Evolutionary Analysis of the Co-occurrence between Code Smells and Community Smells in Code Samples

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

Carla Bezerra
UFC
CE, Brazil
carlailane@ufc.br

Hudson S. Borges
FACOM/UFMS
MS, Brazil
hudson.borges@ufms.br

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

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

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

Abstract

Context: Code samples are widely used to demonstrate best practices and facilitate framework adoption, promoting productivity and innovation. In open-source projects, their evolution often involves diverse contributions, introducing technical complexity and challenges in community collaboration. This scenario can leads to the co-occurrence of *Code Smells* and *Community Smells*, impacting software quality, collaboration, and the sustainability of developer communities. **Problem:** The simultaneous evolution of *Code Smells* and *Community Smells* degrades code quality, reduces developer cohesion, and lowers collaborative efficiency, increasing the likelihood of project failure and making projects less appealing to new contributors. **Solution:** This study investigates how the co-occurrence and evolution of *Code Smells* and *Community Smells* interact over time. It proposes strategies based on metrics and temporal analysis to mitigate these impacts, improve maintainability, and strengthen community dynamics. **Information Systems Theory:** Social Network Theory provides a foundation to analyze the influence of human interactions on project evolution and team cohesion. **Method:** We mined open-source repositories—using techniques such as commit history mining, bug log analysis, and code metric extraction—and temporal analysis to quantitatively track and correlate the evolution of *Code Smells* and *Community Smells* across project milestones. **Results Summary:** Local commit surges trigger transient complexity spikes; geographic dispersion reduces cohesion; and diverse teams correlate with controlled code growth. Overall, smells interact to degrade maintainability and collaboration. **Contributions:** Addressing the SBSI Grand Challenges (2016–2026), this study integrates technical and social perspectives, fostering sustainable practices in open-source projects and enhancing the resilience and effectiveness of developer communities.

CCS Concepts

• **Software and its engineering → Software configuration management and version control systems**; *Software maintenance tools*; **Open source model**.

Keywords

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

1 Introduction

Software ecosystems (SECOs) involve diverse communities of developers, fostering collaboration and innovation in technology development. It is worth mentioning that our study builds on recent evidence highlighting the importance of ecosystem dynamics [11]. Within these ecosystems, *code samples* play a crucial role as artifacts designed to demonstrate best practices, facilitate framework adoption, and simplify the use of core technologies [9, 13]. By providing concrete examples, code samples not only accelerate knowledge transfer and lower learning curves but also promote adherence to coding standards [21]. Prominent examples include those in the Java ecosystem, where code samples support frameworks and development libraries.

Code samples evolve continuously through contributions from internal developers—those affiliated with organizations managing the ecosystem—and external developers, such as independent framework users. These contributions often necessitate interactions between these actors, fostering a collaborative environment. However, such interactions may encounter challenges that give rise to *community smells*—social or organizational dysfunctions that disrupt collaboration and cohesion within development communities [32], as observed by Zhifei Chen et al [8].

This study hypothesizes that *community smells* impact the technical quality of code samples by contributing to the accumulation of *code smells*—design or structural flaws that undermine code maintainability, readability, and evolvability [33]. Specifically, we posit that these two classes of smells not only coexist but also mutually amplify each other's negative impacts, creating a vicious cycle that accelerates quality deterioration and community fragmentation. Understanding the interplay between code smells and community smells is essential for preserving the quality of code samples and sustaining their role as effective vehicles for knowledge transfer and collaboration [8]. Without proactive management, these issues risk compounding over time, potentially reducing community engagement and threatening the long-term sustainability of the ecosystem [25].

To address this gap, this study analyzes the simultaneous evolution of community smells and code smells in open-source projects. **Specifically, we aim to (i) identify the practices and decisions that foster the emergence of these smells and (ii) propose a systematic way to track their evolution over time.** In pursuit of these goals, we define two research questions focusing on the

conditions under which these smells arise and how they evolve longitudinally.

The main contributions of this study are:

- **Interconnected Dynamics of Smells:** *Code Smells* and *Community Smells* mutually reinforce each other, leading to software quality degradation over time.
- **Empirical Evidence from Open-Source Java Repositories:** Our temporal analysis of 170 open-source Java repositories reveals that geographically dispersed teams tend to have lower code cohesion, while clustering analysis shows that team diversity can mitigate code complexity growth—provided structured collaboration practices are in place.
- **Commit Activity and Complexity Fluctuations:** Although increased commit activity does not uniformly raise complexity, local surges coincide with transient increases in code smells.
- **Implications for Proactive Interventions:** These findings suggest that proactive interventions, such as structured code reviews and adaptive communication protocols, can disrupt the amplification cycle of technical and social issues.

The remainder of this paper is structured as follows: Section 2 introduces the background on Code Smells and Community Smells. Section 3 outlines the research methods employed. Section 4 presents the results and analysis. Section 5 discusses threats to validity, and Section 6 reviews related work. Finally, Section 7 concludes with the study's contributions, limitations, and avenues for future research.

```
private void detectAndFrame(final Bitmap imageBitmap) {
    ByteArrayOutputStream outputStream = new
         ByteArrayOutputStream();
    imageBitmap.compress(Bitmap.CompressFormat.JPEG, 100,
          outputStream);
    ByteArrayInputStream inputStream =
        new ByteArrayInputStream(outputStream.toByteArray
             ());

    AsyncTask<InputStream, String, Face[]> detectTask =
        new AsyncTask<InputStream, String, Face[]>() {
            String exceptionMessage = "";

            @Override
            protected Face[] doInBackground(InputStream
                 ... params) {
                try {
                    publishProgress("Detecting...");
                    Face[] result = faceServiceClient.
                         detect(
                        params[0],
                        true,  // returnFaceId
                        false, // returnFaceLandmarks
                        null   // returnFaceAttributes
                    );
                    return result;
                } catch (Exception e) {
                    exceptionMessage = String.format(
                        "Detection failed: %s", e.
                             getMessage()
                    );
                    return null;
                }
            }
        };
}
```

Figure 1: Example of a Java code sample for the Cognitive Services Face API.

