Measuring the Lifecycle of Web GUI Test Smells

A Repository-Mining Study of End-to-End Test Smell Evolution

Vincenzo Di Nardo[1], Luigi Libero Lucio Starace[1,*]

[1]Software Quality and Intelligent Data-driven Systems (SQUIDS) Lab, Department of Electrical Engineering and Information Technology, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125, Napoli, Italy

Abstract

End-to-end (E2E) Web GUI tests are a critical quality gate for modern web applications, but their own maintainability is rarely measured longitudinally. Existing work has characterized the adoption and maintenance of Web GUI testing frameworks, while evidence on the lifecycle of quality problems inside E2E tests remains limited.

This paper frames Web GUI test smells as measurable maintainability-risk events and studies their incidence, evolution, and socio-technical context. We mine 358 open-source repositories from the E2EGit dataset, covering 3,435 smell-related test files, 29,830 commits, and 567,265 detected smell instances in JavaScript and TypeScript projects using Cypress, Playwright, Puppeteer, and Selenium. We operationalize five families of measures: smell incidence, file-level diffusion, transition type, distance from release dates, and developer ownership. The results show that Global Variable is the most frequent smell overall, with 149,673 instances, while Assertion Roulette is the most widespread, affecting 2,560 distinct files. Smells tend to accumulate more often than they are removed: worsening transitions outnumber improving transitions in both TypeScript and JavaScript histories. Smell-related changes are also release-centered, with 76.4% of TypeScript smell introductions and 62.9% of JavaScript smell introductions occurring within seven days before or after a release. Finally, file owners and core maintainers are deeply involved in both smell introduction and removal, and commit messages suggest that Sleepy Test, Redundant Print, and Unknown Test may reflect short-term maintenance trade-offs. These findings indicate that E2E test smells should be treated not only as static code-quality issues, but as measurable lifecycle phenomena connected to release pressure and maintenance behavior.

Keywords

Software measurement, End-to-end testing, Web GUI testing, Test smells, Mining software repositories, Test maintenance

1. Introduction

Automated end-to-end (E2E) testing validates web applications through realistic user-facing workflows. In contemporary JavaScript and TypeScript projects, frameworks such as Cypress, Playwright, Puppeteer, and Selenium are commonly used to exercise the browser interface and to protect release pipelines from regressions [1, 2]. As these suites grow, however, they also become software artifacts that must be maintained. Fragile selectors, arbitrary waits, global state, duplicated assertions, and missing or unclear oracles can make E2E suites slower to evolve and less trustworthy.

These problems are often discussed as test smells: recurring symptoms of poor test design or implementation that can increase maintenance effort without necessarily causing immediate failures [3, 4]. Prior work has studied smell introduction in production code [5], test smells in traditional test suites [3, 4], Web Graphical User Interface (GUI) testing practices [1, 2], and bad practices in adjacent testing domains such as performance testing [6]. This work shows that test-code quality is a recurring software-engineering concern, but it does not yet explain how smells inside Web GUI E2E tests evolve over repository history: when they appear, whether they are removed or accumulated, how close they are to releases, and which developers participate in their lifecycle.

This paper frames the study as a software-measurement contribution. Rather than treating smells only as detector outputs at one point in time, we define a set of lifecycle measures that connect smell

MENSURA 2026: International Conference on Software Measurement, 2026

*Corresponding author.

vincen.dinardo@studenti.unina.it (V. Di Nardo); luigiliberolucio.starace@unina.it (L. L. L. Starace)

https://luistar.github.io/ (L. L. L. Starace)

0000-0001-7945-9014 (L. L. L. Starace)

instances to commits, releases, files, and developers. This view supports a more operational question for test managers and researchers: not only "which smells exist?", but "how do test-quality issues move through the lifecycle of an E2E suite?"

The paper makes three contributions:

- A measurement model for historical Web GUI test smell analysis, covering incidence, diffusion, transition type, release distance, and developer involvement;
- An empirical study of 358 repositories, 3,435 smell-related E2E test files, and 29,830 commits from JavaScript and TypeScript projects;
- Evidence that E2E test smell accumulation is release-centered and socio-technical, motivating future work on release-aware smell monitoring, developer-facing feedback, and links between E2E test smells, flakiness, and Continuous Integration (CI) outcomes.

The remainder of this paper is organized as follows. Section 2 reviews prior work on E2E Web GUI testing, test smells, bad practices, and software evolution, and derives the measurement goal and research questions. Section 3 describes the dataset, mining pipeline, and operational measures. Section 4 reports the empirical results for smell incidence, release-centered evolution, contributor involvement, and commit-message signals. Section 5 discusses the implications of the findings for measuring and monitoring E2E test smells. Section 6 presents threats to validity, and Section 7 concludes the paper.

2. State of the Art and Measurement Goal

2.1. E2E and Web GUI Testing

E2E Web GUI tests simulate user actions through a browser and assert the resulting application behavior. They are valuable because they validate interactions across the client, backend, and supporting services, but this breadth also makes them sensitive to asynchronous behavior, timing assumptions, UI structure, and environment configuration [2]. Research on GUI testing has long studied how to exercise user interfaces effectively, including capture-and-replay and automatic input generation for mobile applications [7, 8], model-inference support for web application testing [9], and representation-learning techniques for web-testing tasks [10].

