An Empirical Study of Web Visual Flakiness: Characterisation and Fix Strategies

Yu Pei, Jeongju Sohn[1], Mike Papadakis

[a]*yu.pei@uni.lu University of Luxembourg, Luxembourg*
[b]*jeongju.sohn@knu.ac.kr Kyungpook National University, Korea*
[c]*michail.papadakis@uni.lu University of Luxembourg, Luxembourg*

Abstract

Unpredictable behaviour in web front-ends undermines test reliability and user experience, often manifesting as inconsistent or unstable visual output. Prior work on web front-end flakiness has mainly focused on asynchronous waits, platform-specific behaviour, and logic errors, many of which stem from client-side execution, back-end dependencies, or timing variations across browsers. However, little attention has been given to pure visual flakiness–instabilities in layout, rendering, and styling that cause the same web content to appear differently across runs or environments.

In this work, we present a focused empirical study dedicated to web visual flakiness, defined as flakiness that manifests as structural or stylistic inconsistencies in web front-ends. We collected 262 cases of web visual flakiness: 144 from 31 open-source web projects and 118 from the Chromium project. Our findings show that web visual flakiness constitutes a notable portion of overall flaky cases and that visual changes are more prone to flakiness than non-visual ones, underscoring the need for focused investigation. We categorised the collected instances into two primary dimensions–structure-related (59.9%) and style-related (40.1%)–and further refined them into five concrete sub-categories by manifestation (three structure-related and two style-related). Finally, our analysis of developer fixes shows that while many address specific structural or stylistic inconsistencies, others are applied across multiple categories, revealing shared repair strategies.

Keywords:
Flaky Behaviour, Web Visuals, Web Testing, Empirical Study

[1]Corresponding author

1. Introduction

As web applications grow increasingly complex, ensuring their reliability has become a critical challenge. Among many obstacles, flaky behaviours, such as test outcomes that show non-deterministic results across repeated runs, pose a constant threat to automated testing Luo et al. (2014); Parry et al. (2021); Lam et al. (2019a). This issue is particularly pronounced in the visual layer, where minor inconsistencies, such as browser-specific differences, asynchronous content loading, and ordering of DOM updates, can lead to unpredictable rendering and test failures Eck et al. (2019); Parry et al. (2022); Habchi et al. (2022); Gruber and Fraser (2022); Haben et al. (2024).

The visual layer encompasses the code and frameworks that define and render visual presentation in web rendering environments, including markup (e.g., HTML), stylesheets (e.g., CSS), and client-side logic (e.g., JavaScript). These components collectively control the structure and styling of the visual layer, making them central to both user experience and test stability. However, even subtle changes to these components can lead to visual inconsistencies that are difficult to detect and reproduce due to their non-deterministic nature (i.e., flakiness). Responsive design and dynamic content further amplify this instability across browsers and environments.

While previous research has investigated flaky tests in web and mobile environments, such as interaction-based GUI flakiness (e.g., Romano et al. (2021)) and deterministic UI rendering defects (e.g., Mahajan et al. (2016); Alameer and Halfond (2016); Althomali et al. (2021)), the focus has primarily been on functional-level non-determinism (e.g., async waits, improper test script logics, or state mismatches) or deterministic visual inconsistencies (e.g., cross-browser layout bugs). Meanwhile, studies on web flaky tests have primarily investigated non-determinism in functional or interaction-driven contexts, addressing causes such as asynchronous waits, platform-specific behaviour, or state management Parry et al. (2021); Lam et al. (2019a).

Taken together, prior studies have examined interaction-based GUI flakiness and other forms of functional non-determinism in UI behaviour, as well as deterministic visual faults, such as cross-browser layout bugs. However, they have not explicitly isolated non-deterministic visual flakiness—often manifesting as unstable layout, rendering, or styling—as a distinct empirical category. In this work, we investigate such visual flakiness as instabilities rooted in the visual layer that are inconsistently observable across executions and environments, bridging the gap between web flakiness research and

studies of visual correctness.

At the same time, web-visual testing is widely practised in industry, for instance, through screenshot-based or layout-diff validation Yeh et al. (2009); Choudhary et al. (2010, 2011); Stocco et al. (2018), yet its flaky behaviour remains largely underexplored. Such flakiness is commonly observed in modern industrial web projects, highlighting the need for a dedicated empirical investigation Rwemalika et al. (2019); Zou et al. (2014); Nass et al. (2023).

We present this study as a focused empirical investigation that systematically characterises non-deterministic visual flakiness in web applications as a distinct analytical focus. To our knowledge, this study provides the first systematic empirical analysis that explicitly focuses on web visual flakiness, examining its prevalence, characteristics, and developer responses. By analysing 262 flaky cases from 31 open-source web projects and Chromium, we identify how, where, and when visual flakiness manifests in real-world development. Based on recurring patterns, we categorise them into structure-related and style-related types. We also analyse how developers address visual flakiness in practice, inspecting how these fixes relate to the defined categories, and identifying common and cross-cutting fix strategies. In the end, we address the following research questions:

- **RQ1**: What is the prevalence of flaky behaviours in web visuals?
- **RQ2**: What are the most common types of web visual flakiness, and how do they manifest in practice?
- **RQ3**: What fix patterns can be employed to mitigate the flaky behaviour in web visuals?

Our findings, combined, highlight the importance of understanding visual flakiness to improve the reliability of web testing. Below summarises the key contributions:

- We conduct a focused empirical study of web visual flakiness as a distinct class of flaky behaviour, analysing 262 real-world cases across 31 open-source web projects and the Chromium codebase, and release the resulting curated dataset to support future research.
- We provide a taxonomy of web visual flakiness, distinguishing two high-level types, i.e., structure-related and style-related, based on recurring manifestation patterns, and further refining each type into five concrete subcategories.
- We investigate how developers handle web visual flakiness in practice,

identifying common, cross-cutting repair strategies and analysing their relationships with different flakiness categories.

2. Methodology

2.1. Motivating Example

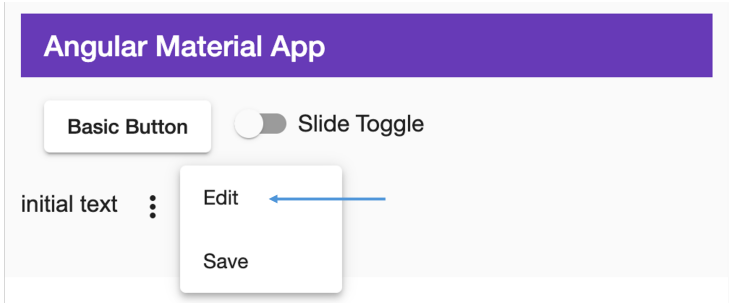
This study is motivated by the growing complexity of modern web front-ends, where frequent modifications to DOM structure and visual styling can introduce instability in the visual behaviour observed during execution. The front-end of the web application is the layer where users directly interact with the system, making visual instability particularly impactful. Even minor inconsistencies, such as unexpected element positioning or missing visual focus, can disrupt user actions and degrade user.

Consider the example in Figure 1a[2], where clicking the *Edit* menu item is intended to automatically focus the text box, enabling users to begin editing immediately (Figure 1b). However, as shown in Figure 1c, the text box sometimes fails to gain focus, requiring users to manually click it again. While the underlying logic remains unchanged, this execution-time visual instability, manifesting as inconsistent focus behaviour (i.e., flakiness), disrupts user workflow and causes frustration, particularly in applications where users expect quick and fluid interactions. Such visual flakiness is often difficult to detect and diagnose. Understanding the nature and recurring patterns of such flakiness requires reasoning about instability in the visual layer, which is inherently subtle, brittle, and sensitive to small variations Mahajan et al. (2016); Pei et al. (2025).

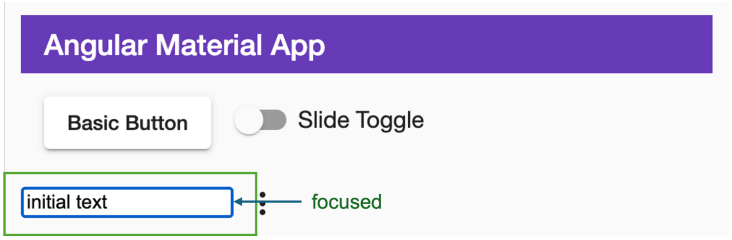
2.2. Study Design

We conducted an empirical study to characterise flaky behaviours in web visuals in real-world projects. To this end, we first collected commits and, when available, their respective pull requests or issue reports related to web visual flakiness from diverse web projects. We then investigated the observed types of visual flakiness derived from recurring patterns and how developers address these issues in practice, including common fix strategies. To support this investigation, we performed a combination of manual inspection and LLM-assisted annotation to categorise both visual flakiness types and their corresponding fix strategies.

