Exploring the Socio-Technical Motivations Behind the Adoption and Migration of Web GUI Testing Frameworks

Giuseppe Di Martino[1], Sergio Di Meglio[1], Valeria Pontillo[2], and Luigi Libero Lucio Starace[1]

[1] Software Quality and Intelligent Data-driven Systems (SQuIDS) Lab, University of Naples Federico II, Italy
[2] Gran Sasso Science Institute (GSSI), L'Aquila, Italy
emails: giuseppe.dimartino10@studenti.unina.it, {sergio.dimeglio, luigiliberolucio.starace}@unina.it, valeria.pontillo@gssi.it

Abstract. Web Graphical User Interface (GUI) testing frameworks are widely used to automate end-to-end testing of modern web applications. While prior research has explored their performance and technical features, limited attention has been directed to the socio-technical motivations that drive their adoption and migration in practice. To address this gap, this paper explores the socio-technical motivations behind the adoption and migration of web GUI testing frameworks through a repository-based study of open-source projects that use them. Specifically, we analyzed commit messages and issue discussions associated with web GUI testing framework adoption and migration events, and manually identified explicit rationale statements. This process yielded 72 relevant messages supporting 52 justified events, from which we inductively derived a thematic categorization of eight socio-technical motivations. Our results indicate that initial adoption is primarily shaped by usability and integration considerations, whereas migration is more strongly associated with reliability, performance, and ecosystem alignment, providing empirical evidence that the criteria guiding framework decisions evolve over the lifecycle of web GUI testing infrastructure.

Keywords: Web GUI testing, Trustworthy software engineering, Explainability, Testing infrastructure, Tool migration.

1 Introduction

Web Graphical User Interface (GUI) testing evaluates web application quality by executing and observing user-facing behavior through browser interactions [25, 5, 15]. By replicating realistic actions such as page navigation, link clicking, and form submission, it helps identify deviations from expected behavior and assess software quality [3]. As modern web applications become increasingly complex and interactive, automated web GUI testing has become a key component of

quality assurance. Consequently, the reliability of the underlying testing infrastructure is critical, as test outcomes must accurately reflect application defects rather than limitations of the testing tools themselves.

Automated web GUI testing relies on browser automation frameworks such as Selenium, Playwright, and Cypress [26]. These frameworks differ in architecture, capabilities, ecosystem support, and developer experience, and their selection can significantly influence the effectiveness, maintainability, and scalability of testing infrastructures. As a result, decisions concerning the adoption and evolution of testing frameworks can be viewed as socio-technical decisions that affect both software quality assurance and the long-term sustainability of testing processes. Although browser automation frameworks can be regarded as software engineering tools, their adoption and migration deserve dedicated investigation because they occupy a distinctive role in the development process. Unlike tools that primarily support implementation or analysis activities, web GUI testing frameworks generate the evidence used to assess externally visible application behavior. Consequently, limitations such as flaky executions, poor synchronization with dynamic interfaces, limited CI integration, or high maintenance costs can directly affect whether test failures are interpreted as real application defects or as artifacts of the testing infrastructure. Studying these frameworks, therefore, provides insight not only into tool adoption but also into how projects establish and revise the infrastructure on which trustworthy software quality assessment depends.

Prior research has extensively investigated browser automation frameworks from a technical perspective, comparing their features, performance, and limitations [33, 26]. A substantial body of work has also examined challenges associated with web GUI testing, including flakiness, fragility, and maintenance costs [32, 6, 14, 28, 8, 30]. However, despite this rich literature, limited attention has been devoted to the explicit rationales through which developers justify adopting or replacing browser automation frameworks, and to how these rationales evolve throughout the lifecycle of testing infrastructures.

Understanding these decisions is important because it helps explain how testing infrastructures evolve over time, what trade-offs practitioners face, and how technical, organizational, and human factors jointly shape framework selection. Such knowledge can support both practitioners evaluating testing technologies and researchers studying socio-technical decision-making in software engineering.

In this study, we investigate the motivations behind the adoption and migration of browser automation frameworks. Leveraging the E2EGit dataset [18, 20], we analyzed 6,748 commit messages and 54,327 issue messages through a combination of keyword-based filtering and manual qualitative analysis. This process yielded 72 rationale-bearing messages describing 52 adoption and migration events, comprising 25 adoptions and 27 migrations.

From these data, we inductively derived a thematic categorization of eight socio-technical motivations and compared their distribution across adoption and migration decisions. Our analysis reveals a systematic shift in priorities: adoption decisions are primarily driven by developer-oriented concerns such as usability

and ease of integration, whereas migration decisions are more frequently motivated by operational concerns including reliability, stability, and performance. This lifecycle shift, which we call *Priority Inversion*, suggests that factors supporting initial adoption differ from those required for long-term sustainability.

Our contributions are threefold. First, we provide empirical evidence on how developers explain and justify adoption and migration decisions involving web GUI testing frameworks. Second, we derive a thematic categorization of eight socio-technical motivations underlying these decisions. Third, we identify the *Priority Inversion* phenomenon, showing that the criteria driving framework adoption differ from those motivating migration and long-term use. Taken together, our findings support a lifecycle-aware view of framework selection, where early adoption decisions should be evaluated not only against immediate development needs but also against longer-term concerns such as reliability, maintainability, scalability, execution cost, and integration with CI/CD infrastructures.

