Socio-Technical Smell Dynamics in Code Samples: A Multivocal Review of Their Emergence, Evolution, and Co-Occurrence

Arthur Bueno
FACOM/UFMS
Campo Grande, Brazil
arthur.ramires@ufms.br

Bruno B. P. Cafeo
UNICAMP
Campinas, Brazil
cafeo@ic.unicamp.br

Maria Istela Cagnin
FACOM/UFMS
Campo Grande, Brazil
istela.machado@ufms.br

Awdren Fontão
FACOM/UFMS
Campo Grande, Brazil
awdren.fontao@ufms.br

ABSTRACT

Context and Motivation: Code samples play an important role in open-source ecosystems (OSSECO), serving as lightweight artifacts that support knowledge transfer, onboarding, and framework adoption. Despite their instructional relevance, these samples are often governed informally, with minimal review and unclear ownership, which increases their exposure to socio-technical degradation. In this context, the co-occurrence and longitudinal interplay of *code smells* and *community smells* become particularly relevant. While each type of smell has been studied in isolation, little is known about how community-level dysfunctions anticipate or exacerbate technical anomalies in code samples over time. **Objective:** This study investigates how code and community smells emerge, co-occur, and evolve within code samples maintained in OSSECOs. **Method:** A Multivocal Literature Review protocol was applied, encompassing 18 peer-reviewed papers and 17 practitioner-oriented sources (2013–2024). Thematic synthesis was conducted to identify recurring socio-technical patterns. **Results:** Nine patterns were identified, showing that community smells often precede or reinforce technical degradation in code samples. Symptoms such as "radio silence" and centralized ownership were frequently associated with persistent structural anomalies. Additionally, limited onboarding, the absence of continuous refactoring, and informal collaboration emerged as recurring conditions for smell accumulation. **Conclusion:** In OSSECOs, particularly within code samples, community-level dysfunctions not only correlate with but often signal maintainability decay. These findings underscore the need for socio-technical quality indicators and lightweight governance mechanisms tailored to shared instructional artifacts.

KEYWORDS

Code Smells, Community Smells, Code Samples, Software Quality, Software Evolution, Developer Communities.

1 Introduction

Code samples are important artifacts in software ecosystems, particularly in open-source environments (OSSECOs), where they are used to illustrate framework usage, propagate best practices, and support rapid onboarding of new contributors [2, 5, 24, 27]. These samples are typically small-scale, self-contained modules designed to be readable and instructional. However, despite their instructional importance, they are frequently treated as second-class citizens, often excluded from automated testing pipelines, static analysis tools, and formal review processes.

This treatment makes code samples especially vulnerable to socio-technical degradation [2]. Contributions frequently originate from external developers within the broader SECO—many of whom are unfamiliar with project-specific conventions or architectural constraints [24]. In the absence of shared norms, clearly defined roles, or structured maintenance routines, these examples may accumulate low-quality code and undocumented practices [60]. For instance, if a developer encounters a sample exhibiting a *Large Class* or missing modular structure, the lack of clarity may lead to incorrect reuse, onboarding friction, or even abandonment of a contribution attempt [46].

Technical and social degradation in such contexts can be described through the lenses of *code smells* and *community smells*. Code smells refer to structural symptoms that compromise maintainability, such as *Large Class*, *Long Method*, or *Feature Envy* [26, 29]. Community smells, in contrast, reflect patterns of collaboration breakdown, including *Lone Wolf* contributors, *Organizational Silo*, and *Radio Silence* [14, 16, 61]. These social anomalies have been associated with delayed maintenance, reduced transparency, and hindered knowledge transfer [59].

While these smells have been extensively studied in isolation, a growing body of evidence suggests they are interrelated, particularly in informally maintained artifacts like code samples. Socioorganizational dysfunctions often act as precursors to technical anomalies [43, 50], and these interdependencies are rarely captured by traditional quality assurance mechanisms. For example, in environments marked by contributor turnover or limited peer feedback, smells may persist across releases unnoticed.

Despite this potential interplay, the co-occurrence and reinforcement of code and community smells in code samples remain poorly understood. Existing studies tend to focus on core modules or monolithic systems, overlooking lightweight artifacts that play key roles in learning and adoption. Moreover, little is known about how smells evolve longitudinally or how their interactions shape sustainability and collaboration at the ecosystem level.

To address these gaps, this study applies a *Multivocal Literature Review* (MLR) [31], synthesizing findings from peer-reviewed literature and grey sources (e.g., blog posts, practitioner reports, community forums). The review focuses explicitly on the socio-technical dynamics of smells in open-source code samples and aims to uncover patterns that explain their emergence, evolution, and co-occurrence.

This investigation is guided by the following research questions:

- **RQ1:** What socio-technical conditions and practices lead to the emergence of Code Smells and Community Smells in code samples?
- **RQ2:** How do Code Smells and Community Smells evolve over time in open-source code samples, and how can this evolution be detected?
- **RQ3:** How do Community Smells signal or reinforce the emergence and persistence of Code Smells in code samples?

By analyzing these aspects, the review provides empirical foundations for the development of socio-technical quality indicators, informs preventive practices, and advances the understanding of how code samples in OSSECOs degrade in the absence of formal governance.

2 Background and Related Work

2.1 Code Smells and Community Smells

Code smells and community smells are widely acknowledged in the software engineering literature as indicators of quality degradation in software systems [56]. *Code smells* denote suboptimal design choices or structural issues within the source code that, while not inherently faulty, may negatively affect maintainability, extensibility, or comprehension over time [29]. Common examples include *Long Method*, *Large Class*, and *Feature Envy* [26], which are often addressed through refactoring practices. However, the detection of these smells—typically performed using automated tools—tends to occur without contextual consideration of the development environment or the collaborative dynamics surrounding the artifact. As a result, smell remediation is neither systematic nor guaranteed, particularly in artifacts that are loosely maintained or excluded from structured quality assurance workflows.

In contrast, *community smells* reflect socio-organizational dysfunctions within development teams, often manifesting through patterns such as *Lone Wolf*, *Organizational Silo*, and *Radio Silence* [16, 61]. These smells are typically inferred from social and contribution metadata, and have been associated with coordination breakdowns, centralized expertise, and contributor disengagement [59]. Systematic reviews and recent empirical studies [22, 49, 53] have emphasized their negative impact on collaborative efficiency, knowledge transfer, and software sustainability.

Despite the growing body of work on both categories of smells, their joint manifestation and evolution in *code samples*—lightweight, instructional artifacts—remain largely underexplored [24]. Unlike core modules, code samples are frequently developed with minimal governance, lacking defined ownership, review practices, or integration into continuous quality pipelines. Such informal conditions make them especially susceptible to the emergence and persistence of both code and community smells [19, 43]. Investigating the co-occurrence and temporal interplay of these smells in code samples, as pursued in RQ2 and RQ3, is therefore crucial to understanding how socio-technical degradation unfolds in this class of artifacts.

2.2 Code Samples in Open-source Software Ecosystems (OSSECOs)

Code samples are lightweight, self-contained artifacts meant to illustrate usage patterns, support onboarding, and promote framework adoption within OSSECOs [2, 5, 24]. Our definition of code samples [39, 40] characterizes them as complete, executable projects in official repositories, distinguishing them from isolated snippets. In this context, maintainability is a developer's ability to understand, execute, and adapt samples for learning. The observed degradation, which includes structural complexity, outdated dependencies, and misaligned configurations, directly impacts this. Our analysis isolates this degradation within code samples as distinct instructional artifacts. Their role is especially relevant in open-source contexts, where they serve as first points of contact for new contributors. However, due to their instructional nature and auxiliary status, samples are frequently excluded from quality assurance pipelines and formal review processes [17, 60, 71].

Unlike traditional systems that benefit from defined ownership and structured reviews, code samples are often maintained informally by external or occasional contributors. This leads to fragmented maintenance, high turnover, and low visibility in governance mechanisms [24, 42]. Empirical studies and practitioner accounts have shown that smells introduced in these artifacts are rarely detected or corrected, especially when the surrounding team lacks process uniformity [16, 34].