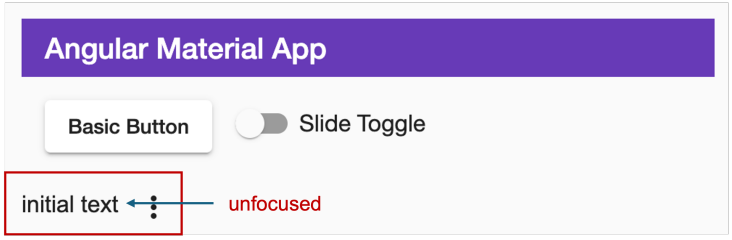
[2]https://github.com/angular/components/issues/18750



(a) Click "Edit" item to edit the "initial text"



(b) The expected behaviour after clicking: auto-focus the text box



(c) The flaky behaviour of the text box sometimes loses focus and is unable to modify the text

Figure 1: Example: flaky behaviour in the text box autofocus

2.2.1. Terminology

To clarify the focus of our study, we introduce two key terms that define what we consider visual-related flakiness in web projects. These definitions distinguish our scope from general categories of web or UI flakiness that include functional or interaction-driven behaviours: *visual commits* and *web visual flakiness*.

- **Visual commits** are commits that include changes that influence the visual behaviour or appearance of a web project, such as those related to DOM structure, styling, or rendering logic.
- **Web visual flakiness** refers to non-deterministic, intermittent incon-

sistencies that manifest in the visual layer of web projects. Such inconsistencies may appear, for instance, as layout shifts, rendering differences, or style-related visual inconsistencies.

Based on the above terminology, we define a web visual flakiness-related commit as any visual commit that exhibits flaky behaviour in the visual layer. In contrast, non-visual commits are those whose changes do not influence the rendering or presentation of web content, such as modifications unrelated to the DOM structure or styling.

2.2.2. Dataset Collection

To collect web visual flakiness cases[3], we followed a systematic procedure similar to prior flakiness studies Luo et al. (2014); Romano et al. (2021); Hashemi et al. (2022). We focus on open-source web application projects on GitHub, where commit-level information is available and enables detailed analysis of code changes and their context.

We first identified repositories primarily using web development languages such as JavaScript and HTML and selected those with over 100 stars to ensure their representativeness. In total, we collected 31 projects comprising 489,926 commits (Table 1, third column).

Next, we identified flaky commits by searching commit messages and, when available, associated issues or pull requests for flakiness-related keywords. We use *"flak*"*, *"intermit*"*, and *"unstable"* as our flakiness-related keyword set. This yielded 1,377 candidate commits related to flakiness (*#Flaky* in Table 1). Our primary interest lies in flakiness arising from changes whose effects manifest in the visual behaviour of web applications. Hence, in parallel with the flaky-commit search, we conducted a keyword search for visual commits using visual-related keywords, which contains *"e2e"*, *"UI"*, *"GUI"*, *"style"*, *"CSS"*, *"visual"*, *"layout"*, and *"display"*. If a commit message, issue, or pull request contained any of these visual keywords, we considered it likely to affect the visual parts of the application.[4] In total, 65,626 visual commits were identified ($\#^{Visual}_{commits}$, Table 1).

[3]We use the term flakiness case to refer to an instance of flaky behaviour, regardless of whether it originates from test code or application code.

[4]While some of the collected commits labelled as *"visual"* based on the keyword matching may not directly interact with visual parts, this does not affect the conclusion of our analysis in Section 3.1, as the more precise identification decreases the denominator ($\#^{Visual}_{commits}$) in the frequency analysis, further supporting the findings.

Table 1: Subjects. *#Commits*, $\#_{Commits}^{Visual}$, *#Flaky*, *#FlakyVisual* are the number of total commits, visual-related commits, all flaky-related cases, and visual-related flakiness cases

Projects	#Stars	#Commits	$\#_{\text{Commits}}^{\text{Visual}}$	#Flaky	#FlakyVisual
Chromium	18.9k	–	–	7780	118
vets-website	240	29,633	3506	235	19
ionic-framework	51k	13,970	1736	62	17
web-platform-tests	4.9k	62,667	17,486	621	15
angular-components	24.3k	12,162	2339	40	12
matrix-react-sdk	1.1k	48,466	6749	78	10
storybook	84.1k	63,182	2858	64	7
material-components	17.1k	7,453	1674	31	7
fluentui	18.4k	18,590	2768	18	6
ant-design	92.1k	28,580	4315	14	5
eui	6.1k	6,187	1127	18	5
openverse	243	11,200	621	44	5
material-ui	93.5k	25,656	3635	21	4
chakra-ui	37.6k	9,938	1865	8	4
tailwindcss	82.3k	5,568	806	7	3
bootstrap	170k	22,900	2378	5	2
vuetify	39.7k	16,186	1831	5	2
quasar	25.8k	14,427	1889	5	2
svelte	78.7k	9,344	830	9	2
ghost	47.1k	39,504	4054	50	2
element	54.1k	4,612	527	4	2
blueprint	20.7k	3,581	447	4	2
balm-ui	506	3,131	129	2	2
bulma	49.2k	1,881	120	1	1
nuxt	54.3k	6,754	268	3	1
semi-design	8.4k	3,785	379	2	1
floating-ui	29.8k	2,221	246	3	1
vue.draggable	20.1k	521	22	1	1
vee-validate	10.8k	4,777	136	2	1
vue-material	9.9k	1,132	217	1	1
vite	67.7k	7,051	481	16	1
xyflow	25k	4,867	187	3	1
Open-source	240–170k	489,926	65,626	1377	144
TOTAL	-	489,926	65,626	9157	262

Lastly, to pinpoint web visual flakiness-related commits, we intersected the sets of flaky and visual commits. Two authors manually reviewed each candidate, examining the code changes[5] and contextual information, such as commit messages and, when available, related issues, pull requests, or developer comments, to ensure that each case genuinely involved web UI flakiness. This conservative strategy—combining the intersection of strict flakiness and visual keywords with manual validation—was chosen to prioritise data precision over exhaustive coverage to support the confidence in our findings from subsequent qualitative analysis. Section 5.2 further discusses this data sampling strategy. The last column (*#FlakyVisual*) in the *"Open-source"* row of Table 1 reports 144 identified web visual flakiness cases across the 31 open-source projects.

Chromium. To enhance the representativeness of our study, we also included Chromium, a major industrial browser project. For Chromium, we adapted the keyword-based matching procedure to its issue tracker[6], where flaky behaviours are reported and managed.

Specifically, we searched issue reports for flakiness-related keywords and filtered them with visual-related terms, using the same keyword sets as for the open-source projects. Although Chromium maintains a mirrored GitHub repository, the correspondence between issue reports and individual commits is often incomplete, indirect, or missing. As a result, commit-level information (e.g., code diffs and commit statistics) is not consistently available for Chromium and, thus, commit-level analyses are not performed for this dataset. The collection process yielded 118 web visual flakiness cases from Chromium (the second row in Table 1), resulting in a total of 262 web visual flakiness cases across all studied subjects (the last row in Table 1).

The two primary data sources employed in this investigation exhibit a notable structural distinction. The open-source web projects dataset is derived from GitHub commits, which enables the identification of web visual flakiness and corresponding fixes through commit messages and code diffs. In contrast,

[5]We mainly considered changes to files written in HTML, JavaScript, TypeScript, CSS, or SCSS, as these languages commonly define and influence the visual structure and style of web applications.

[6]https://issues.chromium.org/issues. Chromium issues hosted on this platform include both current records and historical issues migrated from the legacy Monorail system, which we analyse in this study.

the Chromium dataset is derived from issue tracker records, where flaky behaviours are reported and discussed, often without a reliable correspondence to individual commits. Consequently, analyses that rely on commit-level granularity—such as LLM-assisted categorisation (Section 2.2.4) and flakiness frequency per commit (RQ1)—are only conducted for the open-source projects. For Chromium, the same taxonomic framework and validation procedure are applied to manually analyse issue descriptions, comments, and linked patches, ensuring methodological consistency despite differences in data availability.

2.2.3. Web Visual Flakiness Taxonomy

To better understand and analyse web visual flakiness, we manually inspected the collected instances in our dataset to search for recurring patterns. This analysis revealed two broad families of issues: some involved unstable DOM structures, where elements were not reliably present or positioned (e.g., elements missing, misaligned, or late), while others involved changes in appearance due to inconsistencies in style interpretation or resource loading (e.g., CSS conflicts or delayed fonts). Based on these recurring patterns, we derive a taxonomy of web visual flakiness consisting of two high-level families: **structure-related** and **style-related**.

