definition:	This characteristic describes antipatterns related to problems that make the process models difficult to understand.
Description:	Modularity and other related solutions can be provided.
Contribution:	[43], [55], [73].
Antipattern #:	AN35
Naming:	Composition Defects.
Definition:	This characteristic describes antipatterns relating to the collaboration between actors, e.g., the cooperation of human actors in groups, departments and institutions.
Description:	Class wise composition.
Contribution:	[43], [55], [73].
Antipattern #:	AN36
Naming:	Data-flow-related defect
Definition:	This characteristic describes antipatterns related to data that can be created, edited, deleted, or stored wrongly. It includes violations of security and privacy requirements.
Description:	Component to component module wise data flow**.**
Contribution:	[43], [55], [73].
Antipattern #:	AN37
Naming:	Rule-related defect.
Definition:	This characteristic contains antipatterns related to business rules that are missing, redundant or conflicting or the input for the decision is missing [28].
Description:	Data consistency.
Contribution:	[28]
Antipattern #:	AN38
Naming:	Process-related defect.
Definition:	This describes antipatterns that describe negative properties of the actual process (other than problems in the process model).
Description:	1. Need for process improvements: This relates to weaknesses in the process that can lead to higher costs, longer processing time, lower quality or more errors. Examples for such problems are media disruption, inefficient or double work, and problems related to the organizational structure. 2. Compliance: This subsumes antipatterns that describe the violation of rules established by law, organizational rules, or rules defined by standards.
Contribution:	[28]
Antipattern #:	AN39
Naming:	Deadlocks**.**

Definition:	These antipatterns describe the attempt to synchronize two flows of the control of which at least one has not been activated before [55]. Due to improper allocation of resources, this antipattern may arise, the performace may be degraded. In number of research papers like [10], [17], [36]. [42], [43], [52], [59], [64], [65], [70], [71], [72], [76] it has been shown as antipattern.
Description:	Proper allocation and utilization of recourses.
Contribution:	[55] and Mentioned in all the research papers listed above.
Antipattern #:	AN40
Naming:	Dead Activity.
Definition:	These antipatterns represent activities that can never be reached by the flow of control [59]. When any process or resource never get chance to execute. Due to which it gets out of streamline.
Description:	Proper allocation and utilization of resources.
Contribution:	[59]
Antipattern #:	AN41
Naming:	Infinite Loop Error.
Definition:	This describes antipatterns which represent infinite loops, i.e. set of activities is executed infinitely often as no abort condition exists or the condition never becomes true. It occurs due to missing of boundary value in any iteration.
Description:	Fixed and consistent boundaries
Contribution:	[19], [38], [44]
Antipattern #:	AN42
Naming:	Lack of Synchronization Error.
Definition:	These antipatterns describe the unintentional multiple activation of activities [38]. Improper signaling and communication. (16 papers)
Description:	Strong signaling to achieve synchronization.
Contribution:	[19], [38], [44]
Antipattern #:	AN43
Naming:	Incorrect Termination Error.
Definition:	This describes an antipattern forcing the process to terminate prematurely (Koehler and Vanhatalo, 2007). It occurs due to unexpected termination.
Description:	Set of boundaries as per the requirement.
Contribution:	[56], [57], [58]
Antipattern #:	AN44
Naming:	Language deficit**.**
Definition:	This addresses incomplete, improper or ambiguous textual labels or labels that do not follow naming conventions.
Description:	Consistent Labeling.
Contribution:	[35], [39], [43]
Antipattern #:	AN45
Naming:	Layout deficit**.**

Definition:	This relates to antipatterns addressing the spatial layout of a model. This considers things such as the reading direction or the placing of model elements (spacing, overlapping, etc.).
Description:	Standard layout designing.
Contribution:	[63], [67]
Antipattern #:	AN46
Naming:	Complexity.
Definition:	This addresses overly complex modeling by means of too large diagrams, too many elements of a certain kind or a missed chance to reuse frequently occurring sub-fragments of a model.
Description:	Modularity, class wise set of related information.
Contribution:	[6], [11], [29]
Antipattern #:	AN47
Naming:	Data-flow-related defects**.**
Definition:	It occurs when the flow of data is mismatched with the requirement.
Description:	Implementation of flow graph.
Contribution:	[6], [11], [29]
Antipattern #:	AN48
Naming:	Ecological Impacts.
Definition:	This describes antipatterns that have a negative impact on the ecology. Examples are the negative impact on nature or the climate due to increased energy consumption or a waste of resources.
Description:	Maintenance of energy consumption and proper utilization of resources.
Contribution:	[6], [11], [29]
Antipattern #:	AN49
Naming:	Communication Defects.
Definition:	This describes antipatterns that address the communication within the process model. For instance, the quality and structure of the transmitted information can be adversely affected if the communication channels are not standardized.
Description:	Standardization of communication channel.
Contribution:	[15], [26], [32]