2 Background

2.1 Code Samples

Code samples are indispensable in software ecosystems, streamlining the adoption of frameworks, libraries, and best practices by providing clear, practical examples [9]. They simplify complex technical documentation—which is often extensive and challenging to navigate [22]—and serve as tangible benchmarks for both learning and empirical research in Mining Software Repositories (MSR). For example, an Android code sample used for detecting and framing faces with the Cognitive Services Face API[1], as illustrated in Figure 1, not only demonstrates technical implementation but also provides insight into best practices and usage patterns.

The primary advantage of code samples is their practicality, allowing developers to integrate new technologies without extensive manual reading. They cover use cases ranging from simple RESTful APIs to complex features such as authentication and database interactions [9]. Furthermore, code samples contribute valuable data to MSR studies by revealing trends in developer adoption, effective practices, and usage patterns [21].

2.2 Mining Software Repositories (MSR)

Mining Software Repositories (MSR) involves applying data mining techniques to software repositories to identify patterns and derive metrics that enhance our understanding of development processes [14]. Common methods include commit history mining, bug log analysis, and code metric extraction. In our study, these techniques—specifically commit history mining, bug log analysis, and code metric extraction—were systematically applied to analyze the evolution of smells. **We employ these three techniques in our analysis (mentioned above), without incorporating additional advanced methods, to maintain a focused and reproducible evaluation of smell evolution.** Despite challenges such as incomplete data and project heterogeneity, recent advances in artificial intelligence and deep learning have improved the robustness and scalability of MSR techniques [4, 20].

2.3 Code Smells

Code Smells represent design or implementation issues that, although they may not immediately cause bugs, degrade software maintainability and increase the effort required for future modifications [12]. They are particularly prevalent in large-scale or rapidly evolving projects where structural flaws violate fundamental design principles. To facilitate objective detection, standardized metrics such as Lines of Code (LOC), Weighted Method Count (WMC), and Lack of Cohesion in Methods (LCOM) are used. Table 1 summarizes the key Code Smells analyzed in this study, along with their definitions and associated metrics.

For instance, the *God Class* is a classic code smell where a single class accumulates too many responsibilities, complicating refactoring and increasing the risk of errors [12]. Figure 2 illustrates this smell, showing how high LOC and WMC are indicative of its presence.

[1] https://github.com/Azure-Samples/cognitive-services-vision-face-finder.git

Table 1: Code Smells Considered in This Study

Code Smell	Definition
God Class	A class that centralizes excessive responsibilities, violating the Single Responsibility Principle [2]. Typically detected by high LOC and WMC.
Feature Envy	A class that depends excessively on data and methods from other classes, indicating poor encapsulation and high coupling [36].
Duplicated Code	Repeated code fragments across the system, leading to maintainability issues. Measured through redundancy analysis and high LOC in similar code blocks [5].

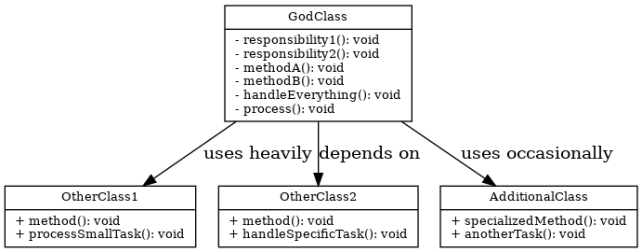

Figure 2: Example of a Code Smell: The God Class.

2.4 Co-occurrence of Code Smells

The co-occurrence of Code Smells refers to the phenomenon where multiple smells appear together in the same code section, thereby amplifying their overall negative impact on software quality [20]. For example, a class exhibiting both *God Class* and *Duplicated Code* smells poses significantly higher maintenance challenges. Empirical studies have demonstrated that such co-occurrences increase refactoring complexity and contribute to technical debt [18]. **This focus on co-occurrence underpins our hypothesis of a mutual amplification effect between technical and social issues.**

2.5 Community Smells

Community Smells are indicators of social or organizational dysfunctions within software development teams that hinder effective collaboration [31]. Factors such as poor communication, lack of coordinated decision-making, and geographic dispersion are measured using quantitative metrics like AuthorCount and TimezoneCount. Table 2 outlines the Community Smells considered in this study, along with their definitions. By quantifying these social factors, our approach provides an objective basis for evaluating community dynamics and their impact on software quality.

For example, the *Lone Wolf* pattern (see Figure 3) exemplifies a situation where a developer works in isolation, which can lead to inconsistencies and hinder knowledge transfer [32]. Addressing such issues is crucial for maintaining team cohesion and ensuring sustainable project development [34].

Table 2: Community Smells Considered in This Study

Community Smell	Definition
Organizational Silo	Siloed areas of the development community that do not communicate, except through one or two of their respective members [3].
Black Cloud	Excessive information overload due to a lack of structured communication or governance [26].
Lone Wolf	A developer who applies changes in the source code without considering the opinions of their peers, working in isolation [6].

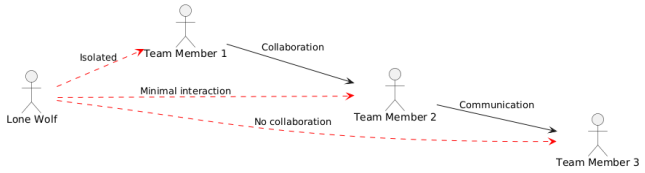

Figure 3: Example of a Community Smell: The Lone Wolf [4].