More recent work has focused directly on modern Web GUI E2E ecosystems. The E2EGit dataset provides a large-scale basis for studying Selenium, Cypress, Playwright, and Puppeteer tests in open-source repositories [1]. Follow-up studies have characterized the adoption and maintenance of these frameworks [2] and investigated the prediction of fragility in E2E web tests [11]. A related line of work uses E2E GUI-level scripts as operational knowledge for performance testing: E2E-Loader and its extensions semi-automatically derive web-app performance tests from E2E GUI tests [12, 13, 14], while studies of open-source performance testing analyze adoption, maintenance, and change taxonomies [15]. These works establish that E2E tests are not only regression guards; they are reusable engineering assets that influence other quality-assurance activities.

Beyond E2E-specific artifacts, related testing work has studied regression-test prioritization with code churn, change awareness, source-code similarity, and curated datasets [16, 17, 18, 19]. This broader line of work motivates treating test suites as evolving assets whose maintenance can be measured historically.

2.2. Test Smells, Bad Practices, and Software Evolution

The concept of test smells was originally formalized to describe maintainability and reliability risks in test code. Empirical studies have examined the nature of test smells [3], their harmfulness during evolution [20], and their detectability, validity, and reliability two decades later [4]. Broader surveys also show that test-code smells form a diverse knowledge base shared by researchers and practitioners [21]. In the Web GUI domain, Fulcini et al. introduced guidelines and a linter for detecting GUI-testing smells in Java and JavaScript suites [22]. In parallel, software-quality measurement work has studied code

Table 1
Web GUI E2E test smell categories considered in this study, adapted from [22].

Smell	Operational meaning in Web GUI E2E tests
Absolute URL	A test opens or asserts against a hard-coded complete URL instead of relying on a configurable base URL or relative route.
Assertion Roulette	A test contains multiple assertions whose failure context is difficult to identify, for example because assertions are not clearly separated or explained.
Complex Test	A test contains a long or multi-purpose workflow that combines several actions, scenarios, or responsibilities.
Conditional Logic	A test contains branches or condition-dependent behavior that make the expected execution path less explicit.
Constructor Initialization	Test state is initialized in constructors or object initialization logic rather than in explicit test fixtures or lifecycle hooks.
Duplicate Assert	The same condition or value is checked repeatedly without adding distinct diagnostic information.
Empty Test	A declared test case contains no meaningful executable behavior or oracle.
Exception Handling	Test success or failure depends on custom exception-handling logic instead of explicit framework assertions.
Global Variable	A test relies on shared mutable variables or state that can create dependencies among tests.
Magic Number	A test contains unexplained numeric literals in actions, waits, or assertions.
Mystery Guest	A test depends on external or implicit resources, such as files, databases, services, or pre-existing application state, that are not made explicit in the test setup.
Redundant Assertion	A test contains an assertion that is tautological, always true, or otherwise does not strengthen the oracle.
Redundant Print	A test contains logging or print statements left from debugging activities.
Sensitive Equality	A test compares values through brittle string representations or overly strict equality checks that can fail after irrelevant UI or formatting changes.
Sleepy Test	A test uses fixed waiting times instead of condition-based synchronization.
Unknown Test	A test lacks a clear oracle, making its expected behavior or pass condition difficult to infer.

smells and architectural anti-patterns in educational and student-built systems [23, 24, 25], reinforcing the role of measurable quality signals as feedback for developers.

Historical smell analysis has mostly targeted production code and general technical debt. Tufano et al. studied when and why code smells are introduced [5], Peters and Zaidman measured the lifespan of code smells through repository mining [26], and Digkas et al. analyzed the evolution of technical debt across software releases [27]. More recently, research has extended the bad-practice perspective to performance testing, including open-source performance-test maintenance [15] and taxonomies of performance-testing bad practices grounded in gray literature and practitioner validation [6]. This evidence is close to our problem because it treats tests as maintainable software artifacts, but it focuses on performance testing rather than browser-level E2E GUI tests.

Taken together, prior studies show that GUI tests, E2E tests, test smells, and test-related bad practices have all been investigated. The missing state-of-the-art link is their lifecycle intersection: we still lack a measurement-oriented view of how Web GUI E2E test smells appear, persist, worsen, or disappear over time, and how those events relate to releases and developers.

2.3. Measurement Goal

This paper addresses this gap by measuring Web GUI test smells as historical maintainability-risk events. We adopt a catalog of Web GUI E2E test smell categories adapted from the GUI-testing guidelines proposed by Fulcini et al. [22] and aligned with established test-smell terminology. Table 1 reports the operational meaning assigned to the smell categories considered in the empirical analysis. The table is intended to make the measurement vocabulary explicit: the smells are used as measurable

maintainability-risk indicators, not as a new taxonomy of GUI test smells.