2 Related Work

Our work relates to four research pillars: studies on software engineering tool adoption, technical comparisons of browser automation frameworks, research on the reliability and maintenance of web GUI tests, and empirical studies on the adoption and evolution of testing practices. The adoption of software engineering tools, including CASE tools, has been investigated extensively in prior work [10, 16]. This literature shows that tool adoption is influenced by factors such as perceived usefulness, ease of use, compatibility with existing processes, organizational support, training effort, and integration costs. These factors are not specific to web GUI testing, and we do not claim that the motivations observed in our study are entirely unique to browser automation frameworks. Instead, our study examines how such general adoption concerns are instantiated in a specific testing context, where framework decisions affect the reliability, maintainability, and interpretability of automated quality-assurance feedback.

Within web GUI testing, a substantial body of work has compared different browser automation frameworks from a technical perspective and focused on the key challenges of GUI-level web testing [21, 3, 26]. These studies help explain what these different web GUI testing frameworks offer and how they perform, but they provide limited evidence on why teams choose one framework over another or decide to migrate to different frameworks.

Another well-established research stream has focused on the technical challenges, code quality, and maintenance practices that can affect the effectiveness of development activities over time [1, 22, 23, 27, 31, 38, 29]. Within the context of web GUI testing, prior works have proposed techniques for repairing broken tests [32], detecting dependencies in web GUI tests [6], predicting fragility [14], localizing the root cause of flaky behavior [28], or supporting the automatic generation of web GUI test suites [25, 7, 9]. Practitioner-oriented studies [24] similarly show that, despite the availability of web GUI testing frameworks that

support automation, manual testing is still widespread due to tool limitations, complexity, and the expertise needed to build reliable suites. Overall, this literature shows that web GUI testing is still affected by both technical limitations and maintenance costs. However, these studies typically investigate these challenges at the test or tool level rather than the decision processes through which teams adopt, retain, or abandon frameworks.

More closely related to our work are studies on the adoption and maintenance of GUI testing in real projects [8, 2]. Di Meglio et al. [20] examined the adoption and maintenance of web GUI testing in real-world repositories, finding both increasing use of modern frameworks and frequent framework migrations, with about 25% of projects migrating at least once. However, while their study showed that such transitions are common, it also found that the rationale behind them is rarely explicit in repository artifacts and identified understanding these decisions as an important direction for future work.

Literature on other testing domains points in a similar direction. Studies on the adoption and maintenance of performance testing [13, 19, 17] and broader research on the evolution of tests and software artifacts [4] show that tooling decisions are shaped by both technical and organizational factors. Yet, compared with unit testing and other quality-assurance practices, the socio-technical motivations behind adopting or migrating web GUI testing frameworks remain largely unexplored.

Our study addresses this gap by investigating the rationale for adopting and migrating to web GUI testing frameworks. Rather than focusing on technical features or prevalence alone, we analyze commit and issue discussions to extract explicit motivation statements and derive a thematic categorization of the socio-technical factors influencing these decisions. In doing so, we complement prior work on framework comparison, maintenance, and adoption trends by shifting attention from what frameworks do and how often they are used to why teams choose them and why those choices evolve over time.

3 Goal and Research Questions

The *goal* of this study is to investigate the socio-technical decision processes that shape the lifecycle of web GUI testing, with the *purpose* of understanding the reasons behind the adoption of and migration between browser automation frameworks. The *perspective* of this work is twofold. For researchers, our study provides empirical insights into socio-technical decision-making and tool evolution in software engineering. For practitioners, the results offer an understanding of the motivations driving adoption and migration, supporting more informed framework selection and risk assessment. In this paper, we use the term socio-technical rationale to denote an explicit justification linking a framework decision to both technical characteristics and the human, organizational, or ecosystem context in which the framework is used. The term socio-technical does not imply that every rationale simultaneously contains both social and technical elements. Rather, the resulting categorization captures motivations that emerge

within a socio-technical setting, where framework decisions are shaped by the interplay between technical characteristics and the organizational, developer, and ecosystem contexts in which they are made.

> **RQ$_1$**. *What factors influence the adoption of web GUI testing frameworks?*

With this research question, we analyze the motivations behind projects to introduce a browser automation framework, focusing on the socio-technical factors that shape early decision-making and lower barriers to entry.

> **RQ$_2$**. *What factors influence the migration between web GUI testing frameworks?*

With this research question, we investigate why projects replace an existing framework with an alternative, capturing the technical and social pressures that arise as systems and requirements evolve.

By analyzing these research questions we aim to assess how decision-making criteria evolve over the project lifecycle, and which socio-technical dimensions become more or less prominent over time.