2.6 Evolutionary Analysis

Evolutionary Analysis examines how Code Smells and Community Smells develop and interact over time as code samples evolve. By analyzing successive snapshots of a system's lifecycle, this approach reveals recurring trends—such as the persistence, resolution, or amplification of smells—that are critical for identifying intervention points [19]. Although the interplay between technical and social issues might seem intuitive, our quantitative analysis provides empirical evidence that these smells can mutually reinforce one another, thereby justifying proactive maintenance strategies. This analysis is essential for developing proactive maintenance strategies SECOs [16].

3 Research Method

This study (Figure 4) investigates the co-occurrence and temporal evolution of Code Smells and Community Smells in open-source Java code samples, focusing on their impact on code quality and team dynamics. We address two primary research questions:

(RQ1): What development and management practices lead to the emergence of Code Smells and Community Smells in code samples?

Motivation: To identify specific development decisions and team behaviors that foster the emergence of technical and social flaws. *Metrics:* For Code Smells, we use *LOC*, *WMC*, and *LCOM* to quantify structural deficiencies. For Community Smells, we employ *CommitCount*, *AuthorCount*, and *TimezoneCount* to capture collaboration patterns and geographic dispersion.

(RQ2): How can the evolution of these smells be monitored and measured throughout a project's lifecycle?

Motivation: To capture longitudinal changes that reveal co-evolution

patterns, thus enabling early detection of adverse trends and targeted interventions.
Metrics: We track the same Code Smell (LOC, WMC, LCOM) and Community Smell (CommitCount, AuthorCount, TimezoneCount) metrics introduced in RQ1 across different snapshots of the project timeline to observe shifts in both structural and social attributes.

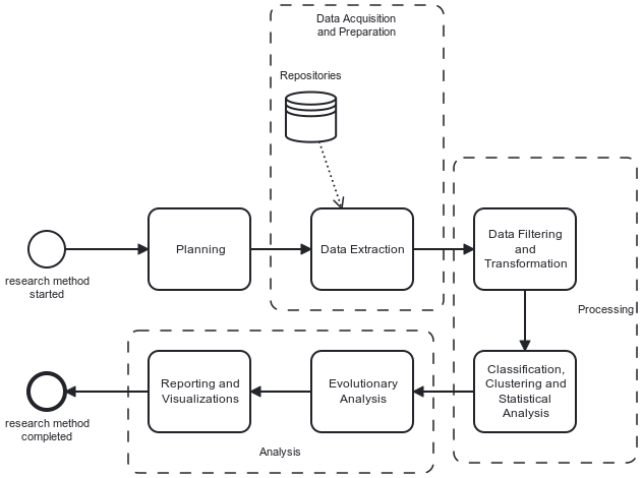

Figure 4: Research Method for Planning, Data Extraction, Processing, and Analysis.

3.1 Planning

To ensure consistency, validity, and replicability, our planning phase includes:

(1) **Language and Repository Selection:**
Java was chosen for its widespread adoption and robust ecosystem [17]. Repositories were selected following established MSR practices [13, 29] based on the following criteria:
- **Evolution Time:** A minimum of six years of documented evolution to capture long-term trends while remaining practical [30].
- **Contributors:** At least five contributors, ensuring meaningful community dynamics.
- **Codebase Size:** Between 500 and 100,000 LOC, to exclude trivial projects and avoid excessively complex monolithic systems (in line with constraints from tools such as SonarQube [7]).
- **Update Frequency:** Only actively maintained repositories (updated within the last year) were included.
- **Archived Status:** Archived projects were excluded.

From an initial pool of 345 repositories, 170 provided usable data after accounting for tool limitations.

(2) **Tool Selection:**
We utilized *csDetector* [4] to detect Community Smells and *Designite* [28] to identify Code Smells. These tools were carefully calibrated and manually validated on a sample to mitigate inherent biases [27].

(3) **Temporal Analysis Approach:**
A snapshot-per-release strategy was adopted, as recommended in longitudinal software evolution studies [10, 32]. To standardize data from repositories with varied release dates, we normalized the timeline into annual "project years" (starting from the first commit). This approach ensures consistent cross-project comparisons and robust tracking of smell evolution [7].

3.2 Metrics Description and Justification

The following metrics were selected to quantitatively assess the prevalence and evolution of smells:

Code Smells Metrics:

- **LOC:** High LOC values suggest excessively large classes or duplicated fragments, common in *God Class* and *Duplicated Code*.
- **WMC:** A high Weighted Method Count indicates complex methods, often a hallmark of a *God Class*.
- **LCOM:** Low cohesion (high LCOM) signals poor encapsulation, a key aspect of *Feature Envy*.

Community Smells Metrics:

- **CommitCount:** Elevated commit frequency can lead to a *Black Cloud* when unstructured activity overwhelms communication channels.
- **AuthorCount:** Low contributor diversity is associated with the formation of *Organizational Silo*.
- **TimezoneCount:** High geographic dispersion, measured via TimezoneCount, may foster *Lone Wolf* behavior; however, recent studies indicate that asynchronous communication tools can mitigate these effects [27].

The consistent application of these metrics across all snapshots ensures that our findings are reproducible and based on objective, standardized criteria.

3.3 Data Extraction and Processing

3.3.1 Repository and Snapshot Collection. Focusing on open-source Java code samples allowed us to control for language-specific variables. Snapshots were captured at each project release using a custom PowerShell script [2]. To harmonize data from repositories with different release schedules, we normalized the timeline into annual snapshots (defining a "project year" from the first commit), enabling consistent longitudinal analysis [30].