4. Future Scope

Various features that can be considered in future are listed in th following points-

4.1.0 Separate System wise or language wise finding of Antipatterns:

In the research paper "An Antipattern Documentation about Misconceptions related to an Introductory Programming Course in C" by [38] , October – 2017, researchers have identified a list of 33 misconceptions related to 7 programming topics (i) Function Parameters use and Scope, (ii) Variables, Identifiers, and scope (iii) Recursion (iv) Iteration (v) Structures (vi) Pointers (vii) Boolean Expression in C language. Likewise the antipatterns in other languages, systems or applications may also be tabulated.

4.2.0 Antipattern Detection Algorithms:

The different algorithms may be defined to extract the antipatterns from the Code.

4.3.0 Accuracy Evaluation of Antippatern detection Tool:

Different tools have been proposed for the identification of antipatterns depends on different environment. The process of evaluation of an antipattern detection tool can consist of different steps and can be done using different strategies

5. Limitations

As a result of studying the numerous bibliographies on antipattern, we recommend to specify and document antipatterns with the following characteristics. We believe that these characteristics improve the dissemination of an antipattern publication:

a. A graphical representation in addition to a textual one should be provided to support a consistent understanding.
b. A problem should be formulated in order to understand the "suboptimal" (bad) solution of the antipattern. The problem should be enriched with a description of the consequences that are associated with the use of the antipattern and thus prevent from its use.
c. The antipattern should not only present the questionable solutions but should also show a better solution that avoids the problems.
d. A limitation of our research is that (a) like most literature-based analyses we cannot guarantee its completeness. However, since we have varied the search query and have used multiple databases, we are quite optimistic to not have overlooked major antipatterns works. In some cases when the authors of identified papers did not explicitly use the term antipattern, we had to decide if the work under consideration qualifies as an antipattern work or not thus introducing a level of subjectivity.

6. Conclusion

Performance measurement of any software or application after its deployment for the use has become an essential quality attribute among all other software attributes and it is not very much useful to analyze the performance of any software when every aspect have already been implemented i.e. performance analysis after application deployment is not very much useful due to delay in processes of its corrections. Actually the analysis is required at initial phases of software development life cycle because analysis after deployment of the application at its working site shows only the dark and negative picture of the overall work. So performance analysis at earlier phases of development has become the necessity of the researchers.

Antipattern is the technique which looks at the negative side of the process used to implementation of any system In advance. It's a method of searching of negative to provide the positivity in the overall implementation of the system.

In this paper we have analyzed the taxonomy of already proposed antipatters previously and summarized them all together with naming, description and contribution. It may help to the researchers and software developers at the time of development of any application or software.

References

1. A. Chatzigeorgiou and A. Manakos, "Investigating the evolution of bad smells in object oriented code." in Proceedings of the 7th International Conference on the Quality ofInformation and Communications Technology, Porto, Portugal, 2010.
2. A. Jbara, A. Matan, and D. G. Feitelson, "High-MCC functions in the linux kernel," in Proceedings
of the IEEE 20th International Conference on Program Comprehension, Passau, Germany, 2012, pp. 83–92.