The measurement goal is to characterize the incidence, diffusion, lifecycle transitions, release proximity, and contributor context of Web GUI E2E test smells in JavaScript and TypeScript repositories. This goal complements richer software-engineering text-mining approaches, such as LLM-based issue-report classification [28], with a deliberately traceable measurement model based on repository history, static detection, release tags, developer metadata, and lightweight commit-message signals.

2.4. Research Questions

The study is organized around four research questions:

- **RQ1, incidence.** Which E2E Web GUI test smells are most frequent and most widespread across languages and frameworks?
- **RQ2, release lifecycle.** How do smell transitions evolve over repository history and around software releases?
- **RQ3, contributors.** What role do file owners and core contributors play in smell introduction, worsening, and removal?
- **RQ4, intent signals.** Do commit messages provide signals that some smells are associated with conscious maintenance activities?

3. Method

The analysis pipeline combines static smell detection, repository mining, relational persistence, and statistical reporting.

3.1. Dataset and Pipeline

The study starts from repositories in E2EGit [1]. We implemented and integrated a dedicated static detector for JavaScript and TypeScript E2E test code. Although the smell catalog is conceptually based on Fulcini et al.'s GUI-testing guidelines [22], the detector used in this study is not the original linter applied as a black box. The original tooling [22] targeted a different implementation context, whereas our pipeline operationalizes the catalog for JavaScript/TypeScript projects and for the E2E frameworks represented in E2EGit, namely Selenium, Playwright, Puppeteer, and Cypress. The detector parses relevant test revisions, recognizes framework-specific idioms, and emits smell records containing the framework, repository, file path, method, line, and smell type. These records are then mapped to the smell categories in Table 1. Selected detector matches were manually inspected during implementation as a sanity check, but this inspection does not constitute an independently labeled benchmark of detector precision and recall.

The historical mining phase then reconstructs the commit history of the files in which smells are detected rather than traversing every file in every repository. This selective strategy keeps the analysis scalable while preserving the lifecycle of the artifacts in which Web GUI E2E test smells occur.

For each relevant file revision, the pipeline records repository and file identifiers, commit hash, author, date, message, detected smell type, method and line information where available, deletion status, and nearest previous and future release tags. PyDriller supports the repository traversal and commit metadata extraction [29]. SQLite stores raw and derived data, while Python and R scripts compute summaries and visualizations [30, 31].

3.2. Operational Measures

We define five families of measures.

Incidence measures count detected smell instances by smell category, language, framework, and repository. Since raw occurrence counts can be dominated by repeated smells within the same file, we also measure **file-level diffusion**: the number of distinct test files affected by each smell.

Table 2
Dataset and full historical transition counts.

Lang.	Repos.	Files	Commits	Smells	Intro.	Impr.	Wors.	Same
TypeScript	247	2,612	24,416	499,965	2,718	2,725	4,174	14,799
JavaScript	111	823	5,414	67,300	819	439	656	3,500
Total	358	3,435	29,830	567,265	3,537	3,164	4,830	18,299

Transition measures classify consecutive file revisions by comparing detected smell instances across commits. A smell instance is identified by its smell type and, when available, by method and line information. A transition is an *introduction* when a file moves from no detected smell to at least one detected smell, a *worsening* when the number of smell instances increases, an *improvement* when the number decreases, and *no-change* otherwise. This simplified transition model, adopted also in prior research [32], measures the direction of aggregate test-quality movement rather than judging developer intent.

Release-distance measures associate each smell-related commit with release tags by considering the nearest previous and future tags observed during repository mining. We compute whether transitions occur within fixed windows around the closest release, especially ±7, ±14, and ±30 days, and summarize the absolute distance from that closest release date.

Contributor measures classify file ownership by commit share. A developer is treated as a file owner when they authored more than 45% of commits touching that file. We also derive newcomer status when at least one of a contributor's first three commits in a repository belongs to the set of smell-related commits, following the intuition used in historical smell-introduction studies [5]; in this paper, the reported contributor evidence focuses on ownership and core-contributor involvement.

Intent-signal measures use keyword-based commit-message classification. Messages are scanned for terms associated with temporary workarounds, debugging, skipped tests, delays, or stabilization. These signals are not used as proof of causality; they provide evidence that some smell-related changes are connected to explicit maintenance activities.

4. Results

Table 2 summarizes the analyzed dataset and full historical transition counts. The TypeScript portion is larger, with 247 repositories, 2,612 smell-related files, 24,416 commits, and 499,965 smell instances. The JavaScript portion includes 111 repositories, 823 smell-related files, 5,414 commits, and 67,300 smell instances.

4.1. RQ1: Incidence and Diffusion

Using the smell categories defined in Table 1, we first compare incidence and diffusion across the complete dataset. Global Variable is the most frequent smell, with 149,673 detected instances. Assertion Roulette follows with 125,985 instances, and Mystery Guest with 86,158 instances. However, raw occurrence counts tell only part of the story. When measuring diffusion across distinct test files, Assertion Roulette becomes the most widespread smell, affecting 2,560 files, while Global Variable affects 2,063 files. This difference matters for measurement: Global Variable is more intense where it occurs, while Assertion Roulette is distributed across a broader part of the E2E test corpus.