3.3.2 Detection and Storage. For each snapshot, we executed *csDetector* and *Designite* to detect, respectively, Community and Code Smells. The identified smells were stored chronologically, forming a structured dataset that facilitates further statistical and clustering analyses.

3.3.3 Data Filtering and Normalization. Outliers (values exceeding ±2 standard deviations) were removed to emphasize significant trends [35]. Code Smell frequencies were normalized by dividing by the number of classes or LOC [12], and Community Smells by the number of active contributors [4], ensuring cross-project comparability.

[2] https://github.com/arthurramires/replication-package-sbsi/tree/main/Scripts

3.3.4 Statistical and Clustering Analysis. Pearson correlation analysis was employed to identify significant relationships between smells, prioritizing correlations above 0.5 [21, 23]. In addition, k-means clustering was applied to the normalized data (after Min-Max scaling and removal of outliers) to discern natural groups within the dataset. The optimal number of clusters was determined using the elbow method, which resulted in three distinct clusters representing different co-occurrence patterns [30].

3.4 Replication Package

All raw and processed data, along with the detection tools and scripts, are available in a publicly accessible replication package[3]. This package includes:

- Detailed repository selection criteria and links.
- Release-based snapshots capturing the evolution of Code and Community Smells.
- Detection tools and configuration scripts with comprehensive documentation.
- Normalized, filtered datasets and outputs from clustering and statistical analyses.
- Complete lists of detected smells per snapshot.

This comprehensive replication package guarantees full transparency and reproducibility of our research methods.

4 Results and Analysis

This section presents a comprehensive quantitative analysis of the co-occurrence and temporal evolution of Code Smells and Community Smells across 170 open-source Java repositories. Based on 1,020 snapshots—normalized into annual "project years" (see Section 3.3.1, [15])—the analysis examines both individual metric trends and interrelations through correlation and clustering techniques.

4.1 Temporal Evolution of Smells

Complexity and Commit Activity:
Figure 5 shows the annual average (`WMC_mean`) Weighted Method Count versus Commit Count (`CommitCount_mean`). Although the overall Pearson correlation is moderately negative ($r = -0.22$), suggesting that increased commit activity does not uniformly raise complexity, local spikes in commit activity coincide with transient surges in `WMC_mean`. This pattern aligns with prior findings in software evolution [15], where rapid development phases can introduce structural inefficiencies—potentially leading to *God Class* code smells. The removal of outliers (values beyond ±2 standard deviations) was essential to ensure that these trends reflect typical project dynamics rather than extreme cases.

Such localized increases in complexity underscore the importance of proactive refactoring strategies, particularly during periods of intense commit activity, to mitigate the accumulation of technical debt. Regular code reviews and automated complexity monitoring can help prevent uncontrolled growth in complex code structures. Consequently, while higher commit frequency does not inherently lead to complexity inflation, close attention to local spikes can help maintain code quality over a project's lifecycle.

[3]https://github.com/arthurramires/replication-package-sbsi.git

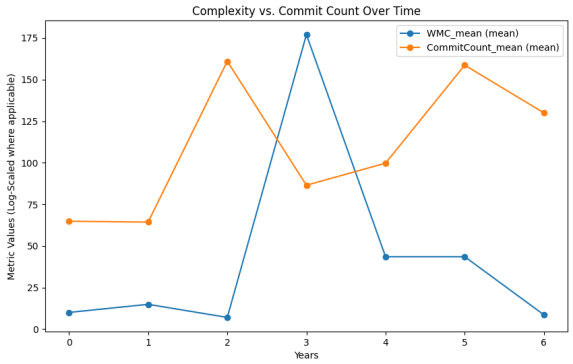

Figure 5: Temporal Evolution of Complexity (WMC) vs. Commit Count.

Cohesion and Geographic Dispersion:
Figure 6 examines the relationship between Lack of Cohesion in Methods (`LCOM_mean`) and (`TimezoneCount_mean`) geographic dispersion. A moderate positive correlation ($r = 0.56$) indicates that as teams become more geographically dispersed, code cohesion tends to decline—supporting the emergence of *Organizational Silo*, possibly due to asynchronous communication and scheduling challenges. However, we identified subclusters where teams maintain relatively high cohesion despite significant dispersion, suggesting that effective communication strategies—such as asynchronous collaboration tools, structured workflows, and well-defined coordination practices—can help mitigate these risks. This finding underscores the importance of proactive interventions to preserve software maintainability. Future research could investigate these high-performing subgroups in more detail to identify best practices for balancing geographic diversity with sustained code quality. Additionally, organizations and open-source communities may benefit from implementing communication protocols and decision-making frameworks to reduce the negative impact of distributed development on software cohesion.

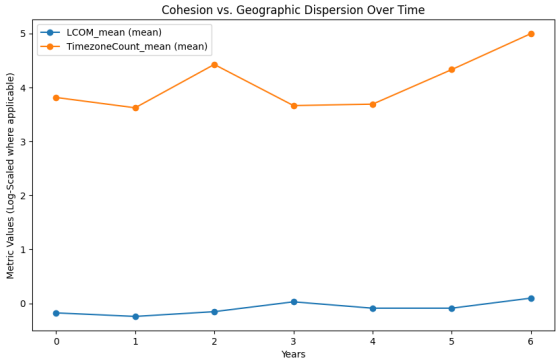

Figure 6: Temporal Evolution of Cohesion (LCOM) vs. Geographic Dispersion (TimezoneCount).