Such conditions directly relate to RQ1 and RQ3: they reveal how structural and social anomalies originate and persist in loosely governed modules. A single contributor acting as a *Lone Wolf* may introduce unresolved smells, while *Radio Silence* may allow quality regressions to propagate. Understanding these dynamics in code samples is important for capturing early signals of decay within OSSECOs.

2.3 Co-occurrence and Evolution: Gaps in Literature

Although the evolution of smells has been studied independently, the intersection between code and community smells remains insufficiently addressed. One study found that social fragmentation often precedes the accumulation of persistent structural smells, suggesting a reinforcing effect between social and technical debt [50]. Additional findings show that organizational misalignments may not only delay remediation, but also shape how smells spread across versions [43, 65].

Still, the causal mechanisms remain unclear. Some authors observe that process smells, like informal review routines, do not always result in structural anomalies [65], while others emphasize that the absence of onboarding pipelines contributes to sustained technical flaws [68]. Tools and datasets capable of tracking longitudinal evolution—such as those proposed in [13, 36]—rarely focus on non-core artifacts like code samples, leaving blind spots in monitoring practices.

Recent contributions call for integrated analyses that combine contributor behavior with smell propagation models [22, 53]. These gaps motivate RQ2 and RQ3 in this study: how do community smells affect the lifecycle of code smells, and how can such entangled phenomena be captured in shared instructional artifacts?

2.4 Motivation for a Multivocal Perspective

This review adopts a Multivocal Literature Review (MLR) approach [31, 58]. Grey sources—such as blog posts, technical forums, and community documentation—offer rich narratives on informal practices, role ambiguity, and tool skepticism that shape how smells manifest and are (not) addressed in practice.

Prior MLRs in software engineering have demonstrated that grey sources can provide early evidence of practical challenges that are only later formalized in academic research [31, 58]. This is particularly true in OSSECOs, where community-maintained artifacts are governed through hybrid (often undocumented) processes [28].

Therefore, this study complements peer-reviewed evidence with practitioner perspectives to build a comprehensive understanding of smell dynamics in code samples—particularly in light of underreported phenomena such as smell co-occurrence, informal resolution strategies, and socio-technical invisibility.

3 Research Method

This study adopts a *Multivocal Literature Review* (MLR) method to investigate how *Code Smells* and *Community Smells* emerge, co-occur, and evolve in open-source code samples.

To better articulate our methodological stance, our study adopted an aggregated, cross-ecosystem approach. While our inclusion criteria (IC2) ensured all studies were situated within OSSECOs, we deliberately did not focus on specific domains (e.g., Apache, Android). The objective was to identify generalizable socio-technical patterns that transcend any single ecosystem. This approach is supported by literature [37], which highlights the value of aggregated studies for establishing a foundational understanding of recurring phenomena. Our work thus aims to build a generalizable theory of smell co-occurrence in the broader OSSECO landscape.

From broader studies, we extracted only segments directly addressing code samples. For instance, from a general study on technical debt, we analyzed only the parts concerning sample repositories. This clarification expands upon our interpretive process, ensuring all findings are grounded in the context of instructional artifacts as defined in IC6 of our MLR protocol (Table 1).

MLRs integrate evidence from both formal academic studies and grey literature sources, providing a broader and more context-sensitive synthesis of socio-technical phenomena in Software Engineering [31, 58]. The organization and definition of our protocol were also inspired by the work of Wassouf et. al [70].

MLRs integrate evidence from both formal academic studies and grey literature sources, providing a broader and more context-sensitive synthesis of socio-technical phenomena in Software Engineering [31, 58].

The research method protocol is organized in two complementary phases:

- An **formal literature mining phase**, focused on identifying peer-reviewed studies that address the causes, evolution, and interdependencies of code and community smells in the context of open-source development;
- A **grey literature phase**, aimed at extracting practitioner-reported insights from blog posts, community forums, white papers, and technical repositories related to the practical maintenance of code samples in software ecosystems.

To ensure conceptual alignment across the review process, the *Goal-Question-Metric* (GQM) approach [20] was applied. This approach enables the explicit mapping between the overarching research goal, guiding questions, and the types of data to be extracted. Table 1 presents the structured mapping used in this study.

Table 1: Mapping of Goal, Research Questions, and Extracted Metrics (GQM)

Goal	Understand how Code Smells and Community Smells co-occur and evolve in open-source code samples.
Research Questions and Corresponding Metrics	
RQ	**Question and Extracted Metrics**
RQ1	**Question:** What socio-technical conditions and practices lead to the emergence of Code Smells and Community Smells in code samples? **Metrics:** Reported causes and triggers; collaboration models; review and maintenance practices; ownership and role structure; onboarding mechanisms.
RQ2	**Question:** How do Code Smells and Community Smells evolve over time in open-source code samples, and how can this evolution be detected? **Metrics:** Temporal analysis strategies; detection tools; persistence and decay indicators; contributor transitions; monitoring workflows.
RQ3	**Question:** How do Community Smells signal or reinforce the emergence and persistence of Code Smells in code samples? **Metrics:** Co-occurrence episodes; symptom cascades; contributor roles; unresolved issues; interaction breakdowns; reinforcement patterns.

Each phase of the MLR followed a structured and documented protocol. In the formal literature mining phase, the selection was based on a refined search string applied to IEEE Xplore, ACM Digital Library, and Scopus. The search strategy incorporated multiple thematic axes—code samples, technical debt, collaboration structures—combined using Boolean logic. A quasi-gold standard approach [31] was applied to validate coverage by checking whether a core set of highly relevant papers was retrieved.

The grey literature phase used effort-bounded Google searches [31], targeting domains such as Dev.to[1], Reddit[2], Medium, and Zenodo. This strategy involved screening the first 10 pages of results for each query and stopping when two consecutive pages yielded no new relevant sources, indicating a point of theoretical saturation for the scope of this review. Selection criteria emphasized accessibility, authorship traceability, and relevance to the guiding questions. A three-step filtering process—title/scope screening, full-text analysis, and thematic validation—was applied to select high-quality non-academic sources.

Evidence was mapped to RQs using the GQM framework (Table 1), which defines a distinct scope for each RQ. The first author conducted the synthesis via a structured, multi-phase approach. First, open coding was performed on the data using the GQM-aligned extraction form. The initial codes (e.g., "absence of peer review") were then iteratively refined over several passes to ensure consistent application. Finally, the refined codes were systematically grouped into nine patterns using axial and selective coding. Each piece of evidence was categorized by its primary focus: causal conditions (RQ1), temporal evolution (RQ2), or reinforcing mechanisms (RQ3). For example, "developer responsibility" was mapped to RQ1 when described as a lack of initial ownership, but to RQ3 when described as sustaining unresolved issues. This operational

[1]https://dev.to/
[2]https://www.reddit.com/

approach grounded all allocation decisions in the textual evidence. The complete mapping is documented in our supplementary artifact 10.

4 Formal Literature Mining

4.1 Rationale and Scope

The formal literature mining phase aimed to identify empirical studies that explore the emergence, evolution, and interplay of *Code Smells* and *Community Smells*, particularly in the context of code samples within software ecosystems. While existing research frequently isolates technical or social anomalies [50, 62], this review sought studies that examine their intersection or provide insight into socio-technical quality degradation over time.

4.2 Control Studies and Validation

To validate the coverage and focus of the search strategy, four control studies were selected based on their relevance to the research questions and recognition in the literature:

- **C1**: [50] — Persistence of code smells and the role of communication breakdowns;
- **C2**: [14] — Detection techniques for community smells in open-source ecosystems;
- **C3**: [26] — Temporal evolution of code smells in large-scale systems;
- **C4**: [62] — Socio-technical impacts of community smells.

The final query string was refined iteratively until at least three of these four studies were retrieved by default, ensuring baseline adequacy.

4.3 Search Strategy

The search string was constructed through six refinement cycles, incorporating expert feedback. It targeted three thematic axes: technical debt in instructional artifacts, socio-technical collaboration, and open-source quality practices. The final string was:

```
(“code samples” OR “sample code”) AND (“code smells”
 OR “community smells” OR “social debt” OR “technical
   debt”) AND (“developer communication” OR “team
structure” OR “developer communities” OR “open source
                      projects”)
```

This query was applied to IEEE Xplore, ACM Digital Library, and Scopus—chosen for their relevance to software engineering research and empirical studies.

