Unraveling Software Antipatterns and Smells Definitions

Richard Rivera[1], **Pamela Flores**[1], **Carlos E. Anchundia**[1], **Marcela Mosquera**[1], **Alejandro Jiménez**[1], **Xavier Carpio**[1]

[1]Departamento de Informática y Ciencias de la Computación.
Escuela Politécnica Nacional. Quito, Ecuador

{richard.rivera01, pamela.flores, carlos.anchundia
evelyn.mosquerae, alejandro.jimenez, xavier.carpio}@epn.edu.ec

Abstract. *Software technical debt may persist if the root cause is misidentified. We argue that this stems from misconceptions about antipatterns and smells. This article reviews the literature to clarify their similarities and differences. Findings suggest that both terms are used interchangeably, with "smell" being more common, though this ambiguity may affect optimal solutions in software projects.*

1. Introduction and Methodology

Antipatterns and smells concepts have become essential for identifying and mitigating design and implementation flaws. "Antipattern" was introduced by [Brown et al. 1998] to define recurring poor solutions to common software problems. On the other hand, "smells" serve as heuristics to identify potential issues in source code [Fowler et al. 2018]. Although the terms could confuse academic and industry contexts when reviewing publications, forums, and books. This research examines the current use of these terms and their practical implications through the questions: **RQ1:** Are antipatterns and smells the same? **RQ2:** What are the implications of confusing these terms?

An exploratory review of the literature was carried out, taking Fowler and Brown as references, given their proposal of concepts and classifications. In addition, the exploration included articles related to classification proposals and taxonomies of antipatterns and smells, among which Neil, Laplante, Wakem Jerzyks, and Madeysky stand out. We gather information such as definitions, classification, structure, and impact on projects.

2. Key Results

Similarities. In software project management, the terms are used as synonyms to denote technical debt issues. Also, researchers and practitioners prioritize problem-solving over conceptual distinctions [Tahir et al. 2018]. Additionally, there are antipatterns, such as "Spaghetti Code," which can be addressed based on their impact on the project. Structurally, researchers organize antipatterns and smells into catalogs, which exhibit striking similarities, potentially confusing for an inexperienced observer (Fig. 1).
Differences. We have grouped different classification schemes into "collections". At first glance, it can be seen that antipatterns focus on high-level context, while smells focus on low-level aspects of implementation (Fig. 1). It is a finding that smells such as "Blob" or "Functional Decomposition" were first classified as antipatterns by [Brown et al. 1998]. Conversely, the presence of antipatterns can be revealed through the manifestation of smells, necessitating a potential restructuring. However, a smell can be identified through various technological tools associated with basic refactoring, such as clean code principles [Sabir et al. 2019].

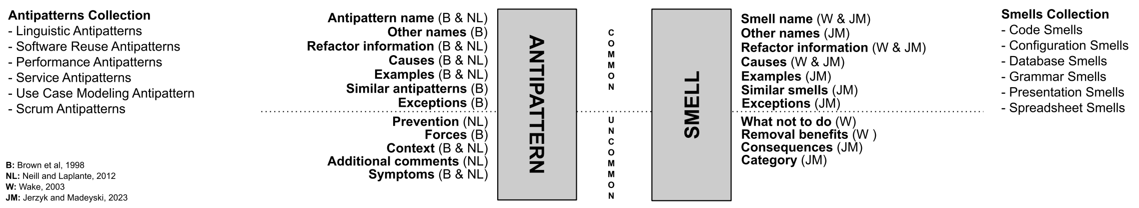

Figure 1. Antipattern and Smell relationship and analogy

3. Discussion

RQ1, antipatterns and smells are different, though both negatively impact software maintainability. Their definitions should not be overlapped, obscuring their true significance. Antipatterns and smells are related to different levels of software project issues.
RQ2, smells are like visible moisture on a wall, signaling underlying issues, whereas antipatterns resemble hidden plumbing leaks that require deeper investigation. While smells can be quickly addressed, antipatterns demand extensive structural changes and a thorough understanding of the development environment (Fig. 2).

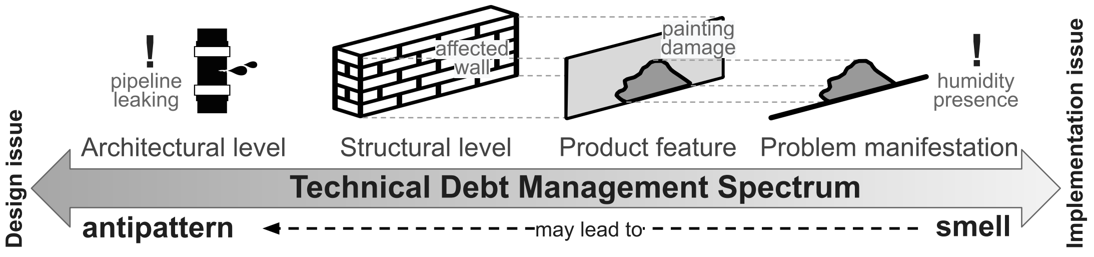

Figure 2. Antipattern and Smell relationship and analogy

4. Conclusions and Future Work

It has been examined how the terms antipattern and smell are often interpreted as similar. However, this can be a mistake when dealing with technical debt due to possible implications for treating causes within a software project. With this knowledge, future work will explore the traceability between antipatterns and smells to distinguish these concepts better.

References

Brown, W. H., Malveau, R. C., McCormick, H. W. S., and Mowbray, T. J. (1998). *AntiPatterns: refactoring software, architectures, and projects in crisis*.

Fowler, M., Beck, K., Brant, J., and Opdyke, W. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley.

Sabir, F., Palma, F., Rasool, G., Guéhéneuc, Y.-G., and Moha, N. (2019). A systematic literature review on the detection of smells and their evolution in object-oriented and service-oriented systems. *Software: Practice and Experience*, 49(1):3–39.

Tahir, A., Yamashita, A., Licorish, S., Dietrich, J., and Counsell, S. (2018). Can you tell me if it smells? a study on how developers discuss code smells and anti-patterns in stack overflow. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, EASE '18, page 68–78. ACM.