Team Diversity and Code Size:
Figure 7 presents the log-scaled relationship between Lines of Code (LOC_mean) and Author Count (AuthorCount_mean). A strong positive correlation ($r = 0.65$) indicates that as the number of contributors increases, the overall codebase grows. Although larger codebases generally imply higher complexity, our analysis suggests that effective collaborative practices—such as structured reviews and frequent refactoring—can help control this growth and mitigate smells like *Feature Envy*. By removing outliers and focusing on typical trends, we observe that teams adopting disciplined development processes tend to keep complexity in check even as the codebase expands.

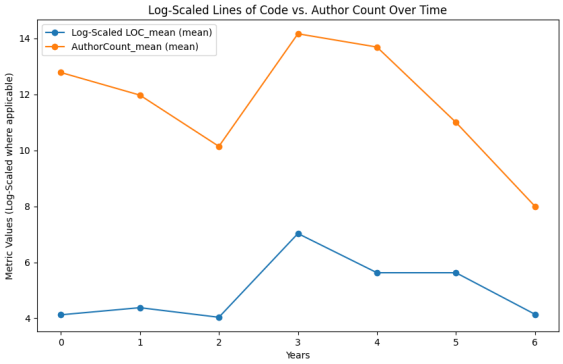

Figure 7: Log-Scaled Analysis of Lines of Code (LOC) vs. Author Count.

4.2 Correlation Analysis

The Pearson correlation matrix shown in Figure 8 reveals important interdependencies among the metrics. A near-perfect correlation ($r = 0.99$) between LOC and WMC_mean confirms that an increase in code size is strongly associated with higher complexity—a characteristic of the *God Class* smell. Additionally, the moderate positive correlation ($r = 0.65$) between AuthorCount_mean and LOC_mean indicates that although diverse teams produce larger codebases, effective collaborative practices are crucial for controlling complexity. The rolling correlation analysis (Figure 9) shows that the relationship between LOC and AuthorCount_mean fluctuates over time, highlighting how changes in team composition can significantly affect code growth. This variability reinforces the need for adaptive management—particularly during periods of rapid evolution—to maintain software quality.

4.3 Clustering Analysis of Smell Co-Occurrence

K-means clustering was applied to the normalized dataset to uncover latent patterns in smell co-occurrence. Data were normalized using Min-Max scaling, and outliers (values beyond ±2 standard deviations) were removed. The elbow method indicated that three clusters were optimal [31].

Figure 10 shows that higher geographic dispersion generally correlates with lower cohesion (LCOM_mean), suggesting an elevated risk of *Organizational Silo*. Notably, subclusters reveal that

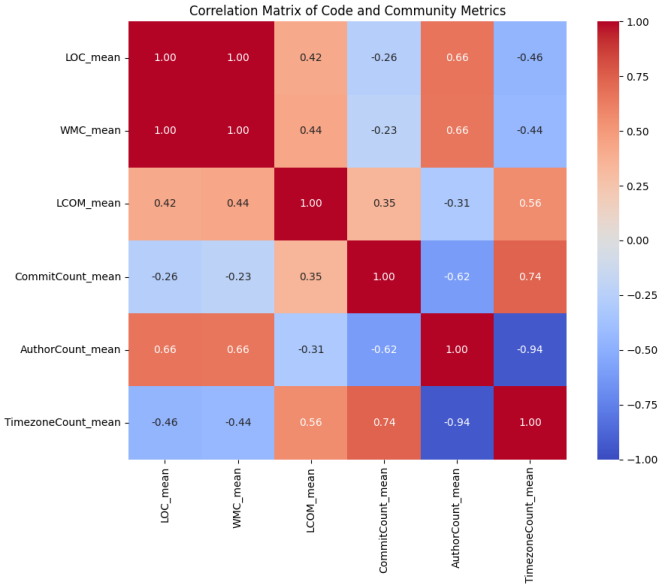

Figure 8: Correlation Matrix of Key Metrics for Code and Community Smells.

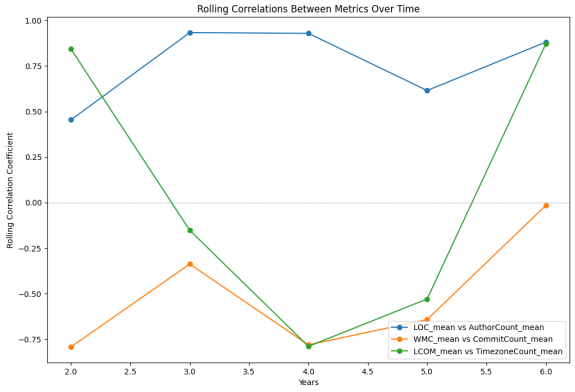

Figure 9: Rolling Correlations Between Code and Community Smells Over Time.

some teams, despite high dispersion, achieve relatively high cohesion—likely due to effective asynchronous communication protocols. This finding not only supports the trends observed in the temporal analysis (Section 4) but also highlights the heterogeneity among projects. In particular, while the overall trend indicates that increased dispersion tends to lower cohesion, a subgroup of projects manages to maintain high cohesion, suggesting that robust communication practices can mitigate the negative effects of geographic dispersion.