4.4 Inclusion and Exclusion Criteria

Inclusion Criteria (IC):

- IC1: Empirical studies addressing code and/or community smells;
- IC2: Studies analyzing open-source projects that belong to or operate within a software ecosystem (SECO);
- IC3: Use of metrics, automated tools, or temporal data (e.g., evolution over versions);
- IC4: Studies that provide evidence relevant to at least one of the guiding research questions, including those addressing evolution and co-occurrence dynamics;
- IC5: Publications written in English;
- IC6: Studies referring explicitly to code samples or equivalent learning artifacts (e.g., documentation-attached examples), excluding trivial code samples.

Exclusion Criteria (EC):

- EC1: Theoretical, position, or review papers lacking empirical evidence;
- EC2: Papers without full-text access;
- EC3: Studies focusing on non-software domains or unrelated artifacts (e.g., education, robotics, hardware).

4.5 Selection Process and Corpus Definition

The selection process was conducted in four stages (Figure 1), each applying the inclusion and exclusion criteria with increasing depth:

1. **Initial Retrieval:** 63 documents were retrieved from the three databases;
2. **Deduplication:** Duplicates across databases were identified by metadata matching and removed. The deduplication step focused on eliminating identical studies indexed differently across platforms. This yielded 37 unique entries for screening;
3. **Title and Abstract Screening:** 28 studies were excluded based on topical misalignment (e.g., education, unrelated artifacts). Nine candidate studies remained;
4. **Full-text Screening and Snowballing:** All nine studies were retained. Backward and forward snowballing identified nine additional studies, resulting in a final corpus of 18.

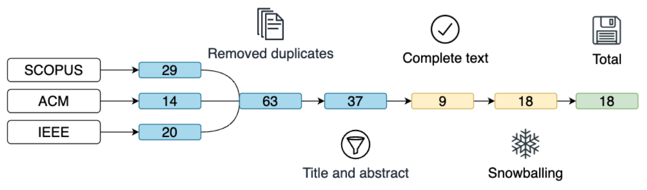

Figure 1: Formal literature selection process

This corpus offers a diverse but focused representation of how code and community smells are studied in open-source contexts, particularly regarding their emergence, evolution, and socio-technical interdependencies.

4.6 Included Studies

The final corpus comprises 18 empirical studies (S1–S18) [15, 21, 23, 25, 30, 32, 35, 38, 41, 44, 45, 48, 51, 56, 57, 63, 69], each selected through systematic screening and full-text analysis. The studies were coded thematically and mapped to the research questions according to their methodological and conceptual relevance.

Each study was coded based on full-text analysis and categorized by its alignment with the research questions. The mapping is shown in Table 2.

5 Findings from Formal Literature

The analysis of the 18 selected peer-reviewed studies reveals not only isolated instances of socio-technical degradation but a web of

Table 2: Studies mapped to each Research Question

Research Question	Studies
RQ1: Socio-technical conditions and practices leading to smell emergence	S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S12, S13, S14, S15, S16, S17
RQ2: Evolution and detection of code and community smells	S1, S3, S4, S5, S6, S8, S10, S11, S12, S13, S14, S15, S16, S18
RQ3: Community smells as precursors or reinforcers of technical degradation	S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18

interdependent conditions that sustain and reinforce smell dynamics in code samples. While the results are structured according to each research question (RQ), the cross-cutting connections among them, particularly the patterns of neglect, invisibility, and informal governance that jointly underpin the emergence, evolution, and entanglement of Code and Community Smells.

5.1 RQ1 – Socio-technical conditions and practices leading to smell emergence

***Fragmented communication and lack of structured review*:**The lack of structured communication and review mechanisms in the maintenance of code samples facilitates a dual-layered degradation: technical anomalies are introduced without contextual scrutiny, and social disconnection prevents their later identification or correction. Studies **S2**, **S4**, and **S6** show that asynchronous contributions, are rarely subject to peer validation. This leads not only to unchecked quality issues but also to an erosion of collective memory, as illustrated by excerpts like: "several community examples are uploaded with minimal or no peer review…" [S2]. With this, developers are unable to trace the rationale behind structural decisions, and smells become normalized through reuse. This aligns with findings from [50, 59] and complements observations by [58] on the invisibility of quality work. Notably, the absence of communication may not merely precede technical degradation but act as a social smell in itself, co-occurring with structural anomalies. Given the frequent replication of code samples, such degradation may propagate, leading to what could be framed as socio-technical smell contagion [68]. For instance, a developer copying a flawed code sample for a new tutorial inadvertently propagates its smells, infecting a new part of the ecosystem's knowledge base. Therefore, fragmented communication does not only allow smells to survive—it creates the conditions for them to spread and multiply within and beyond their original scope.

***Centralized ownership and bottlenecks*:**Evidence from **S3**, **S4**, and **S10** reveals that when code samples are predominantly maintained by a single developer or a small subset of contributors, opportunities for collaborative quality control are significantly reduced. This is reinforced by practitioners who report that such centralization creates bottlenecks, with one stating, "When only one person knows how to maintain the examples, updates can take weeks, stalling community contributions" [D5]. Studies **S8** and **S12** reinforce that lack of distributed participation correlates with long-term stagnation, where smells persist not due to unawareness, but due to social dynamics that discourage intervention. This phenomenon aligns with concerns raised in open-source governance literature [17, 34], which distinguish between healthy stewardship and exclusive gatekeeping. Over time, centralized ownership transforms code samples into socially protected but technically vulnerable artifacts—untouched not because they are robust, but because they are perceived as someone else's responsibility.

***Devaluation of instructional artifacts*:**Despite the availability of static analysis tools such as Designite or SonarQube, studies **S1**, **S5**, **S13**, and **S14** reveal that these technologies are rarely applied to code samples. **S7** suggests that samples are commonly excluded from analysis pipelines due to their illustrative or instructional framing. This exclusion reflects a broader cultural dissonance: while tools exist to detect code anomalies, the artifacts in question are socially perceived as too trivial to merit scrutiny. This perception not only prevents refactoring but also strips samples of their pedagogical potential, transforming them into carriers of outdated or flawed design idioms. Moreover, the integration of tooling into collaborative workflows is rarely adapted to accommodate auxiliary modules, leaving a blind spot in quality assurance. The persistent underutilization of automated tools in these contexts is thus not a technical limitation, but a socio-cultural one.

5.2 RQ2 – Evolution and detection of code and community smells

Building on the emergence factors discussed above, this section examines how those same conditions shape the trajectory of smell persistence. The longitudinal dimension of quality assessment exposes another axis of exclusion: even when tools and practices exist to detect smells, they often fail to encompass code samples.

***Exclusion from longitudinal tracking*:**Despite advancements in tracking code quality over time, the longitudinal evolution of smells in code samples remains largely uncharted. Studies **S3**, **S6**, and **S14** employ snapshot-based approaches to map technical debt trajectories, yet they overwhelmingly target core modules, overlooking instructional artifacts. **S4** and **S16** confirm that auxiliary code is seldom included in quality assessments, resulting in a silent accumulation of long-lived smells. This omission stems from both technical and cultural factors: code samples often exist outside standard monitoring pipelines, lack change frequency, and are excluded from versioning metadata. **S18** advocates the use of contributor metadata to support evolutionary modeling, but such techniques face friction when applied to socially fragmented and historically shallow artifacts like samples. These gaps underscore a broader issue: existing quality models are structurally biased toward artifacts that conform to traditional development patterns.

***Onboarding gaps and turnover*:**Studies **S5**, **S11**, and **S15** demonstrate that contributor transitions—particularly the departure or reallocation of key maintainers—interrupt ongoing remediation cycles in code samples. When onboarding structures are absent or informal, newcomers lack the contextual knowledge to recognize smells, let alone address them. Study [S11] provides evidence for this, noting that "in projects with high turnover and no documented sample guidelines, 60% of smell-introducing commits for sample code came from developers with less than three months of tenure." These findings echo broader insights from software engineering research [43, 64], which link role turnover to socio-technical misalignment. In the context of code samples, the impact is compounded: code samples are often seen as entry points for new contributors, yet their complexity and lack of ownership deter sustained engagement.