The two languages exhibit different profiles. In TypeScript, the dominant smells are Global Variable, Assertion Roulette, and Mystery Guest. In JavaScript, Assertion Roulette is the most frequent smell, followed by Magic Number and Sleepy Test. The framework dimension also changes the distribution. Playwright and Cypress account for most detected instances, reflecting their large presence in the analyzed corpus. Puppeteer is strongly characterized by Unknown Test, which represents 38.8% of its

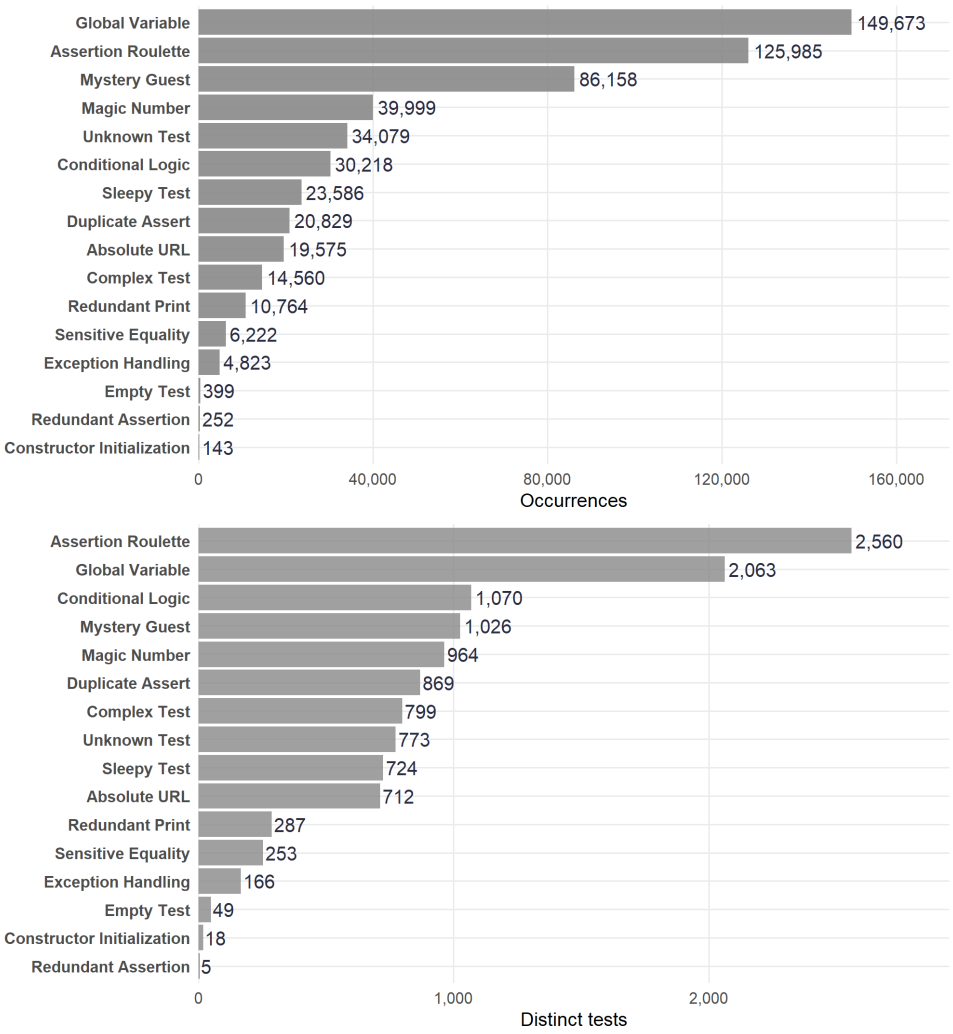

Figure 1: Smell incidence measured by total occurrences (top) and file-level diffusion (bottom).

smell occurrences, while Selenium is characterized by Redundant Print and Sleepy Test. Figure 2 shows that framework choice is not just a volume factor: different frameworks expose different smell profiles.

4.2. RQ2: Release-Centered Evolution

Smells are not only present, but evolve. Worsening transitions outnumber improvements in both ecosystems. In TypeScript, the pipeline identifies 4,174 worsening commits and 2,725 improving commits. In JavaScript, it identifies 656 worsening commits and 439 improving commits. This pattern suggests that smell-related maintainability risks are often accumulated or preserved rather than systematically removed.