4 Research Method

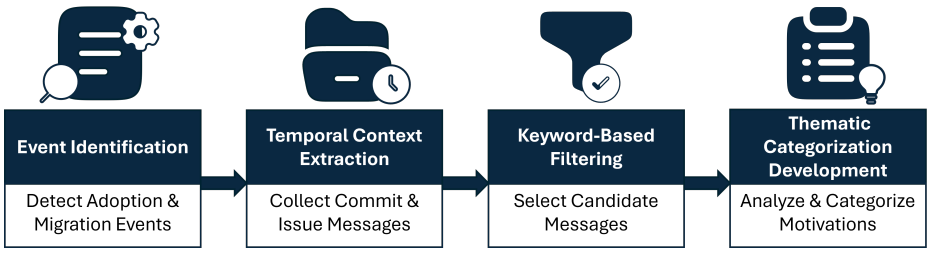

Fig. 1. Overview of the research methodology pipeline.

Figure 1 provides an overview of the research method performed to address our research questions. In the following, we describe each step in greater detail.

4.1 Event Identification

The first step of our methodology consists of identifying events of framework adoption and migration within open-source projects. We rely on the E2EGit dataset [18], a collection of 472 non-trivial web application repositories hosted

on GitHub. The dataset includes projects that adopt automated web GUI testing practices using one of the following browser automation frameworks: Playwright (197 repos), Cypress (187 repos), Selenium (87 repos), and Puppeteer (20 repos). Regarding the implementation of the test suites, the tests are written across four primary programming languages, consisting of 283 repositories using TypeScript, 130 using JavaScript, 72 using Java, and 10 using Python. Additional details are provided in [18].

The E2EGit dataset was previously used by Di Meglio et al. [20] to investigate adoption and maintenance dynamics of web GUI testing. Their publicly available replication package provides metadata that we leverage to operationalize adoption and migration events. Specifically, we extract:

- **Adoption events:** For each of the 472 repositories, we identify the commit in which web GUI tests were first introduced. We marked this commit as the project's initial adoption event of a web GUI testing framework. This results in a total of 472 adoption events, one per repository.
- **Migration events:** For projects that replaced an existing browser automation framework with a different one during their evolution, we identify the commit(s) associated with the framework transition. We marked these commits as migration events, capturing the point in time at which the testing infrastructure changed. In total, 97 projects experienced at least one migration, resulting in 156 migration events identified in the replication package [20], with Playwright emerging as the main target.

4.2 Temporal Context Extraction

Once adoption and migration commits were identified, we collected evidence of the rationale underlying these events. We gathered data from two complementary sources: commit messages and issue discussion threads. These sources frequently include explanations of technical decisions, alternatives considered, trade-offs discussed, and community input that influenced the final choice, thus providing insight into the motivations behind the observed framework changes.

However, the rationale for adopting or migrating to a framework may not always be explicitly documented in the corresponding commit/issue message. In many cases, the decision emerges from prior discussions in issue threads or earlier commits. Therefore, following the approach and adopted in previous studies [19, 20], we applied a temporal window strategy to capture related contextual information. Specifically, for each identified adoption or migration commit, we collected the 10 previous commit messages (including the focal commit) and all issue discussions within a 10-day window before the event (including the day of the focal commit), as established in [20].

4.3 Keyword-Based Filtering

The collection process resulted in an initial sample of 5,032 commit messages and 38,120 issue discussion messages related to adoption events, while we collected

1,716 commit messages and 16,207 issue messages related to migration events. Due to the large size of the initial dataset, a full manual inspection was infeasible. Therefore, we applied a preliminary filtering step to identify candidate messages likely to contain explicit rationale statements.

The filtering strategy was based on terms and expressions commonly associated with framework adoption and migration discussions. These terms were identified through an exploratory inspection of repository discussions and informed by terminology used in prior repository-mining studies [19, 20, 37]. In addition to framework names, we targeted action-oriented language that typically signals deliberate technical decisions. In addition to the names of the supported browser automation frameworks, we targeted action-oriented language that typically signals deliberate technical decisions. For adoption-related events, examples include expressions such as *"add e2e"*, *"first test"*, or *"introduce test"*, as well as expressions indicating intentional introduction or setup (e.g., *"adoption"*, *"create test"*, *"new test suite"*, *"test setup"*). For migration-related events, the filtering focused on terms suggesting replacement or transition, such as *"migrate"*, *"switch"*, *"replace"*, *"move"*, or *"drop"*. We defined a list of 25 keywords for adoption and 53 keywords for migration. The complete list of filtering terms is provided in our replication package [12].

The output of this phase resulted in 587 commit messages and 9,290 issue messages related to adoption events, and 337 commit messages and 4,487 issue messages related to migration events.

The filtering was designed to maximize recall rather than precision, intentionally accepting a high number of false positives. Our exploratory analysis showed that generic terms such as *"bug"*, *"browser"*, or *"ui"* often generated thousands of matches unrelated to framework decisions. As a result, most candidate messages still required manual qualitative assessment to distinguish simple keyword occurrences from genuine rationale statements.

4.4 Thematic Categorization Development

The candidate messages were then manually analyzed by the first and second authors to identify those explicitly reporting motivations behind the adoption or migration of web GUI testing frameworks.