Over time, unresolved smells become normalized, and instructional modules devolve into technical liabilities—under-maintained not due to irrelevance, but due to their exclusion from community routines of care.

***Underutilization of hybrid metrics*:**Although studies **S1**, **S8**, and **S12** propose hybrid metrics that integrate technical indicators with social signals (e.g., contributor churn), such approaches remain largely underutilized. **S14** emphasizes the importance of temporally and contextually aware indicators, yet mainstream tools persist in treating code quality as a purely structural concern. This structural bias overlooks the socio-organizational conditions under which smells emerge and persist. Moreover, while calls for hybrid monitoring abound [31, 68], operational frameworks that translate these metrics into actionable insights are rare. The disjunction between detection capability and governance practice reflects a deeper issue: without integrating collaboration dynamics into quality strategies, instructional modules remain outside the scope of preventive or corrective action.

5.3 RQ3 – Community smells as precursors or reinforcers of technical degradation

Expanding on the previous RQs, this section delves into the reciprocal nature of socio-technical smells. Community Smells do not merely coincide with Code Smells—they actively enable, reinforce, and are sustained by them.

***Social fragmentation and structural flaws*:**Studies **S3**, **S4**, **S6**, and **S9** identify collaboration breakdowns—particularly *Radio Silence* and *Organizational Silos*—as consistent antecedents to complex code smells. These social fractures often render code samples analytically invisible: they are not discussed, not owned, and thus not maintained. **S14** and **S15** add that technical and social anomalies frequently co-occur, especially in modules with unclear accountability. This convergence reinforces recent work on smell entanglement [22, 53], where dysfunction in coordination not only fails to prevent smells, but actively creates the organizational void in which they persist. Unlike core modules that benefit from structured ownership and review, code samples exist at the margins—where social fragmentation functions less as an incidental condition and more as a systemic enabler of technical debt.

***Governance voids and degradation loops*:**The absence of governance structures—such as defined roles or routine peer review—produces governance voids where smells not only emerge but persist through cycles of disengagement. Studies **S1**, **S10**, and **S13** show that in these contexts, contributors perceive code samples as high-cost, low-reward artifacts. **S7** and **S4** document how this perception reinforces neglect: smells discourage engagement, and disengagement reinforces smell persistence. Over time, this modules devolve into "dead zones" of maintenance, where structural anomalies go unobserved and unaddressed. These dynamics reflect the nature of socio-technical debt as not merely an accumulation of flaws, but as a systemic feedback loop of invisibility, dis-ownership, and abandonment [19, 58].

***Degraded examples and community health*:**Studies **S2**, **S8**, **S12**, and **S17** highlight that technically flawed code samples lacking maintainer responsiveness are frequently avoided, abandoned, or rewritten. **S16** emphasizes that such artifacts, once deprecated, lose their instructional function, severing an important bridge for newcomers into the ecosystem. These degraded examples not only fail to teach—they actively repel. The lack of upkeep communicates exclusion, reducing contributor confidence and weakening the social scaffolding that underpins collaborative growth. Over time, this erodes the community's cognitive continuity and onboarding efficacy. Rather than being inert failures, smell-laden samples operate as negative signals—markers of organizational entropy and entry barriers—especially when formal remediation routines are absent [2, 5].

6 Grey Literature Review (GLR)

6.1 Method and Source Selection

To complement the findings from the academic literature and address contextual gaps regarding code samples in open-source software ecosystems, a Grey Literature Review (GLR) was conducted. This approach aimed to integrate practitioner insights on the emergence, evolution, and co-occurrence of *Code Smells* and *Community Smells* in open-source *code samples*. In particular, the GLR focused on scenarios involving decentralized maintenance, informal collaboration, and tooling limitations—conditions often absent from formal publications.

The review followed multivocal literature guidelines [31], incorporating both academic and non-academic sources. These guidelines emphasize the relevance of grey sources in domains where industrial experience precedes theoretical consolidation, and where informal practices shape software quality. In the context of OSSECOs, and especially in artifacts such as code samples, technical debt is frequently discussed in blog posts, community reports, and development forums rather than peer-reviewed channels.

Search Strategy. The search was conducted using Google, following the "effort-bounded" strategy described in [31]. This strategy involved screening the first 10 pages of results for each query and stopping when two consecutive pages yielded no new relevant sources, indicating a point of theoretical saturation for the scope of this review. Queries combined terms related to software quality, code samples, and socio-technical smells, such as `"code smell"`, `"community smell"`, `"developer communication"`, `"refactoring"`, and `"open source"`.

Results were constrained to publicly accessible content written in English or Portuguese, published between 2013 and 2024. Priority was given to sources that demonstrated practitioner credibility (e.g., named authors, affiliations) and direct relevance to the research questions. **Source Types.** The review considered four main categories:

- Technical blogs (e.g., Dev.to, Medium);
- Community discussions (e.g., Reddit, Stack Overflow threads);
- Reports and datasets in community repositories (e.g., Zenodo, GitHub releases);
- Informal white papers or preprints authored by industry professionals.

Platforms such as Dev.to and Reddit were prioritized due to open access, high engagement, and traceable authorship. Sources with paywalls or anonymous/unverifiable content were excluded.

Inclusion and Filtering. An initial corpus of 120 documents was retrieved. The screening process followed three stages as show on Figure 2, and final corpus consisted of 17 documents.

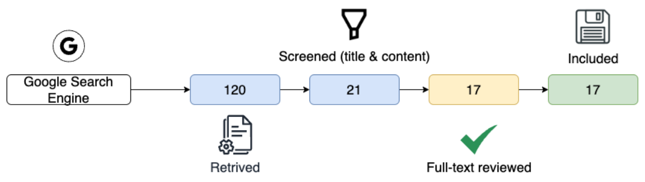

Figure 2: Selection process for the Grey Literature Review (GLR)

(1) *Preliminary scan*: Titles and samples were analyzed for thematic alignment;
(2) *Full-text review*: Each candidate was read to assess clarity, empirical grounding (e.g., examples, metrics, tools), and relevance to the RQs;
(3) *Validation*: Only documents with observable socio-technical content and alignment with code sample contexts were retained.

6.2 Corpus Overview

Seventeen practitioner-oriented documents were selected and analyzed D1-D17 [1, 3, 4, 6–12, 18, 47, 52, 54, 55, 66, 67]. Each source was thematically coded and classified according to its type, relevance, and alignment with the guiding research questions. Table 3 summarizes the mapping of documents to the research questions.

Table 3: Grey literature documents mapped to each Research Question

Research Question	Documents
RQ1: What socio-technical conditions and practices lead to the emergence of Code Smells and Community Smells in code samples?	D1, D3, D4, D5, D6, D8, D10, D14, D16, D17
RQ2: How do Code Smells and Community Smells evolve over time in open-source code samples, and how can this evolution be detected?	D1, D2, D3, D6, D8, D9, D11, D13, D15, D16, D17
RQ3: How do Community Smells signal or reinforce the emergence and persistence of Code Smells in code samples?	D2, D4, D5, D7, D10, D11, D12, D13, D14, D15, D16, D17

7 Findings from Grey Literature

The review of 17 practitioner-oriented documents revealed a rich but fragmented landscape of socio-technical concerns surrounding the emergence, persistence, and co-occurrence of *Code Smells* and *Community Smells* in open-source code samples. These sources—blogs, white papers, discussion threads, and technical evaluations—offered grounded insights into the practical realities faced by developers. Unlike the structured rigor of formal research, the grey literature surfaces experiential knowledge, tool-based reflections, and implicit observations of governance failures. Although explicit references to Community Smells were rare, many documents captured social decay patterns congruent with those described in academic work.