- ***Structure-related flakiness:*** occurs when visual inconsistencies arise due to variations in how elements are rendered, positioned, or updated. These inconsistencies may stem from changes to the DOM structure (e.g., layout shifts due to dynamic element insertion or removal), asynchronous or delayed updates to DOM elements, or failures in loading content that result in missing or incomplete visual components; this type of flakiness occurs regardless of whether the DOM structure is explicitly modified, as even stable structures can exhibit inconsistent rendering under dynamic or environment-dependent conditions. Hence, structure-related flakiness involves all inconsistencies that relate to the state of the DOM structure, including element presence, positioning, and readiness, where positioning refers to whether an element is correctly realised in the layout rather than how its visual style is rendered.
- ***Style-related flakiness:*** occurs when visual inconsistencies arise due to variations in how style properties (e.g., CSS rules, font rendering, or visual appearance attributes) are applied, interpreted, or available across different environments, browsers, or rendering engines at run-

time. These inconsistencies affect the visual appearance of elements without involving changes to the DOM structure or the spatial presence and placement of elements. Style-related flakiness primarily affects visual properties such as colour, size, and font.

In summary, structure-related flakiness concerns which visual elements are present and where they appear on the page, while style-related flakiness concerns how visual elements look. These two represent high-level categories in our taxonomy. Each category includes multiple concrete subcategories that capture distinct recurring manifestations observed in our dataset, which can help developers diagnose and address flaky behaviours. For *structure-related web visual flakiness*, we identify three primary subcategories:

- **DOM Structure Issues**: flakiness where DOM elements (i.e., structural components) are not reliably or fully available in the DOM. This includes elements being delayed, missing, removed, or incompletely rendered due to asynchronous behaviour or race conditions, as well as explicit structural changes (e.g., adding, removing, or reordering elements). These are existential problems independent of event execution – *Whether and how elements are rendered or present.*
- **DOM Layout Instability**: flakiness where the spatial arrangement of elements is inconsistent, leading to overlapping, misaligned, or shifted components. Such effects may be triggered by structural changes such as added, removed, or repositioned nodes. The key distinction from DOM Structure Issues is that the inconsistency appears in layout position rather than in element presence or rendering – *How structural elements are spatially positioned.*
- **Timing-sensitive Visual Triggers**: flakiness that occurs when visual behaviours or event handling are triggered too early or too late relative to when structural elements become ready in the DOM. Examples include situations where expected responses, such as focus, visibility, or transitions, fail because the element was inserted, updated, or removed at the wrong time relative to the event execution. In other words, the failure arises from a synchronisation problem between event execution and element readiness, and not about the presence or position within the DOM. – *When those elements become available relative to an event or trigger.*

For *style-related web visual flakiness*, we identified two primary subcate-

gories:

- **Style Interpretation Issues**: flakiness, where the visual appearance of elements is inconsistent because style rules are interpreted or applied differently at runtime. Examples include variations across environments (e.g., differences in CSS properties, such as flexbox and font), as well as inconsistencies within the same environment (e.g., cascade order, specificity conflicts, or dynamic overrides via inline style injection) – *How styles are interpreted and applied.*

- **Style Resource Latency**: flakiness where the visual appearance changes inconsistently due to delays in loading style-dependent resources. Examples include elements appearing in a fallback or unstyled form due to yet-to-be-loaded style resources, such as fonts or stylesheets – *When style-dependent resources are applied.*

2.2.4. LLM-assisted categorisation

To support consistent and scalable annotation, we employed a large language model (gpt-4o-mini) to assist in labelling verified web visual flakiness instances from open-source web projects. This LLM-assisted procedure was applied across multiple analysis dimensions defined in this study, including the categorisation of visual flakiness types based on the taxonomy introduced in Section 2.2.3, as well as the categorisation of corresponding developer fix strategies analysed in the later research question (RQ3).

For each instance, a structured description including commit metadata and messages, code diffs, and, if available, pull request details or linked issue reports, was processed using a predefined prompt to generate a preliminary label. After preliminary labels were generated through LLM, two authors manually reviewed all label assignments. The LLM-generated labels served as an initial reference during the manual review process, helping structure the inspection and support consistent interpretation of the category definitions across the dataset.

The LLM's confidence scores associated with each label assignment were used to prioritise manual review effort. For label assignments with confidence scores of 0.7 or higher, two authors performed a confirmatory check to verify that the suggested category—whether a visual flakiness subcategory or a fix strategy—aligned with the observed evidence in the code diffs and commit messages associated with resolving the reported flaky behaviour. If

Prompt Template

// Definitions (provided to the model)

Flakiness Type Definitions. Descriptions of web visual flakiness types (three structure-related and two style-related), e.g.,
1. DOM structure issues: ...
2. ...

Fix Strategy Definitions. Descriptions of fix strategies, e.g.,
1. Manage element changes: ...
2. ...

// Task

You are categorising the visual flakiness type and the corresponding fix strategy of a web visual flakiness instance.

// Supporting Evidence

Consider code diffs and issue or pull request descriptions together with commit metadata (e.g., author, date, statistics), commit messages, PR state/labels/-commits/files, and issue labels/state as supporting evidence.

// Instructions

Decide the most likely single flakiness type and single fix strategy from the allowed lists below. Use EXACT strings from the allowed sets. If the evidence is sparse, select the closest plausible category and assign a lower confidence.

Allowed Flakiness Types: {...}
Allowed Fix Strategies: {...}

// Output Format

Return STRICT JSON with keys:
`flakiness_type` (string),
`fix_strategy` (string),
`confidence` (float 0–1),
`rationale` (string),
`evidence` (string).
Use only allowed values.

Figure 2: System prompt used for LLM-assisted categorisation. The system prompt provides type definitions of visual flakiness and fix strategy. It restricts label(type) selection to a predefined set (*Allowed*), and specifies the required structured output format. Statements prefixed with "//" are comments added for readability and are not part of the prompt text itself.

Prompt Template

// Instance-specific context (user prompt)

Context: {*A structured description of the flakiness instance, including commit metadata, commit messages, code diffs, and, when available, titles and descriptions from related pull requests or linked issue reports.*}

Figure 3: User prompt for flakiness instance-specific context. The user prompt provides structured contextual information for each flakiness instance, including commit metadata, code diffs, and, when available, related issue or pull request descriptions.

the confidence score was below 0.7, the authors jointly re-reviewed all relevant artefacts (e.g., code diffs, commit messages, and related pull requests), performing a more in-depth manual inspection and resolving any ambiguities through discussion until consensus was reached.

The final categorisation of each instance, across both visual flakiness types and fix strategies, was determined based on this combined LLM-assisted and manual review process. No additional visual flakiness subcategories emerged in the final categorisation results beyond those derived from the initial pattern analysis described in Section 2.2.3 during the categorisation; the resulting fix strategy assignments are used solely to support the analysis in RQ3 (Section 3.3), where the fix strategy categories are introduced and discussed.

Figures 2 and 3 illustrate the prompts used for the LLM-assisted categorisation. The system prompt provides natural-language definitions of visual flakiness types (sub-categories) and fix strategies, specifies the categorisation task and the types of supporting evidence to consider, lists the allowed labels for each dimension, and defines the required structured output format and labelling rules. The user prompt provides instance-specific context for each case, such as commit metadata, code diffs, and, when available, related issue or pull request descriptions.

As discussed in Section 2.2.2, our dataset also includes web visual flakiness cases from Chromium, for which commit-level artefacts required for LLM-assisted categorisation (e.g., code diffs and commit statistics) are not consistently available. Therefore, the LLM-assisted categorisation procedure above is not applied to the Chromium cases. Instead, Chromium cases are manually labelled by two authors through collaborative review. Each case is assigned to one of the existing web visual flakiness subcategories defined in our taxonomy and derived from the open-source web projects, based on issue

descriptions and discussion comments recorded in the issue tracker and, when explicitly available, linked patches or code review references. The same two authors jointly review each case and resolve disagreements through discussion until consensus is reached. No new flakiness subcategories were introduced beyond those identified from the open-source web projects.