To address the large number of false positives produced by keyword filtering, we manually reviewed all candidate messages. To support this process, we developed a custom web-based interface that enabled the first and second authors to systematically evaluate each artifact. Candidate messages were assessed against three inclusion criteria: (1) *Relevance*, requiring a direct connection to framework adoption or migration; (2) *Substantiveness*, requiring an explicit rationale, evaluation, or comparison rather than a superficial mention (e.g., "bumped Cypress version"); and (3) *Decision Context*, requiring evidence of the trade-offs or reasoning underlying the technical choice. The application of these criteria substantially reduced the dataset, as most candidate messages contained no explicit motivation and therefore did not qualify as rationale-rich artifacts.

The two authors independently performed the qualitative analysis and produced separate preliminary thematic categorizations of socio-technical motivations. The two categorizations were then compared and merged through discussion. In cases of disagreement regarding message inclusion or category assignment, the third author was involved to arbitrate and reach a consensus.

5 Results

We identified 72 relevant messages (10 commits and 62 issues), extracted from 49 repositories, and a total of 52 events connected to adoption and migration of web GUI testing frameworks. More specifically, commit messages were obtained from 8 repositories and issue discussions from 45 repositories, with 4 repositories contributing evidence from both sources. These comprise 25 adoption events (supported by 7 commits and 28 issues) and 27 migration events (supported by 3 commits and 34 issues). Natably, multiple categories were associated with 16 events (30.8%), indicating that adoption and migration decisions are often driven by a combination of concurrent socio-technical factors rather than by a single dominant motivation. This suggests that framework-related decisions are inherently multi-dimensional, reflecting trade-offs among technical, organizational, and developer-centric concerns.

From these messages, we inductively derived a thematic categorization of eight socio-technical rationales underlying these decisions, as shown in Table 1. We do not interpret these categories as necessarily exclusive to web GUI testing; rather, their relevance lies in how they are instantiated, combined, and prioritized in the lifecycle of web GUI testing infrastructure.

5.1 RQ_1: What factors influence the adoption of web GUI testing frameworks?

The analysis of adoption events reveals that decisions are primarily driven by developer-centric and integration-related considerations, rather than by performance or external constraints. This suggests that early-stage framework selection is shaped more by ease of adoption and fit within the existing development context than by long-term operational concerns.

Figure 2 shows the results: the most prominent category is **M3** (*Developer Experience & Usability*) (28.1%), suggesting that projects strongly prioritize frameworks that are easy to learn, well-documented, and supported by effective tooling. Several adoption discussions explicitly emphasize the importance of a smooth onboarding experience, even for developers with no prior exposure to the framework. For instance, a contributor in `badges/shields` project[3] described their first use of Cypress as a *"great experience"*, highlighting its documentation and CI integration as key advantages. Similarly, in `fossbilling/fossbilling`

[3] https://github.com/badges/shields/pull/3261

Table 1. A thematic categorization of socio-technical motivations for framework adoption and migration.

ID	Category	Description
M1	Advanced Features & Testing Capabilities	Decisions driven by the need to leverage advanced or unique framework capabilities that go beyond basic end-to-end testing functionalities.
M2	Compatibility & Environment Support	Motivations related to integration with the existing technology stack, cross-platform support, or the ability to interact with complex or dynamic web elements.
M3	Developer Experience & Usability	Decisions influenced by perceived ease of use, quality of documentation, debugging support, and overall learning curve.
M4	Performance & Efficiency	Motivations centered on improving execution speed, optimizing resource usage, and enabling more efficient CI/CD pipelines.
M5	Reliability & Stability	The need to ensure a stable and trustworthy test suite, where failures reflect actual defects rather than tool-related issues.
M6	Strategic & Ecosystem Alignment	Decisions guided by long-term considerations such as tool standardization, alignment with organizational practices, or consistency with related projects.
M7	Test Architecture & Code Quality	Motivations related to improving test maintainability, readability, and robustness through better design practices and architectural patterns.
M8	Tool Lifecycle & External Factors	External drivers such as deprecation, lack of support, or unresolved critical issues that force adoption or migration decisions.

project[4], a developer emphasized the framework's ease of use and productivity-enhancing features, such as visual test authoring tools, noting that it was *"very easy for any team to adopt"*. These observations reinforce the central role of usability in lowering the barrier to entry.

The second most frequent factor is **M2** (*Compatibility & Environment Support*) (18.8%), which further highlights the importance of seamless integration with the existing technology stack. This indicates that adoption decisions are strongly constrained by the surrounding development ecosystem, including language compatibility, infrastructure requirements, and the ability to interact with

[4] https://github.com/fossbilling/fossbilling/pull/786

dynamic web components. In this sense, ease of integration complements usability by reducing the effort required to incorporate the framework into ongoing development workflows.