The release-distance analysis shows that smell-related transitions are concentrated around release activity. During mining, each analyzed commit is associated with the closest Git tag by considering the nearest previous and future release tags. Table 3 reports the directional transitions used in the paper, namely introductions, improvements, and worsenings. In this dataset, all directional transitions in Table 2 could be assigned a closest release tag; no-change transitions were analyzed as context, as shown in Figure 3, but are omitted from Table 3 because they do not represent a directional change in smell intensity. In TypeScript, 76.4% of smell introductions, 77.2% of improvements, and 80.2% of worsenings occur within seven days before or after a release. In JavaScript, the same window contains 62.9% of introductions, 53.1% of improvements, and 61.4% of worsenings. The median absolute distance from a release is one day for TypeScript introductions, improvements, and worsenings; for JavaScript it

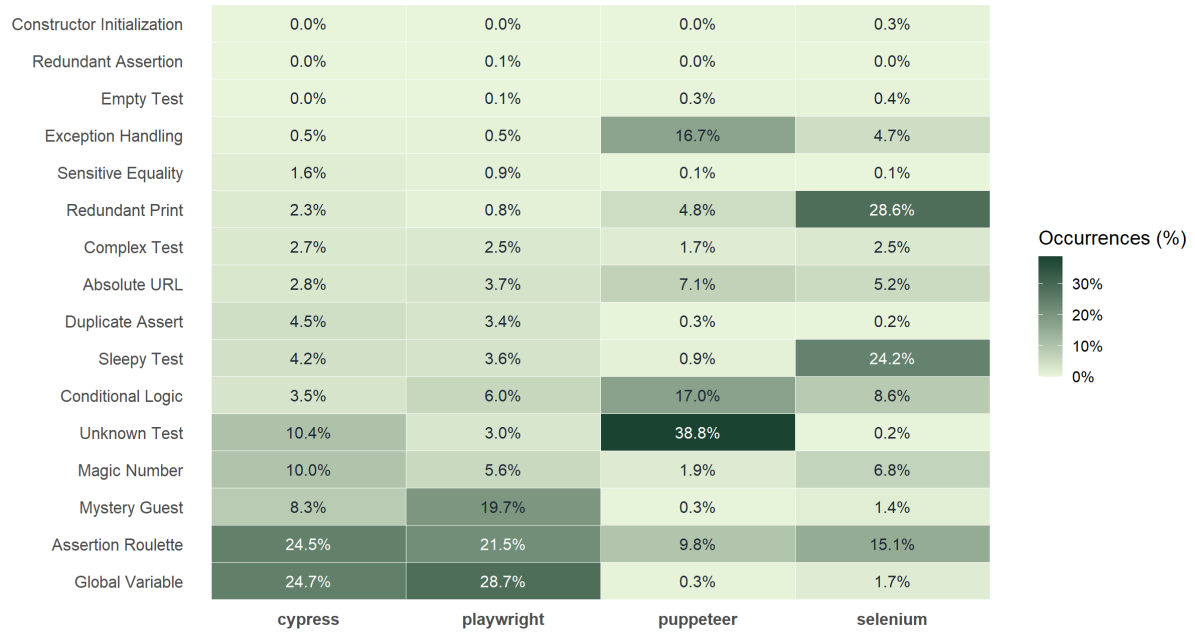

Figure 2: Distribution of smell categories across E2E testing frameworks.

Table 3
Release-distance measures for directional smell transitions with an associated closest release tag.

Lang.	Transition	Commits	±7d	±14d	Median $\lvert d \rvert$
TypeScript	Introduction	2,718	76.4%	84.3%	1
TypeScript	Improvement	2,725	77.2%	84.4%	1
TypeScript	Worsening	4,174	80.2%	87.9%	1
JavaScript	Introduction	819	62.9%	75.2%	4
JavaScript	Improvement	439	53.1%	70.4%	6
JavaScript	Worsening	656	61.4%	76.1%	4

is four, six, and four days, respectively.

Repository age provides another lifecycle signal. Many smell-related variations happen early: approximately 58% of JavaScript variations and 79% of TypeScript variations occur within the first week of repository activity as observed in Git. This suggests that E2E testing conventions are often established early and then persist through later maintenance, although initial imports or migrated histories can affect the apparent timing of early activity.

4.3. RQ3: Contributor Involvement

Ownership measures indicate that smells are not introduced only outside the core maintenance group. File owners introduce approximately 49% of JavaScript smell instances and 47% of TypeScript smell instances. Owners also contribute substantially to removals, accounting for approximately 32% of JavaScript improvement events and 26% of TypeScript improvement events. In other words, the same core contributors who shape test architecture also absorb much of the cleanup work.

This result changes the interpretation of smell prevention. The problem is not limited to onboarding or peripheral contributions. Smell measurement should also be integrated into the working practices of maintainers, especially around releases and initial project setup, when the study observes high smell activity.