7.1 RQ1 – Socio-technical conditions and practices leading to smell emergence

***Ownership ambiguity and lack of stewardship*:**Sources such as **D1**, **D5**, and **D8** expose a recurring pattern in open-source repositories: code samples are frequently contributed without any mechanism for long-term stewardship. This absence of clear ownership generates ambiguity regarding who holds responsibility for maintaining, reviewing, or improving these artifacts. In many cases, even when flaws are identified, contributors hesitate to intervene, not out of lack of competence, but due to uncertainty over authority and accountability boundaries. As illustrated in **D14**, a widely reused snippet with known flaws remained unaltered for several iterations simply because, in the words of a contributor, "no one owns the fix."

This form of socio-technical inertia exemplifies a condition that might be described as a "latent decay zone": an artifact whose deterioration is not due to active neglect, but to the passive absence of governance. Over time, such zones normalize quality erosion. The samples, become central to documentation and onboarding, embedding smells into the developer experience and creating invisible technical debt. These observations align with findings in formal literature regarding informal governance structures and their impact on diferent modules [17, 34], but the practitioner narratives here emphasize the emotional and cognitive cost of contributing to ambiguous artifacts—where taking initiative feels risky and unrewarded.

***Reactive and undervalued quality control*:**Documents such as **D3**, **D4**, and **D7** describe a recurrent dynamic in the treatment of instructional code: quality control is neither preventive nor systematic, but instead activated only after a visible failure occurs. Refactoring efforts are framed as reactive responses to bugs, user confusion, or documentation drift—rather than as part of a proactive maintenance culture. This temporal deferral does not indicate a lack of awareness, but rather reflects a triage-based prioritization in which production-facing code absorbs most of the available attention and effort. As noted in **D11**, one contributor remarks: "We'll clean it up next time we teach this," suggesting that instructional artifacts are implicitly considered transient or disposable—even when reused across multiple contexts.

This perception undermines their inclusion in formal quality assurance routines. The result is a two-tiered system in which code samples are excluded from the normative expectations of code hygiene. Such asymmetries perpetuate technical anomalies not by accident, but by design—implicitly codifying a division between what deserves care and what can be "fixed later."

***Tool distrust and workflow misalignment*:**Practitioner accounts from **D9** and **D13** reflect a pattern of skepticism toward the applicability and reliability of automated quality assessment tools in the context of instructional or auxiliary code. Tools such as SonarQube and CodeScene are often described as producing excessive false positives or lacking the contextual awareness necessary to assess examples fairly. This sentiment was echoed in a community thread, where a developer mentioned, "We had to disable Sonar for the /examples folder. The noise was overwhelming, and every report felt like a false positive in our context" [D9]. As a result, contributors frequently exclude example files from analysis pipelines or disable specific rule checks. These practices, while pragmatic, reflect a deeper misalignment between tooling assumptions and the informal nature of code samples. This dynamic mirrors observations in the formal literature, where the mere presence of detection tools does not translate into systematic use—particularly in artifacts considered marginal or temporary.

***Neglect of pedagogical intent*:**Several practitioner-oriented sources implicitly reveal a divergence between the initial instructional purpose of code samples and the absence of mechanisms to preserve that purpose over time. Documents **D6** and **D15**, for instance, describe scenarios in which tutorials or technical blogs are published with illustrative code that aligns closely with a given toolset or framework version. However, after publication, these examples are seldom maintained. As dependencies evolve, syntactic conventions shift, or APIs deprecate, the code remains static—causing it to become outdated or even misleading to new learners. This form of degradation transcends simple technical obsolescence. It undermines the artifact's pedagogical credibility, transforming what was once a didactic resource into a potential source of confusion or error. The issue is not that the code ceases to compile, but that it ceases to teach effectively.

7.2 RQ2 – Evolution and detection of code and community smells

***Exclusion from longitudinal quality strategies*:**Several practitioner sources indicate that while software quality is increasingly monitored through temporal metrics—such as commit frequency, churn, and defect density—these efforts are overwhelmingly focused on core production code. For instance, **D3** discusses how quality metrics are tracked in enterprise projects, but makes no mention of auxiliary artifacts like code samples. The dataset in **D2** further reinforces this exclusion: although aimed at supporting smell detection at scale, it omits code examples due to their inconsistent structure and lack of traceable authorship. This omission is not merely a matter of pipeline configuration; it reveals a systemic misclassification of code samples as static or disposable. In **D13**, CodeScene is critiqued for failing to contextualize small-scale scripts, contributing to what one user terms a "blind spot in smell surveillance." Consequently, code samples become analytically invisible, despite being versioned and circulated across repositories.

***Recognized temporal drift without diagnosis*:**In contrast to this analytical invisibility, developers consistently report informal recognition of smell accumulation in code samples—often after these artifacts cause misunderstanding or friction. In **D11**, an illustrative tutorial was only flagged as problematic after users misapplied it, prompting a late-stage revision. While tools exist for historical smell analysis [1], they are seldom applied to code samples due to a lack of structured ownership, tagging, or annotated histories. Even in projects with version control, as observed in **D15**, developers rarely document the rationale behind updates to examples, making it difficult to reconstruct their semantic evolution. This absence of longitudinal traceability leads to what **D1** frames as "temporal opacity": developers know that decay is occurring but cannot localize its onset or causes. As a result, maintenance becomes reactive.

7.3 RQ3 – Community smells as precursors or reinforcers of technical degradation

***Social abandonment and maintenance voids*:**The practitioner sources (**D4**, **D5**, **D10**) describe the progressive neglect of auxiliary code—particularly instructional examples—as project priorities shift. Over time, these artifacts lose alignment with the main system, yet remain in the repository. **D8** illustrates that even small contributions aimed at correcting outdated examples often receive no feedback, creating a perception that such modules are no longer under community care. This form of neglect is not merely technical but social: it signals that these files fall outside any active stewardship. The concept of "maintenance voids" refers to zones in the codebase where no individual or group assumes responsibility for upkeep. The invisibility is often unintentional, but the absence of interaction, ownership, or responsiveness turns these artifacts into dormant components that accumulate technical debt.

***Unclear roles and mentorship erosion*:**Practitioner accounts from **D14** and **D16** reveal that contributors often hesitate to engage with instructional modules due to unclear social protocols. In these contexts, developers express uncertainty regarding review expectations, contribution boundaries, and ownership. Questions such as "who maintains this?" or "what constitutes a valid change here?" remain unanswered, leading to inaction. This hesitation illustrates a deeper issue: the absence of mentorship structures and role transparency. When contributors cannot discern what is expected of them, psychological friction increases. These dynamics correspond with recognized community smells, particularly *Weak Onboarding* and *Unclear Roles*, which hinder integration and suppress initiative.

***Low-quality samples as negative signals*:**Documents **D15** and **D17** illustrate how technically degraded instructional examples convey more than technical shortcomings. For prospective contributors, these artifacts function as proxies for project health and governance culture. Rather than serving as onboarding tools, such examples deter engagement by signaling a disconnect between community ideals and practical maintenance routines. This phenomenon reflects a negative social signal: a structural artifact that communicates unintentional messages about community priorities. A developer on a forum [D15] stated this clearly: "When I see a project with sloppy, outdated examples, I immediately question the quality of the core product and the attentiveness of the maintainers. It's a huge red flag." When example files appear neglected, the implied message is one of disengagement. This perception undermines trust not only in the module itself, but in the broader development process.

***Perceived disorganization*:**Documents **D11** and **D12** describe situations where small inconsistencies in code samples—such as outdated library references or naming mismatches—accumulate and foster a broader sense of project disorganization. These technical inconsistencies may appear trivial in isolation, yet function as collective red flags suggesting a lack of coordination or oversight. This perception of disarray undermines what can be termed collaborative perception—the belief that a given artifact is maintained through a coherent community process. When examples appear inconsistent, contributors may infer that broader collaboration practices are equally fragmented, reducing their willingness to engage.