The remaining categories occur less frequently but provide additional insight into secondary decision criteria. **M7** (*Test Architecture & Code Quality*) (15.6%) indicates that some projects already consider maintainability and test design early in the adoption process, suggesting a forward-looking perspective in certain cases. **M6** (*Strategic & Ecosystem Alignment*) (12.5%) reflects organizational and standardization concerns, such as aligning with existing tools or practices across teams. In contrast, **M5** (*Reliability & Stability*) (9.4%) and **M1** (*Advanced Features & Testing Capabilities*) (9.4%) appear to play a supporting role, suggesting that while these aspects may reinforce adoption decisions, they are not typically the primary drivers. Finally, **M4** (*Performance & Efficiency*) (3.1%) and **M8** (*Tool Lifecycle & External Factors*) (3.1%) are rarely cited, suggesting that concerns related to execution speed or tool longevity are not primary drivers at this stage.

5.2 RQ_2: What factors influence the migration between web GUI testing frameworks?

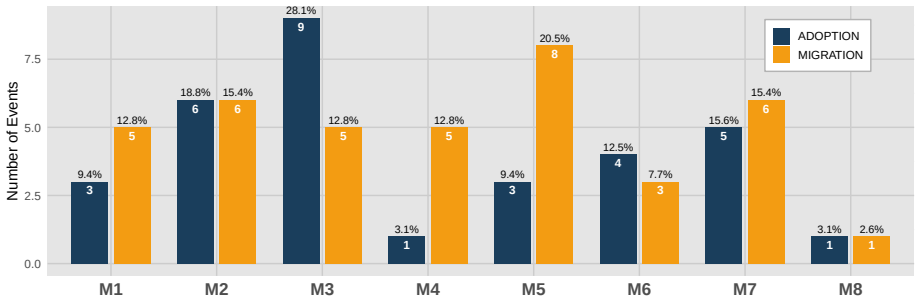

Fig. 2. Grouped bar chart comparing motivation categories for adoptions vs migrations. Bar labels report absolute counts, while percentages are computed within each group over the total number of assigned motivations, as events may be associated with multiple categories.

Figure 2 presents the distribution of motivations driving framework migration. In contrast to adoption, migration is primarily shaped by operational concerns that emerge as projects evolve, with accumulated limitations related to reliability, performance, and integration becoming increasingly prominent.

The most frequently observed factor is **M5** (*Reliability & Stability*) (20.5%), indicating that migration tends to occur when teams experience recurring issues such as flaky or unstable tests. Such issues could gradually reduce confidence in the testing framework and increase maintenance effort, potentially reaching

a point where replacement becomes more practical than continued repair. This pattern is illustrated in `pinterest/gestalt` project[5], where developers explicitly reported the burden of flaky tests, noting that *"a developer needs to re-run tests manually if a test doesn't pass after 2 retries"*.

Other important factors include **M2** (*Compatibility & Environment Support*) (15.4%) and **M7** (*Test Architecture & Code Quality*) (15.4%), suggesting that migration may also be influenced by increasing misalignment between the framework and evolving system requirements. As projects scale, existing tools may become harder to integrate or less suitable for maintaining clean and modular test structures, prompting teams to seek alternatives that better support long-term maintainability.

A second group of factors, namely **M4** (*Performance & Efficiency*) (12.8%), **M3** (*Developer Experience & Usability*) (12.8%), and **M1** (*Advanced Features & Testing Capabilities*) (12.8%), appears to play a more balanced role. In particular, performance concerns seem to become more relevant during migration than during adoption, possibly because execution time and resource usage become more noticeable as test suites expand. For instance, in `pinterest/gestalt` project[6], the migration from Cypress to Playwright was justified not only by maintainability concerns but also by substantial efficiency gains.

Meanwhile, usability and feature-related aspects remain relevant but appear less dominant compared to operational concerns. This suggests that while ease of use may influence tool selection, it is less likely to justify a migration unless accompanied by more critical limitations. Finally, **M6** (*Strategic & Ecosystem Alignment*) (7.7%) and **M8** (*Tool Lifecycle & External Factors*) (2.6%) occur less frequently, indicating that external or organizational drivers play a secondary role compared to technical and operational pressures.

6 Discussion and Implications

The results presented in Section 5 suggest that the criteria used to evaluate web GUI testing frameworks are not static, but evolve over time as projects mature. Different concerns become more or less prominent depending on the stage of the testing lifecycle. To characterize this evolution, we introduce the notion of *Priority Inversion*, defined as a shift in the relative importance of decision criteria between framework adoption and migration. While adoption decisions are primarily influenced by developer experience and usability, migration decisions place greater emphasis on reliability, compatibility, and performance. Notably, these latter factors are closely related to the trustworthiness of testing infrastructures, as they influence whether test outcomes can be relied upon as accurate indicators of software quality. For instance, developer experience decreases from 28.1% in adoption to 12.8% in migration, whereas reliability increases from 9.4% to 20.5%, and performance from 3.1% to 12.8%. The observed Priority Inversion also clarifies why studying web GUI testing frameworks separately from generic

[5] https://github.com/pinterest/gestalt/pull/2125
[6] https://github.com/pinterest/gestalt/pull/2125

software engineering tools is useful. General tool-adoption models help explain why ease of use and integration matter when a tool is introduced. However, in web GUI testing, the long-term adequacy of a framework is strongly tied to the trustworthiness of test outcomes: unstable tests, slow execution, and poor compatibility with evolving web interfaces can undermine confidence and increase the cost of interpreting failures.