3. A. Marcus, D. Poshyvanyk, R. Ferenc, “Using the conceptual cohesion of classes for fault prediction in object-oriented systems”. IEEE Transaction on Software Engineering, 34(2), 287–300, 2008.
4. A. Rao and D. Ram, “Software Design Versioning using Propagation Probability Matrix”, in Proceedings of Third International Conference on Computer Applications, Yangon, Myanmar, 2005.
5. A. Rao and K. Reddy, “Detecting Bad Smells in Object Oriented Design Using Design Change Propagation Probability Matrix”, in International MultiConference of Engineers and Computer Scientists, 2008.
6. A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, “Predicting source code changes by mining change history,” IEEE Transactions on Software Engineering, 30(9): 574–586, 2004.
7. A. Takang, P. A. Grubb, and R. D. Macredie, “The effects of comments and identifier names on program comprehensibility: an experiential study,” Journal of Program Languages, 4(3):143–167, 1996.
8. A. Yamashita and L. Moonen, “Exploring the impact of inter-smell relations on software maintainability: an empirical study,” in Proceedings of the 35th International Conference on Software Engineering, San Francisco, CA, USA, 2013, pp. 682–691.
9. Agnes Koschmider, Ralf Laue, Michael Fellmann, “Business Process Model Antipatterns: A Bibliography and Taxonomy of Published Work”, Association for Information Systems, ECIS 2019.
10. Awad, A. and F. Puhlmann (2008). “Structural Detection of Deadlocks in Business Process Models.” International Conference on Business Information Systems. LNBIP, vol, 7, Springer, pp. 239–250.
11. Awad, A., G. Decker and N. Lohmann (2010): “Diagnosing and Repairing Data Anomalies in Process Models”, BPM Workshops 2009, LNBIP 43, pp. 5-16, Springer.
12. B. Caprile and P. Tonella, “Restructuring program identifier names,” in Proceedings of the International Conference on Software Maintenance, 2000, pp. 97–107.
13. Becker, J., B. Weiß and A. Winkelmann (2011b) “Automatic Identification of Structural Process Weaknesses – Experiences with Semantic Business Process Modeling in the Financial Sector” Wirtschaftsinformatik Proceedings 2011.
14. Becker, J., P. Bergener, D. Breuker and M. Raeckers (2012). “An empirical assessment of the usefulness of weakness patterns in the business process redesign”. Europ. Conf. on IS. 203.
15. Becker, J., P. Bergener, M. Räckers, B, Weiß, B. and A. Winkelmann (2010). “A Pattern-Based Semi-Automatic Analysis of Weaknesses in Semantic Business Process Models in the Banking Sector.” European Conference on Information Systems
16. Bergener, F., P. Delfmann, B. Weiss and A. Winkelmann (2015). ”Detecting potential weaknesses in business processes: An exploration of semantic pattern matching in process models“, Business Process Management Journal, 21(1): 25-54.
17. Borgert, S. and M. Mühlhäuser (2014). “Formal Based Correctness Check for ePASS- IoS 1.1 Process Models with Integrated User Support for Error Correcting.” 6th International Conference on SBPM ONE 2014, Eichstätt, Germany, April 22-23, 2014.
18. C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, “How long will it take to fix this bug?” in Proceedings of the 4th International Workshop on Mining Software Repositories. Minneapolis, Minnesota, USA: IEEE CS Press, 2007, pp. 1–8.
19. Corradini F., F. Fornari, C. Muzi, A. Polini , B. Re and F. Tiezzi (2017). “On Avoiding Erroneous Synchronization in BPMN Processes.” Int. Conf. on BIS. LNBIP, vol 288. Springer.
20. D. C. Atkinson and T. King, “Lightweight detection of program refactorings,” in Proceedings of 12th Asia-Pacific Software Engineering Conference. Taipei, Taiwan: IEEE CS Press, 2005, pp. 663–670.
21. D. Lawrie, C. Morrell, H. Feild, and D. Binkley, “Effective identifier names for comprehension and memory,” Innovations in Systems and Software Engineering, 3(4): 303–318, 2007.
22. D. Lawrie, C. Morrell, H. Feild, and D. Binkley, “What’s in a name? A study of identifiers,” in Proceedings of the International Conference on Program Comprehension, pp. 3–12, 2006.
23. D. Poshyvanyk, A. Marcus, R. Ferenc, T. Gyimothy, “Using information retrieval based coupling measures for imp act analysis. Empirical Software Engineering, 14(1), 5–32, 2009.