2.2.5. Research Questions

This study aims to deepen the understanding of flaky behaviours in web visuals by exploring the following research questions:

RQ1: *What is the prevalence of flaky behaviours in web visuals?*

To assess the prevalence and impact of visual-related flakiness in web applications and their tests, we conduct three investigations from different perspectives. First, we examine the prevalence of visual commits to establish a baseline for how frequently visual components are modified in our target projects. This provides initial context for evaluating the role of visual changes in flaky behaviour. Next, we analyse the ratio of flaky commits to total commits within visual changes and within non-visual ones (e.g., backend or API). This step assesses whether visual commits are more likely to introduce flaky behaviour than other types of code changes. Lastly, we calculate the proportion of visual flaky commits among all flaky commits across projects. While the previous step compares the relative likelihoods of flakiness between visual flakiness and non-visual flakiness, this step evaluates the overall contribution of visual flakiness to the unreliability of web projects. Note that Chromium is excluded from this analysis, as its flakiness is tracked through issues, not commits (Section 3.1).

RQ2: *What are the most common types of web visual flakiness, and how do they manifest in practice?*

To answer this RQ, we analyse the distribution of visual flakiness across two primary categories—structure-related and style-related—and their respective concrete subtypes (three structure and two style). This two-level categorisation allows hierarchical reasoning about the sources of visual instability, enabling both an overall understanding and a fine-grained examination of individual patterns. In the following analysis, we further illustrate representative cases for each subtype to show how these flaky behaviours manifest in real-world projects, together with a quantitative analysis of their distribution and trends.

RQ3: *What fix patterns can be employed to mitigate the flaky behaviour in web visuals?*

Flaky behaviours degrade test stability, complicate debugging, and thereby diminish user experience. While prior work has proposed general solutions for flakiness, web visual flakiness may require more targeted strategies. By analysing how developers address web visual flakiness through the changes applied to address it, we aim to provide practical insights into fix strategies for different flakiness types, helping developers mitigate them effectively.

3. Results

3.1. ***RQ1: Prevalence of Web Visual Flakiness***

Figure 4 presents the prevalence of visual commits across the 31 open-source projects, measured as the ratio of visual commits to total commits in each project. Chromium is excluded from this analysis due to the unavailability of reliable commit-level data (Section 2.2.2). On average, visual commits constitute 11.7% of all commits in these studied projects. This finding suggests that changes to visual components account for a substantial portion of ongoing web development activities and may therefore contribute significantly to overall project instability, warranting dedicated inspection.

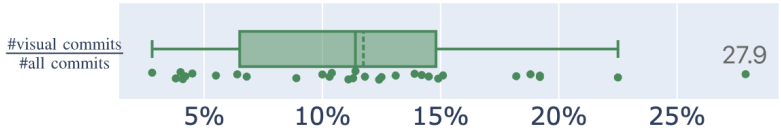
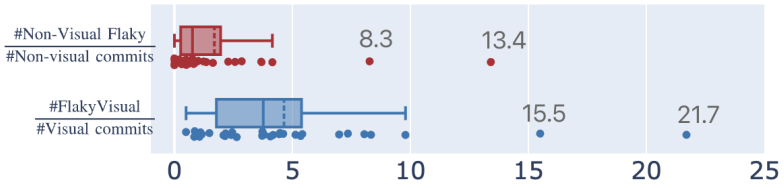


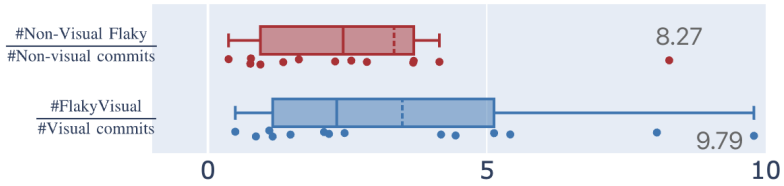
Figure 4: # visual commits to # all commits per project (%). The mean is 11.7%, and the median is 11.4%.

Next, we compare the frequency of flaky commits—i.e., the likelihood of commits exhibiting flakiness—between visual and non-visual commits. Figure 5 presents the distribution of flaky commit ratios for each project (each dot represents one project) within the visual and the non-visual commits.

In this study, we employed a keyword-based search to collect relevant commits and issue reports, following prior flakiness studies Romano et al. (2021). Despite combining multiple metadata sources (commit messages, issue reports, and pull requests), this approach inevitably misses some cases due to incomplete or inconsistent textual descriptions—for instance, when



(a) $\#Flaky > 0$ (no threshold). For $flakyfreq_{visual}$, mean = 4.6, median = 3.7; for $flakyfreq_{non-visual}$, mean=1.7, median=0.8 (‰)



(b) $\#Flaky > 10$. For $flakyfreq_{visual}$, mean = 3.5, median = 2.3; for $flakyfreq_{non-visual}$, mean = 3.3, median = 2.4 (‰)

Figure 5: Flaky frequency. $flakyfreq_{visual} = \frac{\#\text{FlakyVisual}}{\#\text{Visual commits}}$ vs. $flakyfreq_{non-visual} = \frac{\#\text{Non-Visual Flaky}}{\#\text{Non-visual commits}}$. *(‰)* denotes commits per thousand.

developers use ambiguous language or omit flakiness-(or visual) related keywords. Consequently, projects often contained too few identified flaky commits (Table 1), which has also been observed in prior flakiness studies Romano et al. (2021); Hashemi et al. (2022). As this could bias the comparison, we excluded projects with fewer than ten identified flaky commits from the thresholded analysis to ensure reliability, resulting in 14 out of 31 projects. Accordingly, Figure 3 presents two groups of boxplots: one that includes all projects (Figure 5b) and another that applies the threshold (Figure 5a).

Overall, visual commits exhibit a higher frequency of flakiness than non-visual commits. The frequencies shown are normalised per thousand commits (‰). In both analyses, visual commits are consistently more likely to involve flaky behaviour. The median flakiness frequency for visual commits is 3.7‰ (2.3‰ under the threshold), compared to 0.8‰ (2.4‰ under the threshold) for non-visual commits. The corresponding mean frequencies are 4.6‰ (3.5‰) and 1.7‰ (3.3‰), respectively. These results indicate that the visual layers of web projects are often more prone to flaky behaviour than non-visual parts.

Having established that visual commits are both prevalent and more likely to exhibit flaky behaviour than non-visual ones, we lastly examined the overall contribution of visual flakiness to all flaky cases across projects. Figure 6

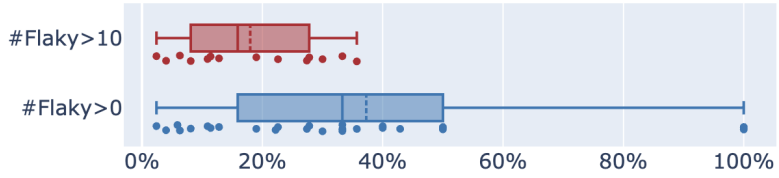


Figure 6: $\#FlakyVisual$ to $\#Flaky$ per project (%). $\#Flaky > 0$ and $\#Flaky > 10$ denote including only the projects with more than 0 (no threshold) and more than 10 flakiness instances. For $\#Flaky > 0$, mean = 37.2%, median = 33.3%; for $\#Flaky > 10$, mean = 17.9% and median=15.9%.

shows the distribution of the proportion of visual flaky commits among all flaky commits, with and without thresholding (i.e., including only projects with at least ten flaky commits). As shown in the lower boxplot of Figure 4, visual flakiness accounts for approximately 37.2% of all flaky cases, while in the thresholded subset (upper boxplot), 17.9% of all flaky commits are visually related across the remaining 14 projects. Combined with the previously established prevalence and relatively high likelihood of visual commits being flaky, these observations highlight the need for a deeper investigation of the underlying characteristics of web visual flakiness.

Answer to RQ1. Visual commits, which account for an average of 11.7% of total commits, are often more prone to flakiness than non-visual ones. With and without the threshold of at least 10 flaky commits per project to be considered, visual flaky commits constitute a notable portion of all flaky cases (37.2% and 17.9%). These results combined underscore the inherently unstable nature of the visual layer of web projects and motivate a detailed examination of its underlying characteristics.

3.2. ***RQ2: Common Types of Web Visual Flakiness***

Table 2 summarises the distribution of web visual flakiness in our dataset. Among the 262 analysed cases, 59.9% (157) are structure-related and 40.1% (105) are style-related. Similar trends appear in both Chromium (47.5% vs. 52.5%) and the 31 open-source web projects (70.1% vs. 29.9%). In the open-source projects, although style-related flakiness forms a smaller portion than in Chromium, it still accounts for nearly one-third of all visual flakiness, showing its practical importance. Overall, these results indicate that both structure- and style-related flakiness are major sources of web visual instability, with neither overwhelmingly dominating the other.