During adoption, decisions tend to prioritize factors that reduce initial friction, such as ease of use and integration. As projects evolve, however, accumulated issues related to flakiness, execution costs, and integration limitations make reliability, compatibility, and performance increasingly important drivers of migration decisions.

Taken together, these findings suggest that migration is not merely the result of identifying superior alternatives, but rather a process of *re-evaluating prior decisions* under changing conditions. In this sense, migration can be interpreted as a corrective step, where choices initially optimized for ease of adoption are revisited in light of long-term system demands.

6.1 Implications for Practitioners

For practitioners, the findings highlight the importance of considering how evaluation criteria may evolve over time, rather than focusing solely on immediate needs. First, tool selection decisions may benefit from explicitly accounting for future operational requirements. In practice, this means complementing short-term criteria (e.g., ease of use, documentation quality) with an early assessment of factors such as scalability, test execution time, CI/CD integration, and long-term maintainability. For instance, teams may incorporate these aspects into their evaluation checklists or pilot studies when comparing frameworks.

The results also suggest that reliability should be treated not only as a technical attribute but also as a long-term cost factor. Teams may monitor indicators such as test flakiness rates, frequency of re-runs, or debugging effort associated with unstable tests. Tracking these signals can help quantify the hidden cost of unreliable test suites and support data-driven decisions about when intervention is needed. Early mitigation strategies, such as improving test design or adopting more robust tooling, may reduce the need for more disruptive changes.

Finally, the notion of *Priority Inversion* may help inform migration timing. Rather than reacting only when operational issues become critical, teams may benefit from periodically reassessing their testing frameworks against evolving project requirements. This could take the form of lightweight periodic reviews (e.g., at major releases) that evaluate whether the current framework still meets performance, reliability, and integration needs. Such proactive reassessment can enable more controlled and incremental transitions, reducing migration risks.

6.2 Implications for Researchers

For researchers, the results suggest that adoption and retention may be driven by different, and potentially competing, factors. While ease of use and developer

experience appear central to initial adoption, long-term usage seems to depend more on reliability, performance, and architectural alignment. This distinction suggests that evaluations based solely on adoption metrics (e.g., popularity or initial uptake) may overlook factors affecting long-term sustainability. Future empirical studies may therefore benefit from distinguishing between adoption, retention, and migration when analyzing software tools.

The *Priority Inversion* phenomenon also raises questions about how tools should be designed and evaluated. Rather than optimizing for a single set of criteria, frameworks may need to support different stages of the project lifecycle. Research could explore ways to reduce trade-offs between usability and operational robustness, such as improving debugging support for flaky tests or enhancing scalability for large test suites.

Further work could deepen the understanding of migration drivers through additional sources of evidence. For example, finer-grained analyses of repository histories may help reconstruct the sequence of events leading to adoption and migration decisions, as explored in prior work [35, 34, 36]. Complementary methods, such as surveys and semi-structured interviews, could provide direct insights into practitioners' reasoning and enable triangulation with repository-based findings.

Finally, future research could investigate LLM-based techniques to support or partially automate the identification of rationale-bearing discussions in software repositories, building on recent evidence that such models can effectively classify issue reports [11]. This could improve the scalability of similar studies and enable analyses on larger datasets.

7 Threats to Validity

In the following, we discuss the possible limitations of our study, along with the mitigation strategies [39].

Construct Validity. Our study relies on an existing dataset and on the identification of adoption and migration events from prior work [18, 20]. While this provides a validated foundation, it may also propagate underlying assumptions. To mitigate this risk, we reviewed the operationalization of events and derived rationale statements from independently collected evidence (i.e., commit messages and issue discussions). To better contextualize events, we used a temporal window established in prior work [20]. Although relevant discussions may occur outside this window, as shown in a previous study [20], the selected configuration represents a practical trade-off between coverage and feasibility. Moreover, commit messages and issue discussions provide a partial view of developers' decision-making. As an inherent limitation of repository-based studies, some motivations may remain undocumented or be discussed informally, leading to an incomplete representation of adoption and migration decisions. The manual qualitative analysis may be subject to researcher bias. To mitigate this risk, two authors independently analyzed the data and resolved disagreements through discussion with a third author [19]. Finally,

the keyword-based filtering, based on 78 predefined terms, may have excluded relevant messages expressed through indirect language, project-specific terminology, or vocabulary not covered by the keyword set. This represents an inherent limitation of keyword-based analyses, as rationale statements may be expressed using heterogeneous or implicit language. To reduce this risk, the keyword set was designed to cover a broad range of adoption- and migration-related expressions. All data and artifacts are publicly available [12].

External Validity. The generalizability of our findings may be limited by the scope of the dataset. Our study relies on the E2EGit [18] dataset, which includes GitHub repositories using four frameworks: Cypress, Playwright, Selenium, and Puppeteer. While these frameworks are widely adopted, they do not cover the full diversity of available tools. However, they span different generations and design approaches, allowing us to observe varied adoption and migration behaviors. Moreover, GitHub repositories may not fully reflect proprietary or enterprise practices. Although open-source projects provide rich and accessible rationale data, the findings may not fully generalize to closed-source or highly regulated environments. Future studies involving industrial datasets or practitioner-based methods could further assess generalizability.

