Stabilizing and Automating Web UI Testing

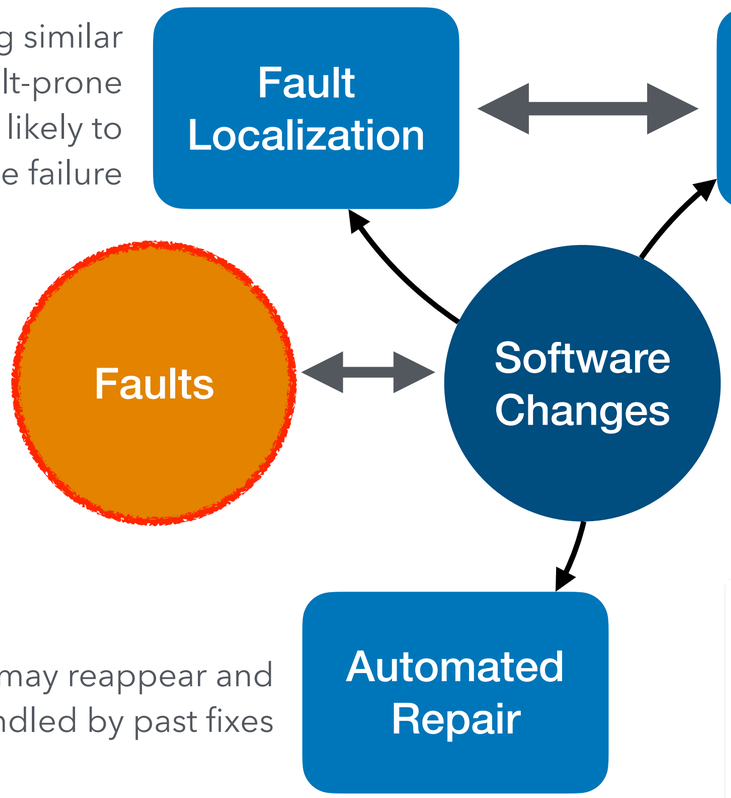
Presented by Jeongju Sohn

Software Evolution and Software Disaster

The dual relationship between Software Evolution and Software Disaster

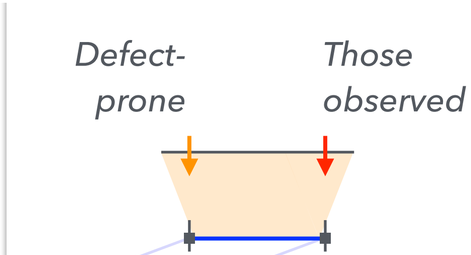
Changes on code elements suspicious in terms of recent faults are fault-prone

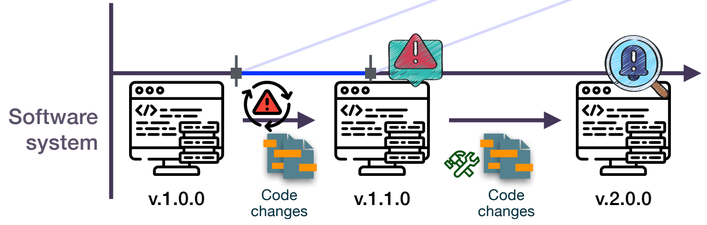
Code elements sharing similar characteristics with fault-prone code elements are more likely to be the cause of the failure

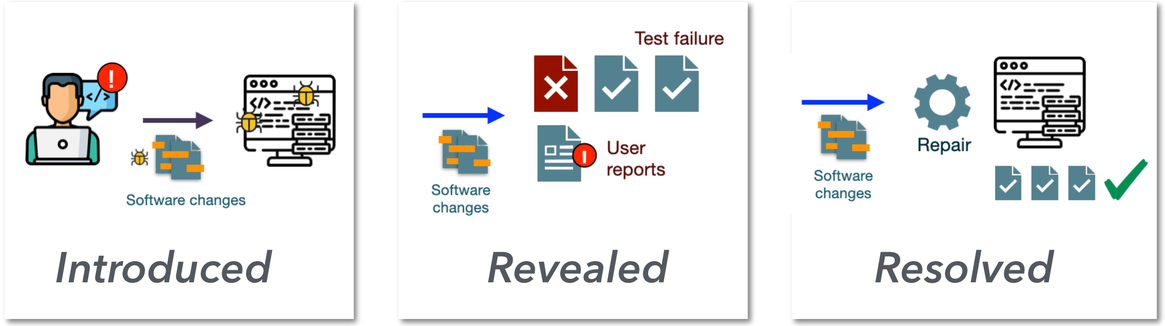


Similar faults may reappear and could be handled by past fixes

Faults:





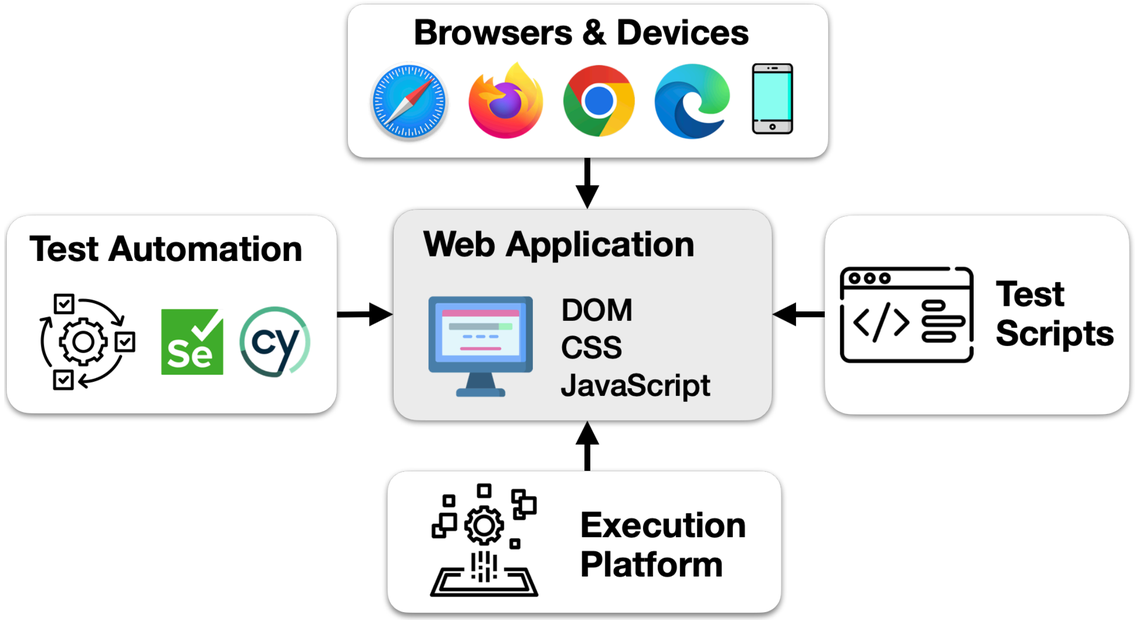


Challenges of Web UI Testing

Web UI Testing

- **Web UI testing** verifies that a web app's user interface *behaves* and *appears* as intended across different browsers, devices, and dynamic conditions, ensuring both *functional correctness and visual consistency.*

  - *Frequent breakages, dynamic content, browser inconsistencies.*
  - *Manual script maintenance & high cost of authoring tests.*



For reliable and efficient web UI testing

- **Stability (Flakiness)**: Ensuring tests are reliable despite dynamic UI changes.
- **Automation (NL Execution)**: Reducing manual effort by automatically interpreting and executing test instructions.

Flakiness in web UI testing and DOM event interactions

Flakiness in Web UI Testing

- Flakiness in web UI testing often arises from inconsistencies or timing issues in interactions between the events and DOM

*The Issue **begins** by clicking on the button to open the details menu.*

*The **flaky line** (line 80) try to locate the element ".qr-code-address" before it finishes rendering, leading to flaky failure*

```
it("should add same account addresses", async() {
  ...
75  await driver.clickElement(
76      '[data-testid="account-options-menu__account-details"]');
77  const detailsModal = await driver.findVisibleElement("span .modal");
78
79
80  const secondAccountAddress = await driver.findElement(".qr-code-address");
81  const secondAccountPublicAddress = await secondAccountAddress.getText();
82
83  await driver.clickElement(".account-modal__close");
84  await detailsModal.waitForElementState("hidden");
  ...
}
```

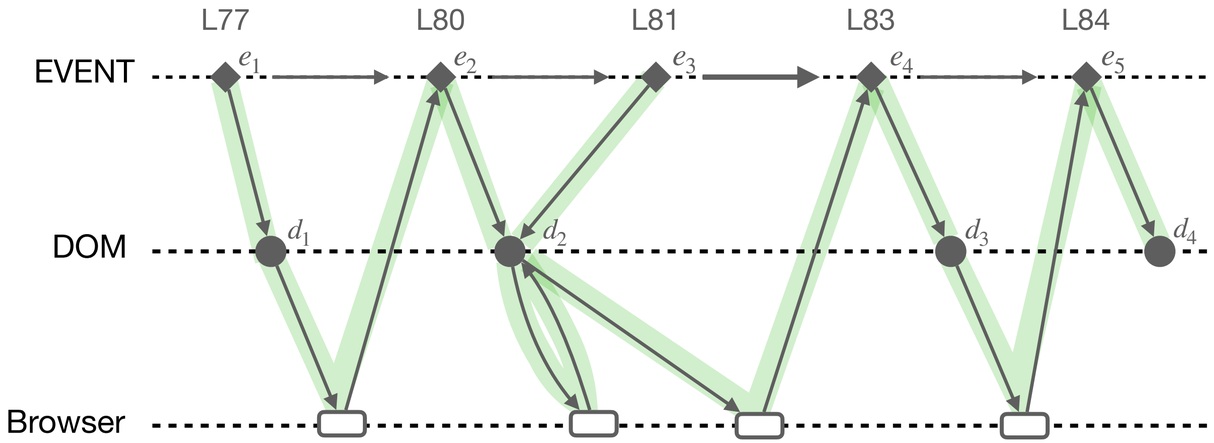
Add a waitFor statement to ensure the element is fully loaded before the interaction

