Detecting and Managing Code Smells: Research and Practice

Tushar Sharma
Dept. of Management Science and Technology
Athens University of Economics and Business
Athens, Greece
tushar@aueb.gr

ABSTRACT

Code smells indicate the presence of quality problems that make the software hard to maintain and evolve. A software development team can keep their software maintainable by identifying smells and refactor them. In the first part of the session, we present a comprehensive overview of the literature concerning smells covering various dimensions of the metaphor including defining characteristics, classification, types, as well as causes and impacts of smells. In the second part, we delve into the details of smell detection methods prevailed currently both in research prototypes and industrial tools. The final part present actionable and pragmatic strategies for practitioners to avoid, detect, and eradicate smells from their codebase.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; *Software maintenance tools*;

KEYWORDS

Code smells, Antipatterns, Software quality, Code Quality, Smell detection tools, Software maintenance, Technical debt

ACM Reference Format:
Tushar Sharma. 2018. Detecting and Managing Code Smells: Research and Practice. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3183460

1 DESCRIPTION

Code smells [3, 20, 22] in a software system indicate the presence of quality problems that make the software hard to maintain and evolve. Smells not only impact maintainability [1, 11, 23], but also negatively affect other quality attributes such as reliability [2, 5, 6] and testability [14]. Given the importance of smells and their potential impact on software quality, software engineering researchers have explored smells and various dimensions associated with them in the great width and depth in the last two decades. Identifying code smells automatically and refactoring them help software engineering practitioners to keep the software maintainable [3]. Therefore, the dimension of automatically identifying smells has enjoyed active interest by the research community and appreciated by the practitioners.

We divide our session into three parts. In the first part of the session, we present a comprehensive overview of the literature by covering various dimensions of the metaphor. We present defining characteristics of smells synthesized from a comprehensive set of definitions [19] discussed in the literature. These defining characteristics are *indicator*, *poor solution*, *violates best practices*, *impacts quality*, and *recurrence*. The smell metaphor has been extended to other similar domains such as configuration management [17], spreadsheets [4], and presentations [15]. We present a summary of the types of smells described in the literature in the form of a taxonomy[1]. Smell could cause from a wide variety of factors including *lack of skill or awareness* and *frequently changing requirements*; we discuss the curated set of ten such factors that cause smells. Further, we summarize the impact of smells on people, artifact, or on process.

In the second part, we delve into the details of smell detection methods prevailed currently both in research prototypes and industrial tools. Traditionally, smells are detected by metrics-based [9] and rule-based approaches [10]. History-based [13] and optimization-based [12] approaches are alternatives that also have been employed by the community. In the recent times, machine-learning-based approaches [7] have been attempted to detect smells. We aim to present a synthesized overview of the current approaches. We touch upon the deficiencies in the current smell detection tools and techniques. These deficiencies include *false-positives and lack of context*, *limited detection support for known smells*, and *inconsistent smell definitions and detection methods* [20].

Additionally, we present the intricacies of developing a smell detection tool that we learned from developing Designite[2] [18], Puppeteer[3] [17], and DbDeo[4] [16]. Designite is software design quality assessment tool that detects smells at implementation, design, and architecture granularity. The

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5663-3/18/05. . . $15.00
https://doi.org/10.1145/3183440.3183460

[1] http://www.tusharma.in/smells/
[2] http://www.designite-tools.com
[3] https://github.com/tushartushar/puppeteer
[4] https://github.com/tushartushar/DbDeo

tool also computes various object-oriented design metrics, detects code clones, prepares DSM (Dependency Structure Matrix), shows distribution of smells in the form of treemap, and performs trend analysis to help a software developer identify issues contributing to technical debt [8] and improve maintainability of the software. The tool offers free academic licenses for all academic purposes. Currently, the tool is serving a large number of practitioners and academics worldwide.

Keeping the software maintainable is a non-trivial challenge for software development teams given the real-life challenges (such as time pressure) [21]. The final part of the session deals with such challenges and present actionable and pragmatic strategies and practices for practitioners to avoid, detect, and eradicate smells from their codebase. These strategies focus on three major pillars of software development — people, process, and tools.

The session offers contributions to both research and practice. For researchers, it provides a comprehensive overview of the domain of code smells. Also, it reveals the intricacies of developing a smell detection tool. At the same time, practitioners can learn the potential quality issues that may arise in their codebase to avoid them. Furthermore, practitioners can apply pragmatic strategies planned in this session to identify, interpret, and refactor smells.

2 SPEAKER BIOGRAPHY

Tushar Sharma is a researcher at Athens University of Economics and Business, Athens, Greece. He has more than ten years of industrial work experience including seven years at Siemens Research and Technology Center, Bangalore, India. He earned an MS degree in Computer Science from the Indian Institute of Technology-Madras, Chennai, India. He co-authored the book "Refactoring for Software Design Smells: Managing Technical Debt" published by Morgan Kaufmann in 2014. He has also co-authored two Oracle Java certification books. He has delivered talks in many academic as well as developer conferences. He is an IEEE Senior Member.