8 Threats to Validity

This review is subject to known limitations associated with multivocal studies. Potential threats are: ***Search and Retrieval*** - In the academic phase, three major digital libraries were used. Still, relevant studies may have been excluded due to terminology variation. In the grey literature phase, results were influenced by Google's ranking, which may limit source diversity. To address this, effort-bounded strategies were used and searches were conducted in English and Portuguese, across multiple iterations. ***Selection and Inclusion*** - The study followed predefined inclusion and exclusion criteria across both literature types. For the grey literature, source credibility was assessed based on traceable authorship and thematic relevance. Nonetheless, grey sources often lack consistent structure, and some judgment calls were needed during screening and coding. ***Synthesis and Interpretation*** -Thematic synthesis involves interpretation, which introduces the risk of bias. To mitigate this, coding was conducted independently for formal and grey sources, and only integrated in the final synthesis stage. Practitioner quotes were paraphrased to preserve anonymity and analytical consistency. ***Transferability*** - Findings are drawn from open-source environments and may not generalize to closed or enterprise settings. Additionally, grey sources represent only publicly shared practices. However, the convergence of findings across literature types suggests that the identified patterns may reflect broader socio-technical dynamics.

9 Discussion and Implications

This section consolidates and interprets the findings from both literature streams. First, we present a summary of the nine identified patterns that serve as the foundation for our analysis (Table 4). Following this, we synthesize these patterns across the research questions, connecting them to broader socio-technical theories. Finally, we derive concrete implications for key stakeholders within OSSECOs.

9.1 Cross-RQ Synthesis

The integrated synthesis of formal and grey literature reveals that the emergence, persistence, and co-occurrence of *Code Smells* and *Community Smells* in open-source code samples are not isolated events. Rather, they represent the entangled outcome of recurring socio-technical conditions that span the three research questions.

In relation to RQ1, the findings indicate that **governance gaps**—including undefined ownership, informal review practices, and ad hoc quality routines—are key enablers of smell emergence. Formal studies such as **S1**, **S3**, and **S6** show that code samples under centralized or ambiguous stewardship are less likely to receive peer validation or undergo refactoring. Grey literature sources (**D1**, **D4**, **D5**) reinforce this by highlighting contributor perceptions that these artifacts are "unclaimed," which inhibits intervention and promotes decay. Governance, in this context, functions not merely as an organizational layer, but as a structural mediator of both technical and social degradation.

Regarding RQ2, the review identifies a persistent **visibility paradox**. Although code samples are widely reused and disseminated (**S9**, **D6**), they are often excluded from quality assurance pipelines (**S7**, **D13**) and smell detection mechanisms (**D9**, **D13**). Tools are available, but their assumptions and configurations are misaligned with the nature of instructional code. Consequently, smells in these artifacts persist not due to the absence of detection capability, but due to a lack of contextual trust and routine adoption. This gap, as noted in **S14** and **D3**, results in "invisible debt zones" where quality decay remains analytically hidden.

In response to RQ3, the co-evolution of code and community smells emerges as a systemic rather than incidental phenomenon. Empirical evidence (**S2**, **S4**, **S5**) shows that social fragmentation—such as *radio silence*, role ambiguity, and siloed communication—often precedes technical decay. These findings are echoed in grey sources (**D10**, **D14**, **D16**), where contributors describe scenarios of social abandonment that leave instructional artifacts without oversight. Such conditions reinforce the notion of **socio-technical incongruence**, in which organizational structures are misaligned with the artifact's intended lifecycle. This phenomenon can be seen as a manifestation of Conway's Law, where the fragmented communication structure of the community is mirrored in the fragmented and decaying quality of its artifacts.

Finally, across all RQs, the findings converge on the existence of **feedback loops of degradation**. Initial neglect—driven by centralized ownership (**S3**, **D8**) or misaligned processes (**S6**, **D7**)—leads to contributor disengagement (**S17**, **D14**), which in turn perpetuates invisibility and decay. Grey literature further enriches this picture by exposing subjective and symbolic dimensions, such as hesitancy to intervene (**D14**), absence of mentorship (**D16**), and negative signaling from degraded examples (**D15**, **D17**). These recursive dynamics are rarely addressed by current models of quality assurance, suggesting that smells in code samples are not merely technical artifacts—but indicators of deeper structural and cultural misalignment in open-source ecosystems.

Notably, the multivocal approach was critical in uncovering these dynamics. While formal studies (e.g., S1, S3, S6) identified structural issues like *governance gaps*, the grey literature (e.g., D1, D9, D14) provided the crucial context behind them: contributor hesitancy, tool distrust, and the perception of artifacts as "unclaimed." This synthesis demonstrates that a traditional SLR would have identified the symptoms, whereas the MLR revealed the underlying socio-technical drivers, thus providing a more holistic and actionable understanding.

9.2 Implications for Practitioners, Managers, and Researchers

The results highlight that code samples, play a strategic role in open-source ecosystems. Their neglect reflects not only technical debt but also organizational fragility. Treating code samples as a first-class citizen is essential to support onboarding, sustain community health, and ensure long-term maintainability.

For managers and maintainers, our findings provide actionable guidelines for targeted governance. We recommend that they: (i) assign explicit stewards to ensure accountability (e.g., via `CODEOWNERS` files); (ii) conduct periodic reviews for both technical correctness and instructional quality; (iii) establish lightweight verification pipelines (e.g., adapted static analysis, build validation); and (iv) maintain structured contribution workflows to manage community

Table 4: Synthesis of the Nine Socio-Technical Patterns Identified

Pattern Name	Description	Primary Evidence
	RQ1: Emergence Conditions	
1. Fragmented Communication	Lack of structured review enables unverified contributions, which introduce and normalize smells.	S2, S4, S6, D14
2. Centralized Ownership	A single developer or a small group acts as a bottleneck, reducing collaborative quality control and causing technical stagnation.	S3, S4, S10, D1, D5
3. Tooling Exclusion & Distrust	Instructional artifacts are culturally devalued and excluded from static analysis pipelines, often due to perceived noise or false positives.	S1, S7, D9, D13
	RQ2: Evolution and Persistence	
4. Longitudinal Invisibility	Code samples are omitted from long-term quality tracking, which allows smells to accumulate silently over time.	S4, S16, D2, D3
5. Onboarding Gaps	High contributor turnover, combined with a lack of mentorship, obstructs remediation cycles as newcomers lack contextual knowledge.	S5, S11, D14, D16
6. Reactive Refactoring	Quality control is reactive and activated only after a visible failure, rather than being a proactive maintenance practice.	S7, S1, D3, D4, D7
	RQ3: Co-occurrence and Reinforcement	
7. Social Abandonment	A lack of stewardship turns examples into "maintenance voids," where social inaction leads to recursive technical degradation.	S3, S9, D8, D10
8. Governance Voids	The absence of defined roles and routines creates feedback loops where flawed samples discourage engagement, which in turn reinforces decay.	S1, S10, D14, D16
9. Negative Social Signaling	Degraded examples act as negative proxies for project health, eroding contributor trust and weakening community onboarding.	S17, D15, D17

feedback and preserve quality. These actions directly translate our identified patterns into practice.

For practitioners and developers, the nine patterns serve as a diagnostic framework. For instance, to counter "Reactive Refactoring," a developer can configure a bot like Dependabot for automated updates. To mitigate "Exclusion from Tooling", a developer can create a dedicated, lenient configuration profile for static analysis tools, ensuring meaningful quality assurance without the noise of false positives.

For tool developers, standard quality tools are miscalibrated for code samples, which prioritize instructional clarity over production-level complexity. This creates excessive false positives, causing maintainers to ignore the results. This limitation exposes a measurement gap, as current tools are not calibrated for instructional artifacts [39]. Incorporating social indicators—such as ownership gaps or contributor turnover—can improve detection in these low-visibility artifacts.

10 Conclusion and Future Work

This multivocal review analyzed how *Code Smells* and *Community Smells* emerge, persist, and intersect in code samples within open-source software ecosystems. Based on 18 peer-reviewed studies and 17 practitioner-oriented documents, the synthesis identified socio-technical patterns that sustain degradation in these code samples.

Despite their small size and instructional purpose, code samples exhibit recurrent neglect due to centralized ownership, informal governance, and absence from structured quality assurance workflows. These conditions contribute to sustained smell accumulation and signal deeper coordination and stewardship gaps. In several cases, technical decay aligned with organizational fragility, reflecting a lack of role clarity and breakdowns in onboarding routines.