Table 2: Distributions of Web Visual Flakiness. The *Webs* refer to the open-source web projects. The values within the parentheses denote the percentage of flakiness instances in each type. The last column *Total* shows the combined statistics of *Webs* and *Chromium*.

Type	Subtype of Flaky Behaviour	# Flaky Cases (%)		
		Webs	Chromium	Total
Structure -related	DOM Structure Issues	34 (23.6%)	25 (21.2%)	59 (22.5%)
	DOM Layout Instability	17 (11.8%)	19 (16.1%)	36 (13.7%)
	Timing-sensitive Visual Triggers	50 (34.7%)	12 (10.2%)	62 (23.7%)
	All structured-related	101 (70.1%)	56 (47.5%)	157 (59.9%)
Style -related	Style Interpretation Issues	35 (24.3%)	45 (38.1%)	80 (30.5%)
	Style Resource Latency	8 (5.6%)	17 (14.4%)	25 (9.5%)
	All style-related	43 (29.9%)	62 (52.5%)	105 (40.1%)
All	-	144 (100%)	118 (100%)	262 (100%)

The following section presents the detailed distribution of structure- and style-related flakiness and provides real-world examples illustrating how each type of web visual flakiness manifests in practice.

3.2.1. ***Structure-related Flakiness***

Within structure-related flakiness (the last column in Table 2), the distribution is largely consistent across Chromium and the Web projects, except for Timing-sensitive Visual Triggers, which show the most notable variation—34.7% in the Web set but only 10.2% in Chromium. This gap may reflect the differences in execution environments. Web applications often rely on asynchronous updates or client-side scripts that are more directly affected by subtle variations in timing, whereas the rendering pipeline of Chromium is more centralised and internally synchronised. In contrast, DOM Structural Issues and DOM Layout Instability exhibit similar proportions across both sets, indicating that these types of structural flakiness (i.e., the presence, appearance, and position of elements) occur at comparable rates in Chromium and general web projects. The following examples illustrate how each type of structure-related flakiness manifests in practice.

DOM Structure Issues (22.5%) DOM structure issues refer to cases where the expected DOM hierarchy or elements are inconsistently constructed across executions. This can happen due to structural disruptions (e.g., element insertions or removals) or timing-related factors (e.g., asynchronous

rendering or race conditions) that affect *whether* they are properly built.

```
  <div class="mat-mdc-snack-bar-label" #label>
     /* fix:  added aria-hidden, aria-live */
-    <ng-template cdkPortal></ng-template>
+    <div aria-hidden="true">
+      <ng-template cdkPortal></ng-template>
+    </div>
+    <div [attr.aria-live]="_live"></div>
  </div>
```

Listing 1: Displaying and using elements dynamically

Listing 1[7] presents an example where rendering flakiness occurred in dynamically inserted content (i.e., a snack bar). The issue arose because the DOM structure lacked a properly defined `aria-live` region, causing unstable update behaviour during content rendering. Lines 4-7 add a `div` with `aria-hidden="true"` (Lines 4–6) and another `div` marked with `aria-live` (Line 7), ensuring a stable and complete DOM structure that prevents premature/inconsistent updates during dynamic rendering.

DOM Layout Instability (13.7%) DOM layout instability concerns flakiness affecting *how* existing DOM elements are spatially positioned, rather than whether they are present. Such flakiness manifests as inconsistent spatial arrangements, leading to overlaps, misalignment, or displacement of elements.

```
/* fix:  narrow viewport width flaky errors */
-  <div #table style="margin: 16px">
+  <div #table style="margin: 16px; max-width:
   90vw; max-height: 90vh;">
     <mat-table [dataSource]="dataSource">
        <ng-container matColumnDef="before">
        </ng-container>
      </mat-table>
   </div>
```

Listing 2: Container size makes the page structure unstable

[7]https://github.com/angular/components/commit/cf8e8eee

Listing 2[8] shows a case of DOM layout instability due to uncontrolled expansion of a container `<div>`. The absence of maximum width and height constraints allowed the table to grow unpredictably, causing structural shifts such as overflow or misalignment. The fix directly modifies the DOM by adding bounding attributes, which constrain spatial behaviour and stabilise rendering. Though expressed via style-like properties, the impact is structural, preventing disordered layouts rather than adjusting visual appearance.

Timing-sensitive Visual Triggers (23.7%) Timing-sensitive visual triggers refer to flakiness that manifests due to variation in *when* the visual state or DOM becomes observable or stable, rather than *whether* it appears at all. Such flakiness arise from timing mismatches between test actions and the completion of UI rendering or state updates.

```
  <input #textInput type="text" [value]="text"
    (focus)="onAcceptEdit()"
    (blur)="onAcceptEdit()"
  /input>
  export class TextEditorComponent {
    @ViewChild("textInput")
    ngAfterViewInit() {
     /* fix:  add delay to ensure the state */
+      window.setTimeout(() => {
        this.textInput.nativeElement.focus();
+      }, 5);
  }}
```

Listing 3: Unexpected element focus behaviour

In Listing 3[9], DOM Structure flakiness occurs due to timing mismatches in when the input field becomes available for interaction. The input field is expected to receive focus when triggered, as indicated by the focus operation ("onAcceptEdit") (Line 2). However, since rendering is not always synchronised with when elements become fully available, focus may be applied too early. This can cause the input field to fail to receive focus, preventing immediate text input intermittently.

[8]https://github.com/angular/components/commit/d2b499da
[9]https://github.com/angular/components/commit/4ae63b39

3.2.2. ***Style-related Flakiness***

For style-related flakiness, Style Interpretation Issues are the most prevalent type in both Chromium (38.1%) and the Web projects (24.3%) (Table 2). Style Resource Latency appears less frequently, though the magnitude of difference varies: it accounts for 14.4% of style-related flakiness in Chromium but only 5.6% in the Web projects. This difference likely stems from Chromium's heavy reliance on diverse and externally hosted style resources, which increases the likelihood of delayed or inconsistent loading, potentially leading to more visual instability.

Style Interpretation Issues (30.5%) This type of flakiness often arises when CSS properties or style interactions are handled inconsistently across rendering environments. Even minor differences in CSS or interactions between style rules—such as differences in the evaluation of properties like display or text-indent between platforms or browsers —can result in visually different outcomes across executions. Such inconsistencies make it challenging for developers to ensure a stable and uniform visual presentation in web applications.

```
  <style>
    input::-webkit-input-placeholder, textarea
   ::-webkit-input-placeholder {
      color: #aaaaaa;
      text-indent: 0;}
    /* fix:  add rule improve compatibility */
+   textarea::-webkit-input-placeholder {
+     color: #aaaaaa;
+     text-indent: -3px;}
  </style>
```

Listing 4: *text-indent* properties cause flakiness behaviour

Listing 4[10] (i.e., style codes) presents an example of style-related flakiness caused by CSS attribute compatibility issues. The placeholder text and cursor become misaligned when the text area is focused due to variations in how browsers interpret the text-indent property for placeholders. This inconsistency arises from differences in CSS rule enforcement across rendering engines. To improve compatibility, the fix explicitly defines *text-indent*

[10]https://github.com/ionic-team/ionic-framework/commit/76ca4757

for `textarea::-webkit-input-placeholder`, ensuring more consistent behaviour across browsers.

Style Resource Latency (9.5%) Style Resource Latency occurs when essential assets — such as fonts, images, or external style sheets — are not fully loaded before the visual rendering is complete. Such delays can lead to temporary or inconsistent appearance, as the web page may be incorrectly styled or displayed without its intended formatting.

```
1  main_resource.Write(HTML(
2   <!doctype html>
3   <head>
4    /* fix:  cache aware font Loading */
5    <link rel="preload" as="font" type="font/
   woff2"
6          href="https://example.com/font.woff
   ">))
```

Listing 5: Font loading contributes to flakiness

Listing 5[11] illustrates a case where flakiness arises due to delayed font loading (Line 6). If custom fonts take too long to load or fail to load, browsers fall back to default fonts, which may differ in size or spacing, causing misaligned or invisible elements. This mismatch between the expected and the actual leads to visual inconsistencies. Developers fixed this issue using preload hints to ensure timely font availability and prevent such flakiness.

Overall, Chromium shows a more even distribution of structure- and style-related flakiness compared to the Web projects. We suspect that the key contributor is the smaller proportion of Timing-sensitive Visual Triggers flakiness, possibly reflecting Chromium's more synchronised event scheduling. At the same time, its complex, multi-threaded rendering pipeline may introduce additional nondeterminism during layout computation and resource loading, contributing to the slightly higher proportions of DOM Layout Instability and Style Resource Latency observed. DOM construction remains largely deterministic within a single engine build. Hence, the modest increase in Style Interpretation Issues may instead reflect the broader redistribution of flaky cases from the decrease in timing-sensitive flakiness, rather than inherent nondeterminism in the style engine itself.