Similarly, Figure 11 depicts the clustering of code size (LOC) versus team diversity (AuthorCount_mean). Although larger teams typically produce more extensive codebases, the clustering analysis

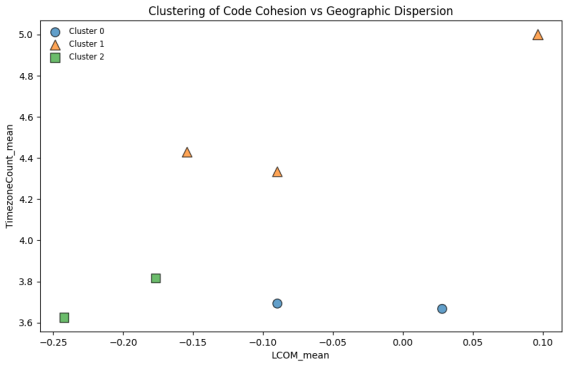

Figure 10: Clustering of Code Cohesion (`LCOM`) vs. Geographic Dispersion (`TimezoneCount`).

reveals that projects with higher contributor diversity often manage code growth more effectively. This parallels the findings for dispersion-cohesion, reinforcing the notion that structured collaboration strategies are critical for balancing technical complexity with social dynamics. In both cases, the clustering analysis uncovers subgroups of projects that deviate from the overall trend, indicating that effective coordination is a key mediator in controlling the negative impacts of both high geographic dispersion and large team sizes.

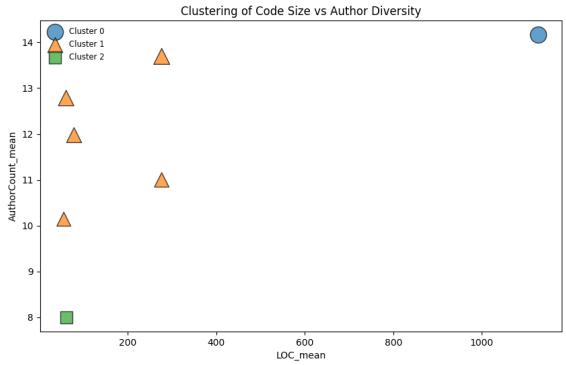

Figure 11: Clustering of Code Size (`LOC`) vs. Author Diversity (`AuthorCount`).

Our results illustrate that technical and social factors evolve dynamically and interdependently in open-source code samples. Although overall commit activity does not uniformly lead to increased complexity, local surges can trigger the emergence of complex code structures (e.g., *God Class*). Similarly, while geographic dispersion tends to reduce code cohesion—supporting the *Organizational Silo* phenomenon—some teams overcome these challenges through well-defined communication protocols. Furthermore, the analysis linking team diversity and code size indicates that, although larger teams tend to generate more extensive codebases, disciplined collaborative practices can control this growth and mitigate adverse effects. The parallel findings between the impacts of geographic dispersion and team diversity underscore the interdependence between social communication structures and code maintenance. This convergence suggests that interventions aimed at improving asynchronous collaboration can simultaneously reduce technical debt and enhance team cohesion.

The results not only confirm the mutual reinforcement of Code and Community Smells but also highlight critical intervention points. The negative impact of geographic dispersion on cohesion, for instance, suggests that even in highly distributed teams, the adoption of asynchronous communication tools and structured collaboration protocols can mitigate the adverse effects. Furthermore, the observed fluctuations in commit activity reinforce the need for adaptive management strategies that address transient spikes in complexity. These insights pave the way for developing predictive models and targeted refactoring strategies in large-scale open-source projects.

5 Threats to Validity

Internal Validity:
Our study employed Java-specific analysis tools (csDetector and Designite), which may introduce measurement biases due to tool limitations and calibration issues. To mitigate these risks, we selected well-established metrics (e.g., LOC, WMC, LCOM) validated in the literature [2, 12], applied rigorous data normalization procedures, and removed outliers (values beyond ±2 standard deviations) to ensure that our statistical analysis reflected typical trends rather than extreme cases [35]. Additionally, we conducted manual validation on a representative sample of repositories to confirm the reliability of the automated smell detection [30]. These measures help ensure that our results accurately reflect the underlying phenomena despite the inherent subjectivity in interpreting software quality issues.

External Validity:
Although our findings are derived from open-source Java repositories—which provide a controlled environment for analysis—they may not fully generalize to closed-source projects or ecosystems based on other programming languages. The specific characteristics of the Java ecosystem—such as prevalent design patterns and tooling constraints (e.g., SonarQube's LOC limit[4])—may affect the observed relationships. Future work should apply our research method to other software environments to assess the broader applicability of our results.

Construct Validity:
The selected metrics capture key dimensions of both Code Smells and Community Smells; however, they may not encompass all aspects of software quality. While metrics such as LOC, WMC, and LCOM provide quantitative measures of structural complexity and cohesion, they may not capture contextual or qualitative nuances. Integrating our quantitative approach with qualitative assessments in future studies could yield a more comprehensive understanding. Moreover, the mapping of specific metrics to the corresponding smells is based on established literature, although alternative interpretations may exist, warranting further investigation.

[4]https://docs.sonarsource.com/sonarqube-server/9.9/instance-administration/lines-of-code

6 Related Work

Research at the intersection of code smells and community smells has revealed significant insights into how organizational and social factors directly influence code quality in open-source projects. In this section, we organize key contributions in the literature into distinct themes.

Bedoya et al. [13] investigated the impact of code samples on developers' learning processes, emphasizing that simplicity and clarity are critical for framework adoption, especially in Java ecosystems. Their findings underscore the importance of maintaining up-to-date samples, a notion that aligns with our focus on the evolution of code samples and the associated smells.

Palomba et al. [1] examined the relationship between code smells and community smells, demonstrating that communication breakdowns and low team cohesion can drive the emergence of technical smells. Analyzing 117 versions across nine projects, they revealed that persistent code issues are often rooted in community dysfunctions. Our study extends this work by incorporating a temporal perspective over a larger dataset of 170 repositories to capture dynamic interactions between code and community smells.

Menezes et al. [22] focused on the evolution of code samples in frameworks such as Android and Spring Boot, highlighting that the need for compatibility drives ongoing maintenance challenges. Their work reinforces our investigation into the temporal evolution of both code and community smells.