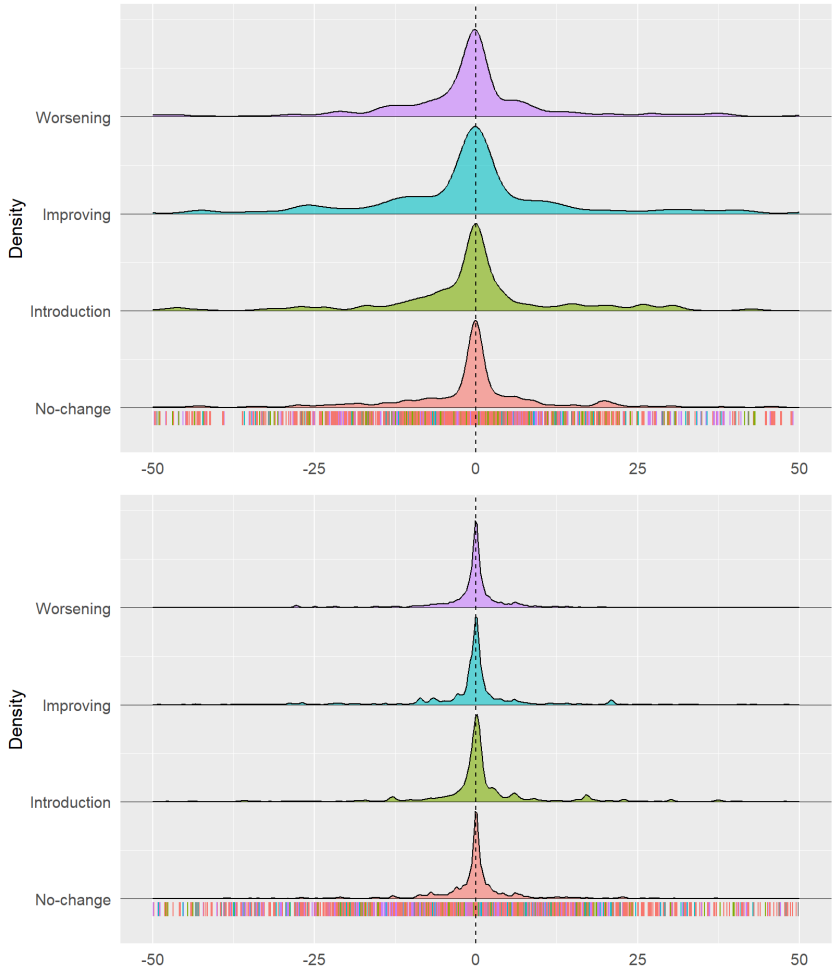

Figure 3: Distribution of smell-related commits relative to the nearest release for JavaScript (top) and TypeScript (bottom).

Table 4
Commit-message keyword matches and selected smell categories.

Lang.	Matches	Intro.	Wors.	Sleepy	Print	Unknown
TypeScript	2,404	296	440	476	182	562
JavaScript	492	99	48	84	60	52

4.4. RQ4: Commit-Message Signals

The commit-message analysis searches for terms associated with temporary solutions, debugging, skipped tests, timing workarounds, and stabilization. Table 4 summarizes the matched commits. In TypeScript, 2,404 commits match at least one keyword associated with smell-related maintenance activities; 440 of them are classified as worsening transitions and 296 as introductions. In JavaScript, 492 commits match the keyword filter; 48 are worsening transitions and 99 are introductions.

The most interpretable categories are Sleepy Test, Redundant Print, and Unknown Test. Sleepy Test often appears with messages mentioning delays, retries, or timeouts. Redundant Print aligns with debugging messages. Unknown Test is often connected to skipped or temporarily disabled tests. These associations suggest that some E2E smells may reflect short-term maintenance trade-offs: developers may introduce fixed waits, logging, or incomplete tests to stabilize or inspect a failing workflow, especially under maintenance pressure. The measurement implication is that smells should not be interpreted only as accidental quality degradation. Some may be visible traces of development decisions

that prioritize short-term test-suite operability over long-term maintainability.

5. Discussion

The results support three practical interpretations. First, E2E test smells should be monitored using both incidence and diffusion measures. Counting only occurrences emphasizes recurring local problems such as Global Variable, while distinct-file diffusion reveals broad risks such as Assertion Roulette. A measurement dashboard should include both views.

Second, release distance is a useful lifecycle measure. The concentration of transitions near releases suggests that E2E test smell evolution is tied to delivery rhythm. This makes release preparation a natural point for smell monitoring, just as teams already monitor build status, flaky tests, and regression failures.

Third, contributor-aware measures help avoid simplistic blame. Owners and experienced maintainers are central to smell evolution because they perform much of the work. Smell prevention therefore needs to be framed as maintainability support for core contributors, not only as onboarding control.

6. Threats to Validity

Construct validity depends on the smell catalog and detector accuracy. False positives, missed smells, and language-specific detector coverage can affect measured incidence. To mitigate underspecification, Table 1 reports the operational interpretation of each smell category used in the empirical analysis; nevertheless, the mapping from Fulcini et al.'s GUI-testing guidelines to JavaScript/TypeScript static checks remains an implementation choice and may affect comparability with other detectors. Some smell checks are language-sensitive and may have different precision or coverage in JavaScript and TypeScript. Therefore, cross-language comparisons should be interpreted as detector-supported measurement differences rather than direct evidence that one language intrinsically contains more of a given smell. The detector was sanity-checked through manual inspection of selected matches, but a full manually labeled benchmark is not available, so precision and recall remain a construct-validity threat.

