Definitions of a Software Smell

Presented by Tushar Sharma and Diomidis Spinellis

Department of Management Science and Technology
Athens University of Economics and Business
{*tushar,dds*}*@aueb.gr*

Abstract

Many authors have defined smells from their perspective. This document attempts to provide a consolidated list of such definitions.

1. Definitions

1. Smells are certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring [5].
2. Code smells are a metaphor to describe patterns that are generally associated with bad design and bad programming practices [19].
3. Code smells are indicators or symptoms of the possible presence of design smells [12].
4. Code smells are implementation structures that negatively affect system lifecycle properties, such as understandability, testability, extensibility, and reusability; that is, code smells ultimately result in maintainability problems [6].
5. A "bad smell" describes a situation where there are hints that suggest there can be a design problem [13].
6. We define design defects as solutions to recurring problems that generate negative consequences on the quality of object-oriented systems [11].
7. Antipatterns are "poor" solutions to recurring implementation and design problems that impede the maintenance and evolution of programs [9].
8. Anti-patterns are bad solutions to recurring design problems [4].
9. An anti-pattern is a commonly occurring solution to a recurring problem that will typically negatively impact code quality. Code smells are

considered to be symptoms of anti-patterns and occur at source code level [14].

10. Antipatterns are defined as patterns that appear obvious but are ineffective or far from optimal in practice, representing worst practices about how to structure and design an ontology [15].
11. Anti-patterns are “poor” solutions to recurring design and implementation problems [10].
12. Developers often introduce bad solutions, anti-patterns, to recurring design problems in their systems and these anti-patterns lead to negative effects on code quality [7].
13. Linguistic antipatterns in software systems are recurring poor practices in the naming, documentation, and choice of identifiers in the implementation of an entity, thus possibly impairing program understanding [1].
14. Design smells are structures in the design that indicate violation of fundamental design principles and negatively impact design quality [18].
15. Code smells are indicators of deeper design problems that may cause difficulties in the evolution of a software system [20].
16. Performance Antipatterns define bad practices that induce performance problems, and their solutions [2].
17. Antipatterns are typically a commonly used set of design and coding constructs which might appear intuitive initially, but eventually may be detrimental to one or more aspects of the system [17].
18. Bad design practices at the code level are known as bad smells in the literature [8].
19. Code smells — microstructures in the program —- have been used to reveal surface indications of a design problem [3].
20. Configuration smells are the characteristics of a configuration program or script that violate the recommended best practices and potentially affect the programs quality in a negative way [16].

References

[1] Arnaoudova, V., Di Penta, M., Antoniol, G., Guéhéneuc, Y.-G., Mar. 2013. A New Family of Software Anti-patterns: Linguistic Anti-patterns. In: CSMR ’13: Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering. IEEE Computer Society, pp. 187–196.

[2] Cortellessa, V., Di Marco, A., Trubiani, C., Feb. 2014. An approach for modeling and detecting software performance antipatterns based on first-order logics. Software and Systems Modeling (SoSyM) 13 (1), 391–432.

[3] da Silva Sousa, L., May 2016. Spotting design problems with smell agglomerations. In: ICSE '16: Proceedings of the 38th International Conference on Software Engineering Companion. Pontifical Catholic University of Rio de Janeiro, ACM, pp. 863–866.

[4] Fourati, R., Bouassida, N., Abdallah, H. B., 2011. A Metric-Based Approach for Anti-pattern Detection in UML Designs. In: Computer and Information Science 2011. Springer Berlin Heidelberg, pp. 17–33.

[5] Fowler, M., 1999. Refactoring: Improving the Design of Existing Programs, 1st Edition. Addison-Wesley Professional.

[6] Garcia, J., Popescu, D., Edwards, G., Medvidovic, N., 2009. Toward a catalogue of architectural bad smells. In: Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems. QoSA '09. Springer-Verlag, pp. 146–162.

[7] Jaafar, F., Guéhéneuc, Y.-G., Hamel, S., Khomh, F., 2013. Mining the relationship between anti-patterns dependencies and fault-proneness. In: Proceedings - Working Conference on Reverse Engineering, WCRE. Ecole Polytechnique de Montreal, Montreal, Canada, IEEE, pp. 351–360.

[8] Khan, Y. A., El-Attar, M., 2016. Using model transformation to refactor use case models based on antipatterns. Information Systems Frontiers 18 (1), 171–204.

[9] Khomh, F., Vaucher, S., Guéhéneuc, Y.-G., Sahraoui, H., 2011. BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. In: Journal of Systems and Software. Ecole Polytechnique de Montreal, Montreal, Canada, pp. 559–572.

[10] Maiga, A., Ali, N., Bhattacharya, N., Sabané, A., Guéhéneuc, Y.-G., Antoniol, G., Aïmeur, E., Sep. 2012. Support vector machines for anti-pattern detection. In: ASE 2012: Proceedings of the 27th IEEE/ACM

International Conference on Automated Software Engineering. Polytechnic School of Montreal, ACM, pp. 278–281.

[11] Moha, N., Guéhéneuc, Y., Duchien, L., Meur, A. L., 2010. DECOR: A method for the specification and detection of code and design smells. IEEE Trans. Software Eng. 36 (1), 20–36.

[12] Moha, N., Guéhéneuc, Y.-G., 2007. Decor: a tool for the detection of design defects. In: ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. University of Montreal, ACM, pp. 527–528.

[13] Pérez, J., Crespo, Y., 2009. Perspectives on automated correction of bad smells. In: the joint international and annual ERCIM workshops. Universidad de Valladolid, Valladolid, Spain, ACM Press, pp. 99–108.

[14] Peters, R., Zaidman, A., 2012. Evaluating the lifespan of code smells using software repository mining. In: Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering. CSMR '12. IEEE Computer Society, pp. 411–416.

[15] Roussey, C., Corcho, O., Svab-Zamazal, O., Scharffe, F., Bernard, S., Nov. 2012. SPARQL-DL queries for antipattern detection. In: WOP'12: Proceedings of the 3rd International Conference on Ontology Patterns - Volume 929. Cemagref, CEUR-WS.org, pp. 85–96.

[16] Sharma, T., Fragkoulis, M., Spinellis, D., 2016. Does your configuration code smell? In: Proceedings of the 13th International Workshop on Mining Software Repositories. MSR'16. pp. 189–200.

[17] Sharma, V. S., Anwer, S., Jan. 2014. Performance antipatterns: Detection and evaluation of their effects in the cloud. In: Proceedings - 2014 IEEE International Conference on Services Computing, SCC 2014. Accenture Services Pvt Ltd., India, Bangalore, India, IEEE, pp. 758–765.

[18] Suryanarayana, G., Samarthyam, G., Sharma, T., 2014. Refactoring for Software Design Smells: Managing Technical Debt, 1st Edition. Morgan Kaufmann.

[19] van Emden, E., Moonen, L., Oct 2012. Assuring software quality by code smell detection. In: 2012 19th Working Conference on Reverse Engineering.

[20] Yamashita, A., 2014. Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data. Empirical Software Engineering 19 (4), 1111–1143.