The evidence challenges the notion that code samples are inconsequential to ecosystem health. Instead, their condition often anticipates broader risks to sustainability, trust, and contributor engagement. Smells in code samples are rarely isolated—they indicate latent structural and social tensions.

Future research should aim to operationalize the identified patterns into hybrid indicators that combine structural metrics (e.g., smell lifespan, ownership dispersion) with behavioral signals (e.g., contributor churn, review latency). Empirical validation across diverse repositories can assess generalizability, while the design of lightweight interventions—such as peer review routines tailored to auxiliary code—may support early detection and mitigation.

Finally, the verification of authors in the grey literature was a quality control step to ensure source credibility, as per MLR guidelines [31]. An analysis of differing author perceptions was outside our study's scope. However, we agree this is a valuable research direction. Understanding these perceptions would require direct methods, like surveys or interviews. We have added this as a promising avenue for future work.

ARTIFACT AVAILABILITY

The supplementary artifact containing the full mapping of sources, textual evidence, and thematic codes is openly available on Figshare at: https://doi.org/10.6084/m9.figshare.29441441.v1.

ACKNOWLEDGMENTS

This work was funded by the Federal University of Mato Grosso do Sul and by Brazilian funding agencies CAPES (Finance Code 001), FAEPEX (grants: 3404/23 and 2382/24) and the Brazilian funding agency FUNDECT (Call No. 42/2024).

REFERENCES

[1] 2013. *Detecting Bad Smells in Source Code using Change History Information.* https://www.slideshare.net/slideshow/historical-smells/26643190

[2] 2019. *Framework Code Samples: How Are They Maintained and Used by Developers?*

[3] 2019. *Research on Software Project Developer Behaviors.* https://trepo.tuni.fi/bitstream/handle/10024/119685/Research_on_software_project_2019.pdf?sequence=2&isAllowed=y

[4] 2020. *MLCQ: Industry-relevant code smell data set.* https://zenodo.org/records/3666840

[5] 2022. *Assessing the Impact of Code Samples Evolution on Developers' Questions.*

[6] 2022. *Evolving collaboration, dependencies, and use in the Open Source Software ecosystem.* https://www.nature.com/articles/s41597-022-01819-z

[7] 2023. *Code Smells and Anti-Patterns: Signs You Need to Improve Code Quality.* https://blog.codacy.com/code-smells-and-anti-patterns

[8] 2023. *Is Code Smell/Code Quality software really useful?* https://www.reddit.com/r/learnprogramming/comments/16fvmfq/is_code_smellcode_quality_software_really_useful/?rdt=56172 Reddit thread.

[9] 2023. *The Pandora's box of social, process, and people debts in software engineering.* https://onlinelibrary.wiley.com/doi/epdf/10.1002/smr.2516

[10] 2023. *Understanding Code Smell Detection in Software Development.* https://teamhub.com/blog/understanding-code-smell-detection-in-software-development/

[11] 2024. *CodeScene – Top Ten Important Things You Need To Know.* https://dotcommagazine.com/2024/05/codescene-top-ten-important-things-you-need-to-know/

[12] 2024. *A Novel, Tool-Supported Catalog of Community Smell Symptoms.* https://advance.sagepub.com/users/830264/articles/1224141/master/file/data/_JSEP_24__A_Novel__Tool_Supported_Catalog_of_Community_Smell_Symptoms/_JSEP_24__A_Novel__Tool_Supported_Catalog_of_Community_Smell_Symptoms.pdf

[13] Nuri Almarimi, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2023. csDetector: An Open Source Tool for Community Smells Detection. ETS Montreal, University of Quebec.

[14] Nuri Almarimi, Ali Ouni, and Mohamed Wiem Mkaouer. 2020. Learning to detect community smells in open source software projects. *Knowledge-Based Systems* 205 (2020), 106201.

[15] Nuri Almarimi, Ali Ouni, and Mohamed Wiem Mkaouer. 2020. Learning to detect community smells in open source software projects. *Knowledge-Based Systems* 204 (2020), 106201.

[16] Nuri Salem Almarimi, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2022. Improving the detection of community smells through socio-technical and sentiment analysis. *Journal of Software: Evolution and Process* 34, 9 (2022), e2505.

[17] Giusy Annunziata, Carmine Ferrara, Stefano Lambiase, Fabio Palomba, Gemma Catolino, Filomena Ferrucci, and Andrea De Lucia. 2023. An Empirical Study on the Relation between Programming Languages and the Emergence of Community Smells. In *Proceedings of the 2023 International Conference on Software Engineering and Advanced Applications (SEAA)*. Salerno, Italy.

[18] Antigoni. 2023. *Community smells like bacon.* https://medium.com/@_antigoni_/community-smells-like-bacon-dfe0af8db5c6

[19] Ahmed Baabad, Hazura Zulzalil, Sa'adah Hassan, and Salmi Baharom. 2020. Software Architecture Degradation in Open Source Software: A Systematic Literature Review. *IEEE Access* 8 (2020), 174109–174129.

[20] Victor R. Basili. 1994. GQM approach has evolved to include models. *IEEE Software* 11, 1 (1994), 8–8.

[21] Isela Macia Bertran. 2011. Detecting architecturally-relevant code smells in evolving software systems. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. Association for Computing Machinery, 1090–1093.

[22] H-M Chen, R. Kazman, G. Catolino, M. Manca, D. Tamburri, and W-J van de Heuvel. 2024. An Empirical Study of Social Debt in Open-Source Projects: Social Drivers and the "Known Devil" Community Smell. In *Proceedings of the 57th Hawaii International Conference on System Sciences (HICSS)*. University of Hawaii and TU Eindhoven, 7239–7248.

[23] Eleni Constantinou and Tom Mens. 2017. Socio-technical evolution of the Ruby ecosystem in GitHub. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 34–44.

[24] Matheus Albuquerque de Melo. 2023. *A Multi-Faceted Analysis of How Organizations Create and Maintain Code Samples.* Master's thesis. Universidade Federal de Mato Grosso do Sul, Campo Grande, Brazil.

[25] Beyza Eken, Francis Palma, Başar Ayşe, and Tosun Ayşe. 2021. An empirical study on the effect of community smells on bug prediction. *Software Quality Journal* 29, 1 (2021), 159–194.

[26] Francesca Arcelli Fontana, Vincenzo Ferme, Alessandro Marino, Bartosz Walter, and Pawel Martenka. 2013. Investigating the Impact of Code Smells on System's Quality: An Empirical Study on Systems of Different Application Domains. , 260–269 pages.

[27] Awdren Fontão, Sergio Cleger-Tamayo, Igor Wiese, Rodrigo Pereira dos Santos, and Arilo Claudio Dias-Neto. 2023. A Developer Relations (DevRel) model to govern developers in Software Ecosystems. *Journal of Software: Evolution and Process* 35, 5 (2023), e2389.

[28] Awdren Fontão, Pedro Paes, Oswald Ekwoge, Rodrigo Pereira Dos Santos, and Arilo Claudio Dias-Neto. 2020. Evaluating Processes to Certify Mobile Applications During Developer Relations Activities. *IEEE Access* 8 (2020), 137462–137476. https://doi.org/10.1109/ACCESS.2020.3009921

[29] Martin Fowler and Kent Beck. 2018. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional, Boston. 45–65 pages.

[30] Oscar Franco-Bedoya, David Ameller, Dolors Costal, and Xavier Franch. 2017. Open source software ecosystems: A Systematic mapping. *Information and Software Technology* 91 (2017), 160–185.

[31] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology* 106 (2019), 101–121. https://doi.org/10.1016/j.infsof.2018.09.006

[32] Zijie Huang, Huiqun Yu, Guisheng Fan, Zhiqing Shao, Ziyi Zhou, and Mingchen Li. 2024. On the effectiveness of developer features in code smell prioritization: A replication study. *J. Syst. Softw.* 210, C (2024), 16 pages.

[33] Zijie Huang, Huiqun Yu, Guisheng Fan, Zhiqing Shao, Ziyi Zhou, and Mingchen Li. 2024. On the effectiveness of developer features in code smell prioritization: A replication study. *Journal of Systems and Software* 210 (2024), 111968.