Internal validity is limited by the transition model. The model is file-level and count-based: changes in smell counts are treated as introductions, improvements, or worsenings, but the model does not prove developer intent and does not distinguish all smell-type substitutions. A commit may remove one smell category and introduce another while producing a neutral or net-worsening count. Commit-message analysis is keyword-based and should be interpreted as suggestive evidence rather than a causal explanation. Repository-age results also depend on the Git timeline observed during mining; initial imports, migrated histories, or late adoption of E2E testing may shift the apparent age of smell-related activity. External validity is limited to open-source repositories represented in E2EGit and may not generalize to industrial E2E suites or proprietary release processes. Finally, release-distance measures rely on Git tags as release proxies, as done in prior works [32]. Projects that tag inconsistently may be under- or over-represented in release-centered analyses.

7. Conclusion

This paper presented a measurement-oriented historical study of Web GUI E2E test smells. Across 358 repositories and 29,830 commits, the results show that smell incidence, diffusion, lifecycle transitions, release distance, and contributor involvement provide complementary views of E2E test-smell evolution. Global Variable is the most frequent smell, Assertion Roulette is the most widespread, and worsening transitions outnumber improvements in both TypeScript and JavaScript histories. Smell-related changes concentrate around releases and early project evolution, while file owners and core maintainers are responsible for a substantial share of both smell introduction and removal. Commit-message evidence further suggests that some smells may be associated with short-term maintenance trade-offs: Sleepy

Test, Redundant Print, and Unknown Test often appear in commits related to delays, debugging, or disabled tests.

Future work will extend this measurement model with finer-grained statistical analysis, industrial validation, links to flaky-test and CI outcomes, and developer-facing interventions that warn teams when release pressure is likely to increase E2E test-smell risk.

Acknowledgments

This work was carried out at the Software Quality and Intelligent Data-driven Systems (SQUIDS) Lab, Università degli Studi di Napoli Federico II. We acknowledge the support of Mr. Daniyl Popov, who contributed to the test smell detector component during his curricular internship at the SQuIDS Lab. This work was partially supported by the Future Artificial Intelligence Research (FAIR) PNRR MUR project (PE0000013).

Declaration on Generative AI

During the preparation of this manuscript, the authors used OpenAI Codex to support text condensation and LaTeX editing. The authors reviewed and edited the content and take full responsibility for the publication's content.

References

[1] S. Di Meglio, L. L. L. Starace, V. Pontillo, R. Opdebeeck, C. De Roover, S. Di Martino, E2EGit: A dataset of end-to-end web tests in open source projects, in: Proceedings of the 22nd IEEE/ACM International Conference on Mining Software Repositories, IEEE/ACM, 2025, pp. 1–5.

[2] S. Di Meglio, L. L. L. Starace, V. Pontillo, R. Opdebeeck, C. De Roover, S. Di Martino, Investigating the adoption and maintenance of web GUI testing: Insights from GitHub repositories, Information and Software Technology 189 (2026) 1–18.

[3] M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, D. Poshyvanyk, An empirical investigation into the nature of test smells, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, 2016, pp. 4–15.

[4] A. Panichella, S. Panichella, G. Fraser, A. A. Sawant, V. J. Hellendoorn, Test smells 20 years later: Detectability, validity, and reliability, Empirical Software Engineering 27 (2022).

[5] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, D. Poshyvanyk, When and why your code starts to smell bad, in: Proceedings of the 37th IEEE/ACM International Conference on Software Engineering, IEEE/ACM, 2015, pp. 403–414.

[6] S. Di Meglio, L. L. L. Starace, V. Pontillo, L. Martins, D. Di Nucci, F. Palomba, A taxonomy of bad practices in software performance testing: Insights from gray literature and practitioner validation, ACM Transactions on Software Engineering and Methodology (2026). doi:10.1145/3821538.

[7] S. Di Martino, A. R. Fasolino, L. L. L. Starace, P. Tramontana, Comparing the effectiveness of capture and replay against automatic input generation for Android graphical user interface testing, Software Testing, Verification and Reliability 31 (2021) e1754. doi:10.1002/stvr.1754.

[8] S. Di Martino, A. R. Fasolino, L. L. L. Starace, P. Tramontana, GUI testing of Android applications: Investigating the impact of the number of testers on different exploratory testing strategies, Journal of Software: Evolution and Process 36 (2024) e2640. doi:10.1002/smr.2640.

[9] A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, Web application testing: Using tree kernels to detect near-duplicate states in automated model inference, in: Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM/IEEE, 2021.

[10] K. Kanaththage, L. L. L. Starace, M. Biagiola, P. Tonella, A. Stocco, Neural embeddings for web testing, in: Proceedings of the 19th IEEE International Conference on Software Testing, Verification and Validation, IEEE, 2026, p. 12 pages.

[11] S. Di Meglio, L. L. L. Starace, Towards predicting fragility in end-to-end web tests, in: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2024. doi:10.1145/3661167.3661179.

[12] E. Battista, S. Di Martino, S. Di Meglio, F. Scippacercola, L. L. L. Starace, E2E-loader: A framework to support performance testing of web applications, in: Proceedings of the 2023 IEEE Conference on Software Testing, Verification and Validation, IEEE, 2023. doi:10.1109/ICST57152.2023.00040.