[11] https://issues.chromium.org/issues/40114104

Flaky behaviours across different front-end frameworks.

Modern web development increasingly relies on front-end frameworks. In this empirical study, we additionally examined whether the choice of framework influences the distribution of visual flakiness types. Among 31 web projects, 87 of 144 cases (60.6%) involved frameworks, such as React, Vue, and Angular[12]. As shown in Figure 7, similar patterns emerge across React, Vue, and Angular, suggesting that the underlying framework does not have much influence on the manifestation of web visual flakiness.

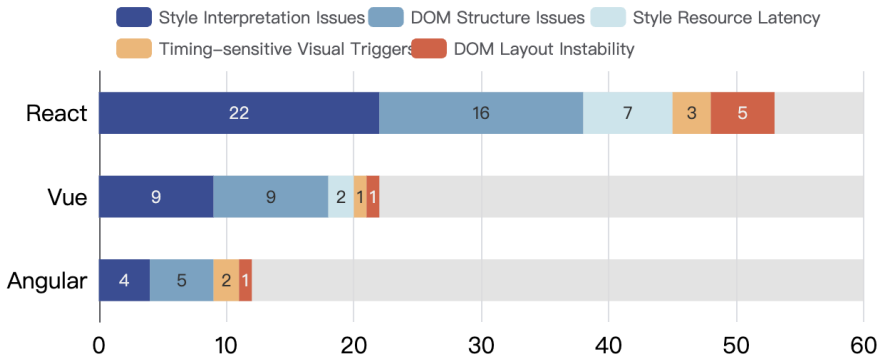


Figure 7: The distribution of flakiness across UI frameworks.

[12]The remaining 39.4% of projects do not use front-end frameworks; some were developed before frameworks became widely adopted, while others are lightweight applications or static websites that avoid the extra complexity that frameworks introduce.

Answer to RQ2. The analysis shows that structure-related and style-related types together account for nearly equal portions of web visual flakiness. Among the structure-related types, Timing-sensitive visual triggers are the most frequent, whereas Style Interpretation Issues are the most common among style-related ones, followed by Style Resource Latency. Overall, both Chromium and the Web projects exhibit similar distributions of visual flakiness types. However, Chromium exhibits a lower proportion of Timing-sensitive visual triggers and a comparatively higher share of style-related flakiness. The smaller ratio of timing-sensitive triggers is likely attributed to more synchronised and centralised rendering and event pipeline of Chromium, reducing timing variability across executions. In contrast, the relatively higher ratio of style-related flakiness might partly result from fewer structure-related cases overall, rather than an inherent increase in style-level instability.

3.3. ***RQ3: Common Flakiness Fix Patterns of Developers***

Based on the categorisation process described in Section 2.2.4, we identified seven categories of recurring fix strategies developers use to address web visual flakiness.

- **Manage element changes**: insert, remove, or update DOM elements to prevent inconsistencies in element presence or structure.
- **Delay**: introduce a conservative timing buffer to handle variability in dynamic rendering or state transitions, ensuring that subsequent operations are not processed prematurely.
- **Explicit size position**: define layout dimensions (e.g., width, height, margins) to avoid instability caused by flexible or implicit sizing.
- **Use grid/flexbox**: employ structured layout systems such as CSS Grid or Flexbox to improve spatial consistency and reduce layout misalignments.
- **CSS compatibility**: modify CSS rules to ensure consistent behaviour across different browsers and rendering engines by applying baseline styles that override browser-specific defaults to standardise the initial rendering behaviour across environments
- **Element-event coordination**: align target elements and triggering events in timing and state to prevent missed or inconsistent interactions.

- **Preload & Cache**: ensure key resources (e.g., fonts, images) are available when needed by preloading them and leveraging caching to prevent delays or rendering failures.

We further analysed how these fix strategies are distributed across different types of web visual flakiness to understand which strategies developers often employ to handle structure- and style-related flakiness issues, as shown in Table 3. Overall, developers tend to employ fix strategies that closely correspond to how each type of web visual flakiness manifests. For instance, in structure-related flakiness, such as DOM Structure issues, developers most frequently applied Element–event coordination, used in 23 of 34 cases, to stabilise interactions between elements and events. In Timing-sensitive Visual Triggers, Delay and Element–Event Coordination were the two dominant strategies, both addressing asynchronous rendering and event ordering issues. For DOM Layout Instability, developers often constrain layout flexibility to prevent unintended spatial shifts by adopting the Explicit size position strategy. For the Style Interpretation, nearly all fixes (33 of 35) were about CSS Compatibility, addressing inconsistent rendering semantics through CSS adjustments. Lastly, for Style Resource Latency, timing- and loading-related strategies, such as Delay and Preload & Cache, were most frequently applied to ensure stable resource availability and rendering.

At the same time, we observed that specific fix strategies are not strictly confined to a single type of web visual flakiness. For instance, CSS Compatibility was applied across all five categories of flakiness, and Delay was utilised in DOM Layout Instability, Timing-sensitive Visual Triggers, Style Interpretation Issues, and Style Resource Latency. Element-event coordination, the most adopted strategy in DOM Structure Issues, was also the most common strategy among all fixes for structure-related flakiness, appearing in 38 of 144 cases (26.4%). We conjecture that this broad applicability stems from the general role these strategies play in stabilising cross-cutting sources of nondeterminism. For instance, modifying CSS rules for CSS Compatibility could mitigate rendering inconsistencies that manifest across structure and style components. Similarly, the Delay strategy reduces premature or mismatched interactions by introducing timing buffers, while Element–event coordination can ensure the synchronisation of elements and their triggering events, thus generally preventing state mismatches in timing-sensitive scenarios.

Developer priorities for different types.

Table 3: Repair strategies for 144 cases of web projects' visual flakiness. Overall, developers typically apply fix strategies aligned with specific visual flakiness types (e.g., *Element–Event Coordination* for *DOM Structure Issues*, and *Delays* for *Timing-sensitive Visual Triggers*), while some strategies—such as *CSS compatibility*—are used across both structure- and style-related flakiness.

Type	Cause	Fix method	# Flaky Cases (%) Webs
Structure -related	DOM Structure Issues	CSS compatibility	3 (2.1%)
		Manage element changes	8 (5.6%)
		Element-event coordination	23 (16.0%)
	DOM Layout Instability	CSS compatibility	2 (1.4%)
		Delay	2 (1.4%)
		Manage element changes	1 (0.7%)
		Element-event coordination	1 (0.7%)
		Explicit size position	5 (3.5%)
		Use grid/flexbox	6 (4.2%)
	Timing-sensitive Visual Triggers	CSS compatibility	3 (2.1%)
		Delay	26 (18.1%)
		Element–event coordination	14 (9.7%)
		Manage element changes	7 (4.9%)
Style -related	Style Interpretation Issues	CSS compatibility	33 (22.9%)
		Delay	1 (0.7%)
		Manage element changes	1 (0.7%)
	Style Resource Latency	CSS compatibility	1 (0.7%)
		Delay	2 (1.4%)
		Preload & Cache	4 (2.8%)
		Manage element changes	1 (0.7%)

We further examined how developers prioritise addressing different types of web visual flakiness using the Chromium dataset, where each issue is assigned an explicit priority level. Figure 8 shows the distribution of these priorities, P1 (high), P2 (medium), and P3 (low), across structure-related and style-related categories of web visual flakiness. Specifically, the first and third subfigures in Figure 5 illustrate priority distributions within style-related and structure-related flakiness, respectively. Overall, while a substan-

tial proportion of both structure-related (44.6%) and style-related (59.7%) issues were assigned medium priority (P2), the structure-related issues were more frequently labelled as low priority (P3) than the style-related (41.1% vs. 24.2%). This implies that developers may allocate relatively more attention to style-related flakiness in Chromium. A likely explanation is that style-related flakiness often produces visually broader or more noticeable disruptions, such as broken layouts or inconsistent rendering, compared to the structure-related ones that may remain localised to specific elements. In fact, our previous analysis on the distribution of web visual flakiness shows that style-related flakiness represents a larger overall share than in the open-source web projects.