24. D. Ratiu, S. Ducasse, T. Gîrba, and R. Marinescu, "Using history information to improve design flaws detection," in Proceedings of the 8th European Conference on Software Maintenance and Reengineering, Tampere, Finland. IEEE Computer Society, 2004, pp. 223–232.
25. Davide Arcelli, Daniele Di Pompeo, "Applying Design Patterns to Remove Software Performance Antipatterns: A Preliminary Approach", The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017).
26. Delfmann, P., M. Steinhorst, H.-A. Dietrich and J. Becker (2015). "The generic model query language GMQL - Conceptual specification, implementation, and runtime evaluation." Inf. Syst. 47: 129-177
27. E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in Proceedings of the 31st International Conference on Software Engineering, 2009, pp. 485–495. ANTIPATTERN DETECTION: METHODS, CHALLENGES, AND OPEN ISSUES
28. E. Ligu, A. Chatzigeorgiou, T. Chaikalis, and N. Ygeionomakis, "Identification of Refused Bequest Code Smells," in Proceedings of the 29th IEEE International Conference on Software Maintenance, 2013.
29. E. Merlo, I. McAdam, and R. De Mori, "Feed-forward and re-current neural networks for source code informal information analysis," Journal of Software Maintenance, vol. 15, no. 4, pp. 205–244, 2003.
30. Elena Navarro, University of Castilla-La Mancha, Carlos E. Cuesta, King Juan Carlos University, Dewayne E. Perry, University of Texas at Austin, Pascual Gonzalez, University of Castilla-La Mancha, "Antipatterns for Architectural Knowledge Management", International Journal of Information Technology and Decision Making – May 2013.
31. F. Deissenbock and M. Pizka, "Concise and consistent naming," in Proceedings of the International Workshop on Program Comprehension, 2005.
32. F. Khomh, M. Di Penta, Y.-G. Guéhéneuc, G. Antoniol, "An exploratory study of the impact of antipatterns on class change- and fault-proneness". Empirical Software Engineering 17(3): 243-275, 2012. [32] F. Khomh, S. Vaucher, Y.G. Guéhéneuc, and H. Sahraoui, "A Bayesian Approach for the Detection of Code and Design Smells", in Proceedings of the 9th International Conference on Quality Software, 2009.
33. F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "Detecting bad smells in source code using change history information," in Proceedings of the 11th ACM/IEEE International Conference on Automated Software Engineering (ASE 2013), Silicon Valley, CA, USA. IEEE, 2013.
34. Francesca Arcelli Fontana, Jens Dietrich, Bartosz Walter, Aiko Yamashita and Macro Zanoni, Department of Informatics, Systems and Communication, University of Milano Bicocca, Milano, Italy, School of Engineering and Advanced Technology, Massey University, Palmeston North, New Zealand, "Antipattern and Code Smell False Positives: Preliminary Conceptualisation and Classification", 2016 IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering.
35. G. A. Miller, "WordNet: A lexical data base for English," Communications of the ACM, 38(11): 39-41, 1995.
36. G. Bavota, A. De Lucia, A. Marcus, R. Oliveto. "Automating Extract Class Refactoring: an Improved Method and its Evaluation," Empirical Software Engineering, To appear. PALOMBA ET AL.
37. G. Bavota, A. De Lucia, and R. Oliveto, "Identifying extract class refactoring opportunities using structural and semantic cohesion measures," Journal of Systems and Software, vol. 84, pp. 397–414, 2011.
38. G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia. "Methodbook: Recommending Move Method Refactorings via Relational Topic Models". In Transactions on Software Engineering, To appear.
39. G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," in Proceedings of 11th IEEE International Symposium on Software Metrics.Como, Italy: IEEE CS Press, 2005, pp. 20–29.

40. G. Canfora, L. Cerulo, and M. Di Penta, "Identifying changed source code lines from version repositories," in Proceedings of 4th International Workshop on Mining Software Repositories. Minneapolis, Minnesota, USA: IEEE CS Press, 2007, pp. 14–21.
41. G. Gui and P. Scott, "Coupling and cohesion measures for evaluation of component reusability," in Proceedings of the 5th International Workshop on Mining Software Repositories, 2006, pp. 18–21.
42. H. Gall, M. Jazayeri, and J. Krajewski, "CVS release history data for detecting logical couplings," in Proceedings of the 6th International Workshop on Principles of Software Evolution, 2003, pp. 13–23.
43. I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in Proceedings of the International Conference on Software Maintenance, 1998, pp. 368–377.
44. I. D. Baxter, C. Pidgeon, and M. Mehlich, "Program transformations for practical scalable software evolution," in Proceedings of the International Conference on Software Engineering, 2004, pp. 625–634.
45. L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," in Proceedings of the International Conference on Software Engineering, 2010.
46. M. Abbes, F. Khomh, Y.-G. Guéhéneuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in Proceedings of the 15th European Conference on Software Maintenance and Reengineering, Oldenburg, Germany. IEEE Computer Society, 2011, pp. 181–190.
47. M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Identification and Application of Extract Class Refactorings in Object Oriented Systems," Journal of Systems and Software, vol. 85, no. 10, pp. 2241-2260, 2012.
48. M. Fowler, "Refactoring: improving the design of existing code". Addison- Wesley, 1999.
49. M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," Journal of Systems and Software, vol. 1, pp. 213–221, 1980.
50. M. Ohba and K. Gondow, "Toward mining concept keywords from identifiers in large software projects," in Proceedings of International Workshop on Mining Software Repositories, St. Louis, Missouri, USA, 2005, pp. 1–5.
51. M. P. Robillard, W. Maalej, R. J. Walker, T. Zimmermann, Recommendation Systems in Software Engineering. Springer, 2014.
52. N. Moha, Y.G. Guéhéneuc, L. Duchien, and A.F. L. Meur, "DECOR: A method for the specification and detection of code and design smells," IEEE Transactions on Software Engineering, 36(1): 20–36, 2010.
53. N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities," IEEE Transactions on Software Engineering, vol. 35, no. 3, pp. 347–367, 2009.
54. N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou, "JDeodorant: Identification and removal of type-checking bad smell," in Proceedings of the 23rd IEEE International Conference on Software Maintenance, Paris, France, 2007.
55. R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993, pp. 207– 216.
56. R. Arcoverde, A. Garcia, and E. Figueiredo, "Understanding the longevity of code smells: preliminary results of an explanatory survey," in Proceedings of the International Workshop on Refactoring Tools, 2011, pp. 33– 36.
57. R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. Addison-Wesley, 1999.
58. R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in Proceedings of the 20th International Conference on Software Maintenance, Chicago, IL, USA. IEEE Computer Society, 2004, pp. 350–359.
59. R. Oliveto, F. Khomh, G. Antoniol, and Y.-G. Guéhéneuc, "Numerical signatures of antipatterns: An approach based on b-splines," in Proceedings of the 14th Conference on Software Maintenance and Reengineering, IEEE Computer Society Press, 2010. PALOMBA ET AL.