[13] S. Di Meglio, L. L. L. Starace, S. Di Martino, E2E-loader: A tool to generate performance tests from end-to-end GUI-level tests, in: Proceedings of the 2025 IEEE Conference on Software Testing, Verification and Validation, IEEE, 2025. doi:10.1109/ICST62969.2025.10989035.

[14] S. Di Meglio, L. L. L. Starace, S. Di Martino, Semi-automated generation of web app performance tests from end-to-end GUI-level tests with E2E-loader, Science of Computer Programming (2026) 103461. doi:10.1016/j.scico.2026.103461.

[15] S. Di Meglio, L. L. L. Starace, V. Pontillo, R. Opdebeeck, C. De Roover, S. Di Martino, Performance testing in open-source web projects: Adoption, maintenance, and a change taxonomy, in: Proceedings of the 2025 IEEE International Conference on Software Maintenance and Evolution, IEEE, 2025. doi:10.1109/ICSME64153.2025.00027.

[16] F. Altiero, A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, Inspecting code churns to prioritize test cases, in: Testing Software and Systems, Springer, 2020, pp. 272–285. doi:10.1007/978-3-030-64881-7_17.

[17] F. Altiero, G. Colella, A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, Change-aware regression test prioritization using genetic algorithms, in: Proceedings of the 48th Euromicro Conference on Software Engineering and Advanced Applications, IEEE, 2022. doi:10.1109/SEAA56994.2022.00028.

[18] F. Altiero, A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, ReCover: A curated dataset for regression testing research, in: Proceedings of the 19th International Conference on Mining Software Repositories, IEEE/ACM, 2022.

[19] F. Altiero, A. Corazza, S. Di Martino, A. Peron, L. L. L. Starace, Regression test prioritization leveraging source code similarity with tree kernels, Journal of Software: Evolution and Process 36 (2024) e2653. doi:10.1002/smr.2653.

[20] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, D. Binkley, Are test smells really harmful? an empirical study, Empirical Software Engineering 20 (2014). doi:10.1007/s10664-014-9313-0.

[21] V. Garousi, B. Küçük, Smells in software test code: A survey of knowledge in industry and academia, Journal of Systems and Software 138 (2018) 30.

[22] T. Fulcini, G. Garaccione, R. Coppola, L. Ardito, M. Torchiano, Guidelines for gui testing maintenance: A linter for test smell detection, in: Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation, 2022, pp. 17–24.

[23] M. De Luca, S. Di Meglio, A. R. Fasolino, L. L. L. Starace, P. Tramontana, Automatic assessment of architectural anti-patterns and code smells in student software projects, in: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2024. doi:10.1145/3661167.3661290.

[24] M. De Luca, S. Di Martino, S. Di Meglio, A. R. Fasolino, L. L. L. Starace, P. Tramontana, Rookie mistakes: Measuring software quality in student projects to guide educational enhancement, in: Software Engineering and Advanced Applications, Springer, 2025, pp. 137–154. doi:10.1007/978-3-032-04207-1_10.

[25] S. Di Meglio, V. Pontillo, L. L. L. Starace, REST in pieces: RESTful design rule violations in student-built web apps, in: Software Engineering and Advanced Applications, Springer, 2025, pp. 191–200. doi:10.1007/978-3-032-04207-1_13.

[26] R. Peters, A. Zaidman, Evaluating the lifespan of code smells using software repository mining, in: Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, 2012.

[27] G. Digkas, M. Lungu, A. Chatzigeorgiou, P. Avgeriou, The evolution of technical debt in the apache ecosystem, in: Software Architecture, Springer International Publishing, 2017, pp. 51–66.

doi:10.1007/978-3-319-65831-5_4.

[28] G. De Vito, L. L. L. Starace, F. Palomba, S. Di Martino, F. Ferrucci, Advancing LLM-based issue report classification with explained few-shot learning, intent extraction, ensemble, and summarization, ACM Transactions on Software Engineering and Methodology (2026). doi:10.1145/3815577.

[29] D. Spadini, M. Aniche, A. Bacchelli, PyDriller: Python framework for mining software repositories, in: Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering: Companion Proceedings, ACM, 2018, pp. 908–911. doi:10.1145/3236024.3264598.

[30] T. E. Oliphant, Python for scientific computing, Computing in Science & Engineering 9 (2007) 10–20. doi:10.1109/MCSE.2007.58.

[31] R. Ihaka, R. Gentleman, R: A language for data analysis and graphics, Journal of Computational and Graphical Statistics 5 (1996) 299–314. doi:10.1080/10618600.1996.10474713.

[32] S. Di Meglio, V. Pontillo, L. L. L. Starace, L. Martins, D. Di Nucci, F. Palomba, Smelly bad practices in performance system tests: Detection, prevalence, and lifetime, in: Proceedings of the 2025 IEEE International Conference on Software Maintenance and Evolution, IEEE, 2025.