ACKNOWLEDGMENTS

This work is funded by the SENECA project, which is part of the Marie Skłodowska-Curie Innovative Training Networks (ITN-EID) under grant agreement number 642954.

REFERENCES

[1] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and David Binkley. 2012. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *IEEE International Conference on Software Maintenance, ICSM*. Universita di Salerno, Salerno, Italy, IEEE, 56–65.

[2] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2014. Are test smells really harmful? An empirical study. *Empirical Software Engineering* 20, 4 (May 2014), 1052–1094.

[3] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Programs* (1 ed.). Addison-Wesley Professional.

[4] F. Hermans, M. Pinzger, and A. van Deursen. 2012. Detecting code smells in spreadsheet formulas. In *28th IEEE International Conference on Software Maintenance (ICSM)*. 409–418. https://doi.org/10.1109/ICSM.2012.6405300

[5] Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Foutse Khomh. 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, 351–360.

[6] Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. An exploratory study of the impact of antipatterns on class change- and fault-proneness. *Empirical Software Engineering* 17, 3 (June 2012), 243–275.

[7] Foutse Khomh, Stéphane Vaucher, Yann-Gaël Guéhéneuc, and Houari Sahraoui. 2009. A Bayesian Approach for the Detection of Code and Design Smells. In *QSIC '09: Proceedings of the 2009 Ninth International Conference on Quality Software*. IEEE Computer Society, 305–314.

[8] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. 2012. Technical Debt: From Metaphor to Theory and Practice. *IEEE Software* 29, 6 (2012), 18–21.

[9] R Marinescu. 2005. Measurement and quality in object-oriented design. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*. Universitatea Politehnica din Timisoara, Timisoara, Romania, IEEE, 701–704.

[10] Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien, and Anne-Françoise Le Meur. 2010. DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Trans. Software Eng.* 36, 1 (2010), 20–36. https://doi.org/10.1109/TSE.2009.50

[11] Leon Moonen and Aiko Yamashita. 2012. Do code smells reflect important maintainability aspects?. In *ICSM '12: Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*. Simula Research Laboratory, IEEE Computer Society.

[12] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, and Katsuro Inoue. 2015. Web Service Antipatterns Detection Using Genetic Programming. In *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. Osaka University, ACM, 1351–1358.

[13] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. 2015. Mining version histories for detecting code smells. *IEEE Transactions on Software Engineering* 41, 5 (May 2015), 462–489.

[14] Aminata Sabané, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. 2013. A Study on the Relation between Antipatterns and the Cost of Class Unit Testing. In *CSMR '13: Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 167–176.

[15] Tushar Sharma. 2016. Presentation smells: How not to prepare your conference presentation. http://xrds.acm.org/blog/2016/06/presentation-smells-to-avoid-in-conference-presentation/. (2016). [Online; accessed 12-Oct-2017].

[16] Tushar Sharma, Marios Fragkoulis, Stamatia Rizou, Magiel Bruntink, and Diomidis Spinellis. 2018. Smelly Relations: Measuring and Understanding Database Schema Quality. In *Proceedings of 40th International Conference on Software Engineering (ICSE '18)*. https://doi.org/10.1145/3183519.3183529

[17] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In *Proceedings of the 13th International Workshop on Mining Software Repositories (MSR'16)*. 189–200. https://doi.org/10.1145/2901739.2901761

[18] Tushar Sharma, Pratibha Mishra, and Rohit Tiwari. 2016. Designite — A Software Design Quality Assessment Tool. In *Proceedings of the First International Workshop on Bringing Architecture Design Thinking into Developers' Daily Activities (BRIDGE '16)*. ACM. https://doi.org/10.1145/2896935.2896938

[19] Tushar Sharma and Diomidis Spinellis. 2017. Definitions of a Software Smell. (Nov. 2017). https://doi.org/10.5281/zenodo.1066135

[20] Tushar Sharma and Diomidis Spinellis. 2018. A survey on software smells. *Journal of Systems and Software* 138 (2018), 158 – 173. https://doi.org/10.1016/j.jss.2017.12.034

[21] Tushar Sharma, Girish Suryanarayana, and Ganesh Samarthyam. 2015. Challenges to and Solutions for Refactoring Adoption: An Industrial Perspective. *IEEE Software* 32, 6 (Oct. 2015), 44–51.

[22] Girish Suryanarayana, Ganesh Samarthyam, and Tushar Sharma. 2014. *Refactoring for Software Design Smells: Managing Technical Debt* (1 ed.). Morgan Kaufmann.

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