At a finer granularity, DOM Layout Instability exhibited the highest proportion of high-priority (P1) assignments (21.2%), followed by Style Resource Latency (17.6%) and Style Interpretation Issues (15.6%). These results further support the above conjectures, suggesting that developers often prioritise unstable behaviours with broad visual impact, especially those that affect page layout or rendering consistency. Timing-Sensitive Visual Triggers were mostly given medium priority (83.3%), indicating a medium level of urgency. DOM Structure Issues had the highest proportion of low-priority (P3) assignments (56%), suggesting that element-specific inconsistencies are often viewed as less critical to the overall user experience. This further explains the higher proportion of low priority in structure-related flakiness compared to style-related flakiness.

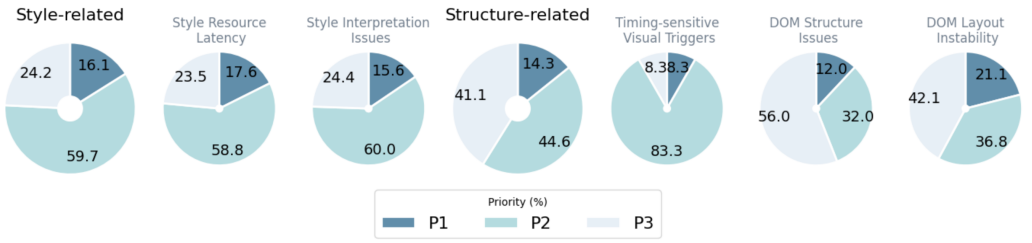


Figure 8: The distribution of developers addressing priorities (P1 (high), P2 (medium), and P3 (low)) under different causes, the proportions are in percentages (%).

Answer to RQ3. While developers tend to choose fix strategies that align closely with the specific types of web visual flakiness, some fix strategies appear across multiple types, reflecting their broad applicability. Additional analysis of issue priorities shows that developers tend to prioritise flakiness with broader visual impact, such as layout or style-related problems, over those with narrower or isolated effects.

4. Discussion

4.1. Positioning web visual flakiness against general UI issues and previous Web flakiness studies

Unlike general web visual issues discussed in prior studies Althomali et al. (2021); Alameer and Halfond (2016); Mahajan et al. (2016); Romano et al. (2021), which are typically deterministic or consistently reproducible, web visual flakiness is inherently intermittent and non-deterministic. It often stems from asynchronous rendering, transient DOM states, or timing inconsistencies, and manifests across the structure and style dimensions that we categorised in this study. Such characteristics of these issues complicate their detection and resolution, as they depend on subtle runtime interactions rather than static or deterministic failures. Prior studies on Web flakiness have primarily examined broad or functional causes (e.g., network latency, environment configuration) or deterministic rendering faults Romano et al. (2021). In contrast, our work isolates and empirically characterises flaky behaviours that specifically originate in the visual layer, i.e., those directly tied to the presentation logic of web structures and styles, thereby addressing a class of issues that has often been overlooked in prior work.

5. Threats to Validity

5.1. Internal validity

While our categorisations of flaky behaviours were performed through iterative discussion and cross-validation among the two authors, along with LLM assistance, the process is inherently affected by subjective judgment. Although we ensured consistency through shared criteria and multiple passes over the data, potential bias or misclassification may remain. Flakiness in web visuals can stem from subtle timing issues, rendering inconsistencies, style misinterpretations, etc. While we applied multiple-round manual

checks, some edge cases may still have been misplaced. In addition, our categorisation may not capture all manifestations in diverse contexts. Future work could refine and expand these criteria for a more precise.

5.2. Construct validity

Construct validity threats primarily stem from the use of keyword-based filtering of commit messages, issue reports, and pull requests to identify visual commits and web visual flakiness. Keyword matching may introduce misclassification, including false negatives when visual or flaky behaviour is only implicitly described, as well as false positives when keywords do not correspond to actual visual or flaky behaviour.

To mitigate this, we adopted a conservative sampling strategy focused on precision. Specifically, we focused only on the interaction of flaky and visual keywords, manually validating each candidate case. Two authors jointly inspected code changes and artifacts to confirm the presence of visual flakiness, and thereby further eliminated false positives, increasing confidence in the final dataset.

As this conservative choice likely results in false negatives, the reported prevalence should be interpreted as a lower bound. However, since our categorisation and fix-pattern analyses focus on recurring behavioural characteristics observed across consistently validated cases, rather than on complete enumeration, this limitation is less likely to affect the qualitative findings.

5.3. External validity

A potential limitation of our study is the generalisability of the results beyond the specific projects and test suites analysed. Although our findings are rooted in web projects and Chromium, they may also be relevant to other contexts, such as mobile applications and hybrid frameworks. Mobile browsers, which exhibit a variety in hardware setups and screen sizes, are also prone to similar structural and style inconsistencies, especially when it comes to responsive development. Despite differences in technical implementation, manifestations of flakiness, like layout shifts and inconsistent style application, remain conceptually aligned. This suggests that our categorisations and observed fix patterns may help inform broader efforts to address visual flakiness across diverse platforms.

6. Related work

Flaky tests have been the subject of extensive research in the software testing community, particularly in programming languages, such as Java, Python, and JavaScript Luo et al. (2014); Lam et al. (2020a,b); Gruber et al. (2021); Hashemi et al. (2022); Leong et al. (2019). Numerous studies have examined the prevalence, causes, and mitigation strategies for flaky tests in code-based systems Lam et al. (2020a); Pinto et al. (2020); Dutta et al. (2020). For example, Luo et al. conducted a pioneering study on flaky tests, analysing large-scale systems to identify the underlying causes of test flakiness Luo et al. (2014). While these findings are valuable for traditional software, they do not directly address the challenges posed by style-related issues, where visual rendering discrepancies could also lead to flakiness.

For research on flaky tests in web environments, Hashemi et al. analysed flaky behaviour in JavaScript frameworks, identifying browser-related issues as significant contributors to flakiness testing Hashemi et al. (2022). Romano et al. analysed flaky UI tests from 62 web and Android projects and identified four categories of causes for flaky UI tests Romano et al. (2021). Moran et al. focused on identifying the root cause of flakiness in web applications by analysing the execution of tests under diverse combinations of environmental factors that may contribute to flakiness Morán et al. (2020). Pei et al. investigated the async await flakiness in web tests, focusing on the DOM and time issues Pei et al. (2024). However, these studies rarely isolate style-related code as a distinct source of flaky behaviour, although CSS is highly susceptible to the inconsistencies that lead to flakiness. Our study aims to fill this gap by specifically investigating flaky behaviours in CSS, providing a more nuanced understanding of the problem in web development.

Besides empirical studies on flaky tests, there has been considerable work on mitigation strategies Ziftci and Cavalcanti (2020); Camara et al. (2021); Haben et al. (2021); Rahman and Shi (2024); Fallahzadeh and Rigby (2022). Bell et al. leveraged code evolution and code coverage to determine whether new test failures between two commits are due to flaky tests Bell et al. (2018). There have also been several tools that have been targeted to detect or repair certain types of flaky tests, such as iDFlakies Lam et al. (2019b), iFixFlakies Shi et al. (2019), FLASH Dutta et al. (2020), Flakify Fatima et al. (2022), Flakeflagger Alshammari et al. (2021), and so on Dutta et al. (2021); Cordy et al. (2022); Cai et al. (2024).

While there is a growing of research on flaky tests in software systems,

the focus on web visual-related flakiness is minimal. Our research builds on previous studies of flaky tests by extending the analysis to the web visuals. By focusing specifically on flaky behaviour in style and visual layers, this study aims to bridge the gap in current literature and provide actionable insights for developers dealing with flaky behaviours.

7. Conclusion

This paper presents a focused empirical study of web visual flakiness as a distinct and explicitly characterised class of flakiness, based on 262 cases collected from 31 open-source projects (144 cases) and the Chromium project (118 cases), revealing its recurring presence across diverse modern web projects. We found that visual commits, comprising 11.4% of all commits, exhibit a higher frequency of flakiness than non-visual commits. We categorised web visual flakiness into two primary dimensions–structure-related and style-related–, which we further refined into three and two concrete subtypes, respectively. Through an inductive analysis of developers' fixes, we found seven recurring fix strategies, which address common instability patterns, such as element-event coordination, timing, layout stabilisation, and style consistency. Although some are closely associated with specific flakiness types, others span across multiple types, showing the intertwined nature of structure and style aspects in rendering. Together, these findings highlight the need to treat web visual flakiness as a distinct subject in web development and testing. To facilitate further research and reproducibility, we make our dataset and replication package publicly available at: `https://doi.org/10.5281/zenodo.17324581`

8. Acknowledgments

This work is supported by the Luxembourg National Research Funds (FNR) through the CORE project grant C20/IS/14761415/TestFlakes and partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2021R1A5A1021944).