[34] Stefano Lambiase. 2024. Cultural and Socio-Technical Aspects in Software Development. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024)*. ACM, 1–6.

[35] Stefano Lambiase. 2024. Investigating Cultural Dispersion: on the Role of Cultural Differences in Software Development Teams. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24)*. Association for Computing Machinery, 187–191.

[36] Riasat Mahbub, Mohammad Masudur Rahman, and Muhammad Ahsanul Habib. 2024. On the Prevalence, Evolution, and Impact of Code Smells in Simulation Modelling Software.

[37] Konstantinos Manikas. 2016. Revisiting software ecosystems Research: A longitudinal literature study. *Journal of Systems and Software* 117 (2016), 84–103. https://doi.org/10.1016/j.jss.2016.02.003

[38] Gabriel Menezes, Willian Braga, Awdren Fontão, Andre Hora, and Bruno Cafeo. 2022. Assessing the Impact of Code Samples Evolution on Developers' Questions. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering (SBES '22)*. Association for Computing Machinery, 321–330.

[39] Gabriel Menezes, Bruno Cafeo, and Andre Hora. 2019. Framework Code Samples: How Are They Maintained and Used by Developers?. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11. https://doi.org/10.1109/ESEM.2019.8870139

[40] Gabriel Menezes, Bruno Cafeo, and Andre Hora. 2022. How are framework code samples maintained and used by developers? The case of Android and Spring Boot. *Journal of Systems and Software* 185 (2022), 111146. https://doi.org/10.1016/j.jss.2021.111146

[41] Karolina Milano and Bruno Cafeo. 2024. Navigating Expertise in Configurable Software Systems through the Maze of Variability. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 450–454.

[42] Behnaz Moradi-Jamei, Brandon L. Kramer, J. Bayoa´ n Santiago Caldero´ n, and Gizem Korkmaz. 2021. Community Formation and Detection on GitHub Collaboration Networks. In *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 244–252.

[43] Haris Mumtaz, Paramvir Singh, and Kelly Blincoe. 2022. Analyzing the Relationship between Community and Design Smells in Open-Source Software Projects: An Empirical Study. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Helsinki, Finland, 11.

[44] Haris Mumtaz, Paramvir Singh, and Kelly Blincoe. 2022. Analyzing the Relationship between Community and Design Smells in Open-Source Software Projects: An Empirical Study. In *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '22)*. Association for Computing Machinery, 23–33.

[45] Biruk Asmare Muse, Mohammad Masudur Rahman, Csaba Nagy, Anthony Cleve, Foutse Khomh, and Giuliano Antoniol. 2020. On the Prevalence, Impact, and Evolution of SQL Code Smells in Data-Intensive Systems. In *17th International Conference on Mining Software Repositories (MSR '20)*. Association for Computing Machinery, 327–338.

[46] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What Makes a Good Code Example? A Study of Programming Q&A in StackOverflow. In *2012 IEEE International Conference on Software Maintenance (ICSM)*. 25–34.

[47] F. Palomba. 2022. *When and Why Your Code Starts to Smell Bad.* https://fpalomba.github.io/pdf/Conferencs/C4.pdf

[48] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2021. Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? . *IEEE Transactions on Software Engineering* 47, 01 (2021), 108–129.

[49] Fabio Palomba, Damian A. Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2021. Beyond Technical Aspects: How

Do Community Smells Influence the Intensity of Code Smells? *IEEE Transactions on Software Engineering* 47, 1 (2021), 108–129.

[50] Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, Andy Zaidman, Francesca Arcelli Fontana, and Rocco Oliveto. 2018. How do community smells influence code smells?. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 240–241.

[51] Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, Andy Zaidman, Francesca Arcelli Fontana, and Rocco Oliveto. 2018. How do community smells influence code smells?. In *40th International Conference on Software Engineering: Companion Proceeedings (ICSE '18)*. Association for Computing Machinery, 240–241.

[52] Juliana Pereira. 2023. *How to track software quality metrics?* https://medium.com/@juli1pb/how-to-track-software-quality-metrics-4965507d77c

[53] Antonio Della Porta, Stefano Lambiase, Gemma Catolino, Filomena Ferrucci, and Fabio Palomba. 2024. A Novel, Tool-Supported Catalog of Community Smell Symptoms. (Sept. 2024).

[54] Better Programming. 2023. *Dealing With Code Smells and Metrics in Complex Software*. https://betterprogramming.pub/dealing-with-code-smells-and-metrics-in-complex-software-879f9fc4a767

[55] Joabe Ramone. 2023. *Code Smells, não deixa para depois*. https://dev.to/joaberamone/code-smells-nao-deixa-para-depois-5gnb

[56] Anshul Rani and Jitender Kumar Chhabra. 2017. Evolution of code smells over multiple versions of softwares: An empirical investigation. In *2017 2nd International Conference for Convergence in Technology (I2CT)*. 1093–1098.

[57] Jaswinder Singh, Anu Gupta, and Preet Kanwal. 2023. The vital role of community in open source software development: A framework for assessment and ranking. *Journal of Software: Evolution and Process* 36 (12 2023).

[58] Jacopo Soldani, Damian A. Tamburri, Alexander Chatzigeorgiou, and Paris Avgeriou. 2021. On the use of grey literature to improve software architecture knowledge. *Empirical Software Engineering* 26, 4 (2021), 1–34.

[59] Manuel De Stefano, Emanuele Iannone, Fabiano Pecorelli, and Damian Andrew Tamburri. 2021. Impacts of Software Community Patterns on Process and Product: An Empirical Study. In *Proceedings of the 2021 International Conference on Software Engineering (ICSE)*. 244–252.

[60] Swaminathan Subramanian, Reid Holmes, and Robert J. Walker. 2014. Feature-specific Usage Scenarios for API Learning. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 992–1001.

[61] Damian Andrew Tamburri, Fabio Palomba, and Rick Kazman. 2021. Exploring Community Smells in Open-Source: An Automated Approach. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING* (2021).

[62] Damian A. Tamburri, Fabio Palomba, and Rick Kazman. 2021. Exploring Community Smells in Open-Source: An Automated Approach. *IEEE Transactions on Software Engineering* 47, 3 (2021), 630–652.

[63] Damian A. Tamburri, Fabio Palomba, and Rick Kazman. 2021. Exploring Community Smells in Open-Source: An Automated Approach. *IEEE Transactions on Software Engineering* 47, 3 (2021), 630–652.

[64] Stuti Tandon, Vijay Kumar, and V. B. Singh. 2024. Study of Code Smells: A Review and Research Agenda. *International Journal of Mathematical, Engineering & Management Sciences* 9, 3 (2024), 472.

[65] Erdem Tuna, Carolyn Seaman, and Eray Tüzün. 2024. Do code reviews lead to fewer code smells? *Journal of Systems and Software* 215 (2024), 112101.

[66] u/ExperiencedDevs. 2023. *Code smell in feature factories*. https://www.reddit.com/r/ExperiencedDevs/comments/1bq1jds/code_smell_in_feature_factories/ Reddit post.

[67] u/stealthmusic. 2023. *Enterprise Software Development*. https://dev.to/stealthmusic/enterprise-software-development-383o

[68] Liang Wang, Ying Li, Jierui Zhang, and Xianping Tao. 2022. Quantitative Analysis of Community Evolution in Developer Social Networks Around Open Source Software Projects.

[69] Liang Wang, Ying Li, Jierui Zhang, and Xianping Tao. 2022. Quantitative Analysis of Community Evolution in Developer Social Networks Around Open Source Software Projects. https://arxiv.org/abs/2205.09935

[70] Edvaldo R. Wassouf, Débora Paiva, Kiev Gama, and Awdren Fontão. 2025. The Developer Experience of LGBTQIA+ People in Agile Teams: A Multivocal Literature Review. In *2025 IEEE/ACM 18th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE)*. 15–26.

[71] Aiko Yamashita and Leon Moonen. 2013. Do Developers Care about Code Smells? An Exploratory Survey.