Conclusion Validity. Conclusion validity may be affected by the interpretation of the distributions. The reported differences between adoption and migration motivations should be interpreted as indicative patterns rather than causal relationships. The categorization of motivations may also introduce subjectivity. To mitigate this, multiple authors independently performed the classification and resolved disagreements through discussion. Nevertheless, minor variations in category assignment could influence the results. Finally, as the study is based on observational data and focuses on a specific ecosystem of web GUI testing frameworks, it does not allow causal inference and its findings should not be interpreted as a universal model of software engineering tool adoption and migration. The identified patterns provide plausible explanations of decision-making behavior, but further validation through controlled studies, practitioner feedback, and investigations in other tooling ecosystems would be needed to confirm and generalize them.

8 Conclusion

This paper presents an empirical study of the motivations behind adopting and migrating web GUI testing frameworks, based on 472 open-source repositories. Through the manual analysis of 52 adoption and migration events, we derived a thematic categorization of eight socio-technical motivations. Adoption was mainly driven by developer-oriented factors, such as *Developer Experience & Usability*, whereas migration was more often motivated by operational concerns, including *Reliability & Stability* and *Performance & Efficiency*. We also identified a *Priority Inversion*: factors that encourage initial adoption gradually give way to those required for long-term sustainability. This suggests that framework

selection is an evolving process shaped by changing technical and organizational needs. More broadly, our findings highlight the importance of continuously reassessing testing infrastructures to maintain trustworthy software systems. Future work will extend this investigation to industrial settings of these software systems. Future work will extend this investigation to industrial settings and complementary methods such as surveys and interviews.

Acknowledgments. This paper was partly supported by the Italian PNRR MUR project PE0000013-FAIR and the European HORIZON-KDT-JU-2023-2-RIA research project MATISSE (grant 101140216-2, KDT232RIA 00017).

References

1. Afeltra, A., Cannavale, A., Pecorelli, F., Pontillo, V., Palomba, F.: A large-scale empirical investigation into cross-project flaky test prediction. IEEE Access **12**, 131255–131265 (2024)
2. Alégroth, E., Feldt, R., Kolström, P.: Maintenance of automated test suites in industry: An empirical study on visual GUI testing. Information and Software Technology **73**, 66–80 (2016)
3. Balsam, S., Mishra, D.: Web application testing—challenges and opportunities. Journal of Systems and Software **219**, 112186 (2025)
4. Barbosa, L., Hora, A.: How and why developers migrate Python tests. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 538–548. IEEE (2022)
5. Battista, E., Di Martino, S., Di Meglio, S., Scippacercola, F., Starace, L.L.L.: E2E-Loader: A framework to support performance testing of web applications. In: 2023 IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 351–361 (2023)
6. Biagiola, M., Stocco, A., Mesbah, A., Ricca, F., Tonella, P.: Web test dependency detection. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 154–164 (2019)
7. Biagiola, M., Stocco, A., Ricca, F., Tonella, P.: Diversity-based web test generation. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 142–153 (2019)
8. Christophe, L., Stevens, R., De Roover, C., De Meuter, W.: Prevalence and maintenance of automated functional tests for web applications. In: 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE (2014)
9. Corazza, A., Di Martino, S., Peron, A., Starace, L.L.L.: 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 (ESEM). pp. 1–6 (2021)
10. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly **13**(3), 319–340 (1989)
11. De Vito, G., Starace, L.L.L., Palomba, F., Di Martino, S., Ferrucci, F.: Advancing LLM-Based issue report classification with explained few-shot learning, intent extraction, ensemble, and summarization. ACM Trans. Softw. Eng. Methodol. (Apr 2026)