60. R. Peters and A. Zaidman, "Evaluating the lifespan of code smells using software repository mining," in Proceedings of the European Conference on Software Maintenance and Reengineering, 2012, pp. 411–416.
61. Ricardo Caceffo, Raysa Benatti, Breno de Franca, Tales Aparecida, Rodolfo Azevedo, Guilherme Gama, Tania Caldas, "An Antipattern Documentation about Misconceptions related to an Introductory Programming Course in C", October 2017.
62. S. Bajracharya and C. Lopes, "Mining search topics from a code search engine usage log," in Proceedings of the 6th IEEE International Working Conference on Mining SoftwareRepositories, Washington, DC, USA. IEEE Computer Society, 2009, pp. 111–120.
63. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer,R. Harshman, Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6): 391–407,1990.
64. S. Demeyer, S. Ducasse, and O. Nierstrasz, Object-Oriented Reengineering Patterns. Morgan Kaufmann Publishers Inc., 2002.
65. S. M. Olbrich, D. S. Cruzes, and D. I. K. Sjøberg, "Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems," in Proceedings of the International Conference on Software Maintenance. IEEE, 2010, pp. 1–10.
66. S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, 20(6): 476–493, 1994.
67. S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Modeling the evolution of topics in source code histories," in Proceedings of the 8th Working Conference on Mining Software Repositories, 2011, pp. 173–182.
68. T. Chen, S. W. Thomas, M. Nagappan, and A. E. Hassan, "Explaining software defects using topic models," in Proceedings of the 9th Working Conference on Mining Software Repositories, 2012.
69. T. Kamiya, S. Kusumoto, and K. Inoue, "CCfinder: a multilinguistic token-based code clone detection system for large scale source code," Transactions on Software Engineering, no. 4, 2002.
70. T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in Proceedings of the 26th International Conference on Software Engineering, 2004, pp. 563–572.
71. V. Arnaoudova, M. D. Penta, G. Antoniol, and Y.-G. Guéhéneuc, "A new family of software antipatterns: linguistic antipatterns," in Proceedings of the European Conference on Maintenance and Reengineering, Genova, Italy, 2013, pp. 187–196.
72. V. Wahler, D. Seipel, J. W. von Gudenberg, and G. Fischer, "Clone detection in source code by frequent itemset techniques," in Proceedings of the IEEE Working Conference on Source Code Analysis and Manipulation, 2004, pp. 128–135.
73. W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis", 1st ed. John Wiley and Sons, 1998.
74. W. Li and R. Shatnawi, "An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution," Journal of Systems and Software, pp. 1120–1128, 2007.
75. W. Li and S. Henry, "Maintenance metrics for the object oriented paradigm," in Proceedings of International Symposium on Software Metrics, 1993, pp. 52–60.
76. Y. Lee, B. Liang, S. Wu, and F. Wang, "Measuring the coupling and cohesion of an object oriented program based on information flow," in Proceedings of the International Conference on Software Quality, 1995, pp. 81–90.