Neugebauer et al. [24] concentrated on the impact of community smells on code refactoring by proposing automated smell detection methods. Although their approach emphasizes automation, our research differs by focusing on the temporal co-evolution of code and community smells, thereby addressing an important gap in the literature.

Unlike previous studies that primarily examined either code or community aspects in isolation, our work integrates both perspectives through a temporal lens. This comprehensive approach provides a deeper understanding of the interdependencies between technical and social factors in the evolution of open-source projects.

7 Conclusion and Future Work

This study provides a comprehensive and reproducible analysis of the co-occurrence and temporal evolution of Code Smells and Community Smells across 170 open-source Java repositories. By quantitatively examining standardized metrics and employing automated detection methods, rigorous normalization (including the use of annual "project years" to align varied release dates), and both correlation and clustering analyses, we have detailed how these smells emerge, evolve, and mutually reinforce each other, ultimately affecting software quality and team dynamics.

Our results indicate that, in the absence of continuous maintenance, both Code Smells (e.g., *Long Method*) and Community Smells (e.g., *Organizational Silo*) tend to accumulate over time, leading to a measurable decline in code quality. In particular, the analysis shows that localized surges in commit activity are associated with transient increases in complexity, supporting the hypothesis that technical deficiencies and social challenges can reinforce one another. Moreover, the statistically significant correlations between complexity metrics (such as WMC) and the prevalence of smells, combined with the clustering analysis, suggest that projects characterized by a balanced number of contributors and commit patterns exhibit a lower degree of adverse effects.

The implications of these findings are twofold. First, they underline the necessity of regular maintenance activities—such as systematic refactoring, structured code reviews, and the implementation of adaptive communication protocols—in order to prevent the gradual accumulation of both Code and Community Smells. Second, the methodological framework developed in this study serves as a reproducible basis for tracking the evolution of these phenomena, thereby offering a practical tool for project managers and practitioners to monitor and intervene in a timely manner.

Future Work. Several avenues for future research are apparent from the present study:

- **Refinement of Predictive Models:** Future work should focus on enhancing the predictive capacity of the current framework. This could involve the integration of additional quantitative metrics or alternative normalization techniques to more precisely forecast the emergence and progression of both Code and Community Smells.
- **Extension to Other Smell Types:** The current study is limited to a defined set of smells. Further research could expand the taxonomy to include other technical and social smells, thus providing a more granular understanding of the factors that influence software quality.
- **Cross-Ecosystem Validation:** Applying the research method to repositories in different programming languages and across varied software ecosystems will be crucial for assessing the generalizability of the findings. Comparative analyses in different contexts may also reveal environment-specific dynamics.
- **Integration of Qualitative Evaluations:** While the present study is grounded in quantitative metrics, incorporating qualitative methods—such as developer interviews or structured surveys—could help elucidate the contextual factors behind smell evolution and offer a more complete picture of the underlying causes.
- **Controlled Experiments on Intervention Strategies:** Future studies might design controlled experiments or case studies to evaluate the effectiveness of targeted interventions (e.g., periodic refactoring, enhanced communication protocols) in reducing the accumulation of smells over time.

In summary, this study not only advances our understanding of the temporal dynamics between Code Smells and Community Smells but also establishes a robust framework for ongoing monitoring and intervention. The proposed directions for future work aim to refine this framework and expand its applicability, thereby contributing to improved software quality and sustained team collaboration in open-source projects.

Acknowledgments

This work was funded by the Federal University of Mato Grosso do Sul and by Brazilian funding agencies CAPES (Finance Code 001) and FAEPEX (grants: 3404/23 and 2382/24).

References

[1] 2018. *How do community smells influence code smells?* Association for Computing Machinery, New York, NY, USA.

[2] Ashraf Abdou and Nagy Darwish. 2022. Severity classification of software code smells using machine learning techniques: A comparative study. *Journal of Software: Evolution and Process* 36, 1 (2022), e2454.

[3] Nuri Almarim, Ali Ouni, and Mohamed Wiem Mkaouer. 2020. Learning to Detect Community Smells in Open Source Software Projects.

[4] 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.

[5] Abdullah Almogahed, Hairulnizam Mahdin, Mazidah Mat Rejab, Abdulwadood Alawadhi, Samera Obaid Barraood, Manal Othman, Omar Al-Jamili, Abdulwahab Ali Almazroi, and Shazlyn Milleana Shaharudin. 2024. Code Refactoring for Software Reusability: An Experimental Study. In *2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*. 1–6.

[6] Giusy Annunziata, Carmine Ferrara, Stefano Lambiase, Fabio Palomba, Gemma Catolino, Filomena Ferrucci, and Andrea De Lucia. 2024. An Empirical Study on the Relation Between Programming Languages and the Emergence of Community Smells. In *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society, Los Alamitos, CA, USA, 268–275.

[7] Gemma Catolino, Fabio Palomba, Damian A. Tamburri, and Alexander Serebrenik. 2021. Understanding Community Smells Variability: A Statistical Approach. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. 77–86.

[8] Zhifei Chen, Wanwangying Ma, Lin Chen, and Wei Song. 2022. Collaboration in software ecosystems: A study of work groups in open environment. *Information and Software Technology* 145 (2022), 106849.

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

[10] Serge Demeyer and Tom Mens. 2008. *Software Evolution*. Springer, Boston.

[11] Rodrigo dos Santos, George Valença, Davi Viana, Bernardo Estácio, Awdren Fontão, Sabrina Marczak, Claudia Werner, Carina Alves, Tayana Conte, and Rafael Prikladnicki. 2014. Qualidade em Ecossistemas de Software: Desafios e Oportunidades de Pesquisa.