12. Di Martino, G., Di Meglio, S., Pontillo, V., Starace, L.L.L.: squidslab/wgt-shift: Seaa replication package (Jun 2026). https://doi.org/10.5281/zenodo.19741349
13. Di Meglio, S., Pontillo, V., Starace, L.L.L., Martins, L., Di Nucci, D., Palomba, F.: Smelly bad practices in performance system tests: Detection, prevalence, and lifetime. In: 42nd IEEE International Conference on Software Maintenance and Evolution (ICSME 2026). IEEE (2026)
14. Di Meglio, S., Starace, L.L.L.: Towards predicting fragility in end-to-end web tests. In: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering. pp. 387–392 (2024)
15. Di Meglio, S., Starace, L.L.L., Di Martino, S.: E2E-Loader: A tool to generate performance tests from end-to-end GUI-level tests. In: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 747–751 (2025)
16. Di Meglio, S., Starace, L.L.L., Di Martino, S.: Web app performance testing in industrial contexts: Supporting workload generation with e2e-loader++. Journal of Systems and Software **232**, 112684 (2026)
17. Di Meglio, S., Starace, L.L.L., Pontillo, V., Martins, L., Di Nucci, D., Palomba, F.: A taxonomy of bad practices in software performance testing: Insights from gray literature and practitioner validation. ACM Trans. Softw. Eng. Methodol. (Jun 2026)
18. Di Meglio, S., Starace, L.L.L., Pontillo, V., Opdebeeck, R., De Roover, C., Di Martino, S.: E2EGit: A dataset of end-to-end web tests in open source projects. In: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR). pp. 836–840 (2025)
19. Di Meglio, S., Starace, L.L.L., Pontillo, V., Opdebeeck, R., De Roover, C., Di Martino, S.: Performance testing in open-source web projects: Adoption, maintenance, and a change taxonomy. In: 41st IEEE International Conference on Software Maintenance and Evolution (ICSME 2025). pp. 199–210. IEEE (2025)
20. Di Meglio, S., Starace, L.L.L., Pontillo, V., Opdebeeck, R., De Roover, C., Di Martino, S.: Investigating the adoption and maintenance of web GUI testing: Insights from GitHub repositories. Information and Software Technology **189**, 107928 (2026)
21. García, B., del Alamo, J.M., Leotta, M., Ricca, F.: Exploring browser automation: A comparative study of Selenium, Cypress, Puppeteer, and Playwright. In: Bertolino, A., Pascoal Faria, J., Lago, P., Semini, L. (eds.) Quality of Information and Communications Technology. pp. 142–149. Springer Nature Switzerland, Cham (2024)
22. Giordano, G., Festa, G., Catolino, G., Palomba, F., Ferrucci, F., Gravino, C.: On the adoption and effects of source code reuse on defect proneness and maintenance effort. Empirical Software Engineering **29**(1), 20 (2024)
23. Giordano, G., Sellitto, G., Sepe, A., Palomba, F., Ferrucci, F.: The yin and yang of software quality: On the relationship between design patterns and code smells. In: 2023 49th Euromicro conference on software engineering and advanced applications (SEAA). pp. 227–234. IEEE (2023)
24. Junior, N., Costa, H., Karita, L., Machado, I., Soares, L.: Experiences and practices in gui functional testing: A software practitioners' view. In: Proceedings of the XXXV Brazilian Symposium on Software Engineering. pp. 195–204 (2021)
25. Kanaththage, K., Starace, L.L.L., Biagiola, M., Tonella, P., Stocco, A.: Neural embeddings for web testing. In: 2026 IEEE Conference on Software Testing, Verification and Validation (ICST) (2026)

26. Leotta, M., García, B., Ricca, F., Whitehead, J.: Challenges of end-to-end testing with Selenium WebDriver and how to face them: A survey. In: 2023 IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 339–350 (2023)
27. Martins, L., Pontillo, V., Costa, H., Ferrucci, F., Palomba, F., Machado, I.: Test code refactoring unveiled: where and how does it affect test code quality and effectiveness? Empirical Software Engineering **30**(1), 27 (2025)
28. Morán, J., Augusto, C., Bertolino, A., De La Riva, C., Tuya, J.: Flakyloc: flakiness localization for reliable test suites in web applications. Journal of Web Engineering **19**(2), 267–296 (2020)
29. Parziale, A., Voria, G., Giordano, G., Catolino, G., Robles, G., Palomba, F.: Fairness on a budget, across the board: A cost-effective evaluation of fairness-aware practices across contexts, tasks, and sensitive attributes. Information and Software Technology p. 107858 (2025)
30. Pontillo, V., Palomba, F., Ferrucci, F.: Test code flakiness in mobile apps: The developer's perspective. Information and Software Technology **168**, 107394 (2024)
31. Recupito, G., Giordano, G., Ferrucci, F., Di Nucci, D., Palomba, F.: When code smells meet ml: on the lifecycle of ml-specific code smells in ml-enabled systems. Empirical Software Engineering **30**(5), 139 (2025)
32. Stocco, A., Yandrapally, R., Mesbah, A.: Visual web test repair. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 503–514 (2018)
33. Talakola, S.: Exploring the effectiveness of end-to-end testing frameworks in modern web development. International Journal of Emerging Research in Engineering and Technology **3**(3), 29–39 (2022)
34. Tufano, M., Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., Poshyvanyk, D.: An empirical investigation into the nature of test smells. In: Proceedings of the 31st IEEE/ACM international conference on automated software engineering. pp. 4–15 (2016)
35. Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D.: When and why your code starts to smell bad. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, pp. 403–414. IEEE (2015)
36. Vassallo, C., Grano, G., Palomba, F., Gall, H.C., Bacchelli, A.: A large-scale empirical exploration on refactoring activities in open source software projects. Science of Computer Programming **180**, 1–15 (2019)
37. Vidoni, M.: A systematic process for mining software repositories: Results from a systematic literature review. Information and Software Technology **144**, 106791 (2022)
38. Voria, G., Scala, B., Todisco, L., Venditto, C., Giordano, G., Catolino, G., Palomba, F.: Fair and square? evaluating fairness of llm-generated synthetic datasets. Information and Software Technology p. 107980 (2025)
39. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., et al.: Experimentation in software engineering, Computer Science, vol. 236. Springer, Berlin, Germany (2012)