References

Alameer, A., Halfond, W.G., 2016. An empirical study of internationalization failures in the web, in: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 88–98.

Alshammari, A., Morris, C., Hilton, M., Bell, J., 2021. Flakeflagger: Predicting flakiness without rerunning tests, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE. pp. 1572–1584.

Althomali, I., Kapfhammer, G.M., McMinn, P., 2021. Automated visual classification of dom-based presentation failure reports for responsive web pages. Software Testing, Verification and Reliability 31, e1756.

Bell, J., Legunsen, O., Hilton, M., Eloussi, L., Yung, T., Marinov, D., 2018. Deflaker: Automatically detecting flaky tests, in: Proceedings of the 40th international conference on software engineering (ICSE), pp. 433–444.

Cai, X., Dong, Z., Wang, Y., Tiwari, A., Peng, X., 2024. Reproducing timing-dependent gui flaky tests in android apps via a single event delay, in: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp. 1504–1515.

Camara, B., Silva, M., Endo, A., Vergilio, S., 2021. On the use of test smells for prediction of flaky tests, in: Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing (SAST), pp. 46–54.

Choudhary, S.R., Versee, H., Orso, A., 2010. Webdiff: Automated identification of cross-browser issues in web applications, in: 2010 IEEE International Conference on Software Maintenance, IEEE. pp. 1–10.

Choudhary, S.R., Zhao, D., Versee, H., Orso, A., 2011. Water: Web application test repair, in: Proceedings of the First International Workshop on End-to-End Test Script Engineering, pp. 24–29.

Cordy, M., Rwemalika, R., Franci, A., Papadakis, M., Harman, M., 2022. Flakime: Laboratory-controlled test flakiness impact assessment, in: 2021 IEEE/ACM 44rd International Conference on Software Engineering (ICSE), IEEE.

Dutta, S., Shi, A., Choudhary, R., Zhang, Z., Jain, A., Misailovic, S., 2020. Detecting flaky tests in probabilistic and machine learning applications, in: Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis (ISSTA), pp. 211–224.

Dutta, S., Shi, A., Misailovic, S., 2021. Flex: fixing flaky tests in machine learning projects by updating assertion bounds, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 603–614.

Eck, M., Palomba, F., Castelluccio, M., Bacchelli, A., 2019. Understanding flaky tests: The developer's perspective, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 830–840.

Fallahzadeh, E., Rigby, P.C., 2022. The impact of flaky tests on historical test prioritization on chrome, in: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 273–282.

Fatima, S., Ghaleb, T.A., Briand, L., 2022. Flakify: A black-box, language model-based predictor for flaky tests. IEEE Transactions on Software Engineering (TSE) 49, 1912–1927.

Gruber, M., Fraser, G., 2022. A survey on how test flakiness affects developers and what support they need to address it, in: 2022 IEEE Conference on Software Testing, Verification and Validation (ICST), IEEE. pp. 82–92.

Gruber, M., Lukasczyk, S., Kroiß, F., Fraser, G., 2021. An empirical study of flaky tests in python, in: 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), IEEE. pp. 148–158.

Habchi, S., Haben, G., Papadakis, M., Cordy, M., Le Traon, Y., 2022. A qualitative study on the sources, impacts, and mitigation strategies of flaky tests, in: 2022 IEEE Conference on Software Testing, Verification and Validation (ICST), IEEE. pp. 244–255.

Haben, G., Habchi, S., Micco, J., Harman, M., Papadakis, M., Cordy, M., Le Traon, Y., 2024. The importance of accounting for execution failures when predicting test flakiness, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1979–1989.

Haben, G., Habchi, S., Papadakis, M., Cordy, M., Le Traon, Y., 2021. A replication study on the usability of code vocabulary in predicting flaky tests, in: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), IEEE. pp. 219–229.

Hashemi, N., Tahir, A., Rasheed, S., 2022. An empirical study of flaky tests in javascript, in: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 24–34.

Lam, W., Godefroid, P., Nath, S., Santhiar, A., Thummalapenta, S., 2019a. Root Causing Flaky Tests in a Large-Scale Industrial Setting, in: Proceedings ofthe 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp. 101–111. doi:10.1145/3293882.3330570.

Lam, W., Muşlu, K., Sajnani, H., Thummalapenta, S., 2020a. A study on the lifecycle of flaky tests, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE), pp. 1471–1482.

Lam, W., Oei, R., Shi, A., Marinov, D., Xie, T., 2019b. IDFlakies: A framework for detecting and partially classifying flaky tests. Proceedings - 2019 IEEE 12th International Conference on Software Testing, Verification and Validation (ICST) , 312–322doi:10.1109/ICST.2019.00038.

Lam, W., Winter, S., Astorga, A., Stodden, V., Marinov, D., 2020b. Understanding reproducibility and characteristics of flaky tests through test reruns in java projects, in: 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), IEEE. pp. 403–413.

Leong, C., Singh, A., Papadakis, M., Le Traon, Y., Micco, J., 2019. Assessing transition-based test selection algorithms at google, in: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), IEEE. pp. 101–110.

Luo, Q., Hariri, F., Eloussi, L., Marinov, D., 2014. An empirical analysis of flaky tests, in: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering (FSE), pp. 643–653.

Mahajan, S., Gadde, K.B., Pasala, A., Halfond, W.G., 2016. Detecting and localizing visual inconsistencies in web applications, in: 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), IEEE. pp. 361–364.

Morán, J., Augusto, C., Bertolino, A., De La Riva, C., Tuya, J., 2020. Flakyloc: flakiness localization for reliable test suites in web applications. Journal of Web Engineering 19, 267–296.

Nass, M., Alégroth, E., Feldt, R., Coppola, R., 2023. Robust web element identification for evolving applications by considering visual overlaps, in: IEEE Conference on Software Testing, Verification and Validation, ICST 2023, Dublin, Ireland, April 16-20, 2023, IEEE. pp. 258–268. URL: https://doi.org/10.1109/ICST57152.2023.00032, doi:10.1109/ICST57152.2023.00032.

Parry, O., Kapfhammer, G.M., Hilton, M., McMinn, P., 2021. A survey of flaky tests. ACM Transactions on Software Engineering and Methodology (TOSEM) 31, 1–74.

Parry, O., Kapfhammer, G.M., Hilton, M., McMinn, P., 2022. Surveying the developer experience of flaky tests, in: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 253–262.

Pei, Y., Sohn, J., Habchi, S., Papadakis, M., 2024. Non-flaky and nearly-optimal time-based treatment of asynchronous wait web tests. ACM Transactions on Software Engineering and Methodology (TOSEM) .

Pei, Y., Sohn, J., Papadakis, M., 2025. An empirical study of web flaky tests: Understanding and unveiling dom event interaction challenges, in: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST), IEEE. pp. 92–102.

Pinto, G., Miranda, B., Dissanayake, S., d'Amorim, M., Treude, C., Bertolino, A., 2020. What is the vocabulary of flaky tests?, in: Proceedings of the 17th International Conference on Mining Software Repositories (MSR), pp. 492–502.

Rahman, S., Shi, A., 2024. Flakesync: Automatically repairing async flaky tests, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE), pp. 1–12.

Romano, A., Song, Z., Grandhi, S., Yang, W., Wang, W., 2021. An empirical analysis of ui-based flaky tests, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE. pp. 1585–1597.

Rwemalika, R., Kintis, M., Papadakis, M., Traon, Y.L., Lorrach, P., 2019. On the evolution of keyword-driven test suites, in: 12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019, IEEE. pp. 335–345. URL: https://doi.org/10.1109/ICST.2019.00040, doi:10.1109/ICST.2019.00040.

Shi, A., Lam, W., Oei, R., Xie, T., Marinov, D., 2019. ifixflakies: A framework for automatically fixing order-dependent flaky tests, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 545–555.

Stocco, A., Yandrapally, R., Mesbah, A., 2018. 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.

Yeh, T., Chang, T.H., Miller, R.C., 2009. Sikuli: using gui screenshots for search and automation, in: Proceedings of the 22nd annual ACM symposium on User interface software and technology, pp. 183–192.

Ziftci, C., Cavalcanti, D., 2020. De-flake your tests: Automatically locating root causes of flaky tests in code at google, in: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 736–745.

Zou, Y., Chen, Z., Zheng, Y., Zhang, X., Gao, Z., 2014. Virtual DOM coverage for effective testing of dynamic web applications, in: Pasareanu, C.S., Marinov, D. (Eds.), International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014, ACM. pp. 60–70. URL: https://doi.org/10.1145/2610384.2610399, doi:10.1145/2610384.2610399.