[12] Martin Fowler and Kent Beck. 2018. *Refactoring: Improving the Design of Existing Code*. Vol. 1. Addison-Wesley Professional, Boston, 464. The guide to how to transform code with a safe and rapid process, vital to keeping it cheap and easy to modify for future needs.

[13] Oscar Franco-Bedoya, David Ameller, Dolors Costal, and Xavier Franch. 2017. Open Source Software Ecosystems: A Systematic Mapping. *Information and Software Technology* (2017).

[14] Hadi Hemmati, Sarah Nadi, Olga Baysal, Oleksii Kononenko, Wei Wang, Reid Holmes, and Michael W. Godfrey. 2013. The msr cookbook: Mining a decade of research. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 343–352.

[15] Zi-Jie Huang, Zhi-Qing Shao, Gui-Sheng Fan, Hui-Qun Yu, Xing-Guang Yang, and Kang Yang. 2022. Community Smell Occurrence Prediction on Multi-Granularity by Developer-Oriented Features and Process Metrics. *Journal of Computer Science and Technology* 37, 1 (2022), 182–206.

[16] N.A.A. Khleel and K. Nehéz. 2024. Improving accuracy of code smells detection using machine learning with data balancing techniques. *Journal of Supercomputing* 80 (2024), 21048–21093.

[17] Noah Lambaria and Tom Černý. 2022. A Data Analysis Study of Code Smells within Java Repositories. In *Communication Papers of the 17th Conference on Computer Science and Intelligence Systems*, Vol. 32. ACSIS, 313–318.

[18] Riasat Mahbub, Mohammad Masudur Rahman, and Muhammad Ahsanul Habib. 2024. On the Prevalence, Evolution, and Impact of Code Smells in Simulation Modelling Software. In *2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*. 154–165.

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

[20] Júlio Martins, Carla Bezerra, Anderson Uchôa, and Alessandro Garcia. 2020. Are Code Smell Co-occurrences Harmful to Internal Quality Attributes? A Mixed-Method Study. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*. Association for Computing Machinery, New York, NY, USA, 52–61.

[21] Gabriel Menezes, Willian Braga, Awdren Fontão, Andre Hora, and Bruno Cafeo. 2022. Assessing the Impact of Code Samples Evolution on Developers' Questions. In *36th Brazilian Symposium on Software Engineering (SBES 2022)*. 1–10.

[22] Gabriel Menezes, Bruno Cafeo, and Andre Hora. 2019. Framework Code Samples: How Are They Maintained and Used by Developers?. In *IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 564–574.

[23] Behnaz Moradi-Jamei, Brandon L. Kramer, J. Bayoán Santiago Calderón, and Gizem Korkmaz. 2021. Community Formation and Detection on GitHub Collaboration Networks. In *2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 244–252.

[24] Tomasz Neugebauer, Pierre Lasou, and Pamela Carson. 2024. What are the characteristic community smells influencing the sustainability of open-source repository software communities? Open Repositories 2024 (OR2024).

[25] Fabio Palomba, Annibale Panichella, Andrea De Lucia, Rocco Oliveto, and Andy Zaidman. 2016. A textual-based technique for Smell Detection. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 1–10. https://doi.org/10.1109/ICPC.2016.7503704

[26] Antonio Della Porta, Stefano Lambiase, Gemma Catolino, Filomena Ferrucci, and Fabio Palomba. 2025. A Novel, Tool-Supported Catalog of Community Smell Symptoms. In *Proceedings of the Brazilian Symposium on Information Systems (SBSI25)*. Recife, PE, Brazil.

[27] Bárbara Beato Ribeiro, Luiz Alexandre Costa, Juliana Carvalho Silva Do Out ao, and Rodrigo Pereira Dos Santos. 2023. Towards Power Relationship Dynamics and Community Smells in the Proprietary Software Ecosystem. Association for Computing Machinery, New York, NY, USA, 142–147.

[28] Tushar Sharma. 2024. Multi-faceted Code Smell Detection at Scale using DesigniteJava 2.0. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. 284–288.

[29] Thiciane Suely Couto Silva, Fabio Gomes Rocha, and Rodrigo Pereira dos Santos. 2019. Resource Demand Management in Java Ecosystem. In *Proceedings of the XV Brazilian Symposium on Information Systems*. Association for Computing Machinery, New York, NY, USA, Article 3.

[30] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. 2014. The (R) Evolution of social media in software engineering. In *Future of Software Engineering Proceedings*. Association for Computing Machinery, New York, NY, USA, 100–116.

[31] D. A. Tamburri, P. Lago, and H. V. Vliet. 2013. Organizational social structures for software engineering. *ACM Comput. Surv.* 46, 1 (jul 2013), 3:1–3:35.

[32] 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.

[33] 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.

[34] Liang Wang, Ying Li, Jierui Zhang, and Xianping Tao. 2022. Quantitative Analysis of Community Evolution in Developer Social Networks Around Open Source Software Projects. arXiv:2205.09935 [cs.SE]

[35] Zeyi Wang, Eric Bridgeford, Shangsi Wang, Joshua T. Vogelstein, and Brian Caffo. 2024. Statistical Analysis of Data Repeatability Measures. (2024). arXiv:2005.11911

[36] Quanxin Yang, Dongjin Yu, Xin Chen, Yihang Xu, Wangliang Yan, and Bin Hu. 2024. Feature envy detection based on cross-graph local semantics matching. *Information and Software Technology* 174 (2024), 107515.

Received 18 November 2024; revised 04 February 2025; accepted 18 February 2025