An example of DOM event interaction flaky test case **[1]**

[1] https://github.com/oxgersa/metamask-extension/commit/a3d232ed

Flakiness in Web UI Testing

- Flakiness in web UI testing often arises from inconsistencies or timing issues in interactions between the events and DOM



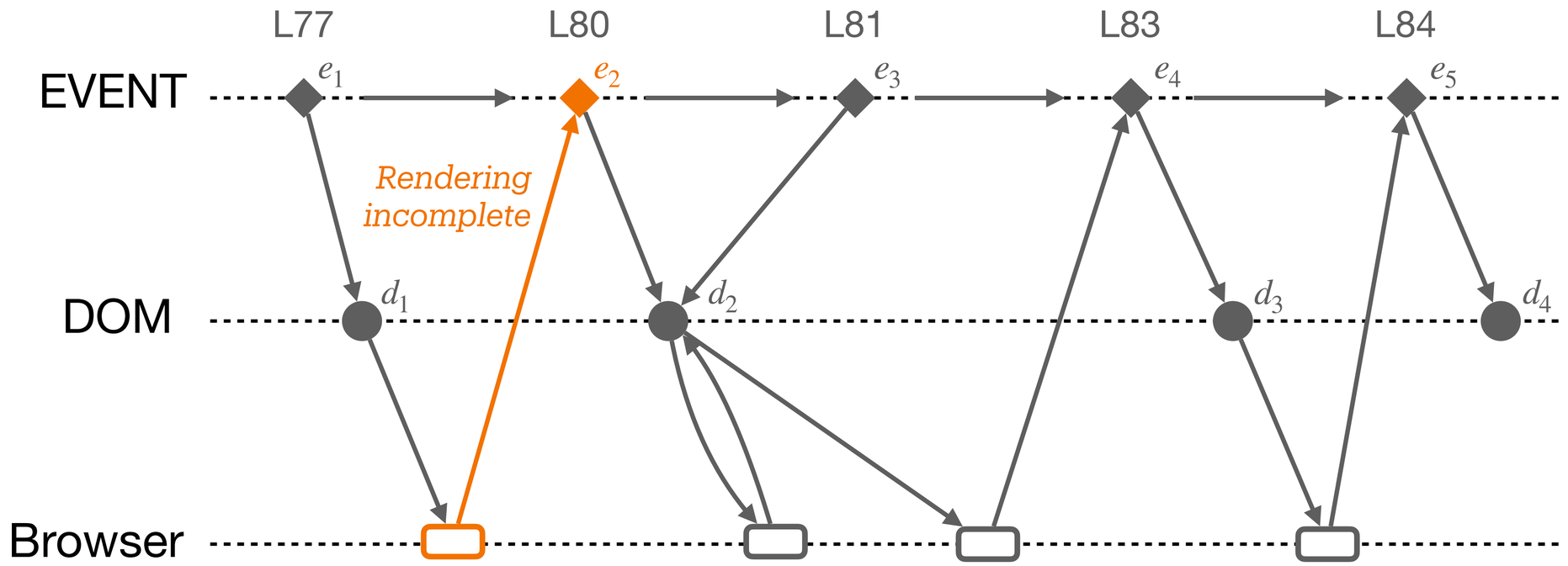
DOM event interaction model

```
    it("should add same account addresses", async() {
      ...
75    await driver.clickElement(
76        '[data-testid="account-options-menu__account-details"]');
77    const detailsModal = await driver.findVisibleElement("span .modal”);
78
79
80    const secondAccountAddress = await driver.findElement(“.qr-code-address");
81    const secondAccountPublicAddress = await secondAccountAddress.getText();
82
83    await driver.clickElement(“.account-modal__close");
84    await detailsModal.waitForElementState(“hidden");
      ...
    }
```

[1] https://github.com/oxgersa/metamask-extension/commit/a3d232ed

DOM Event Interaction Model

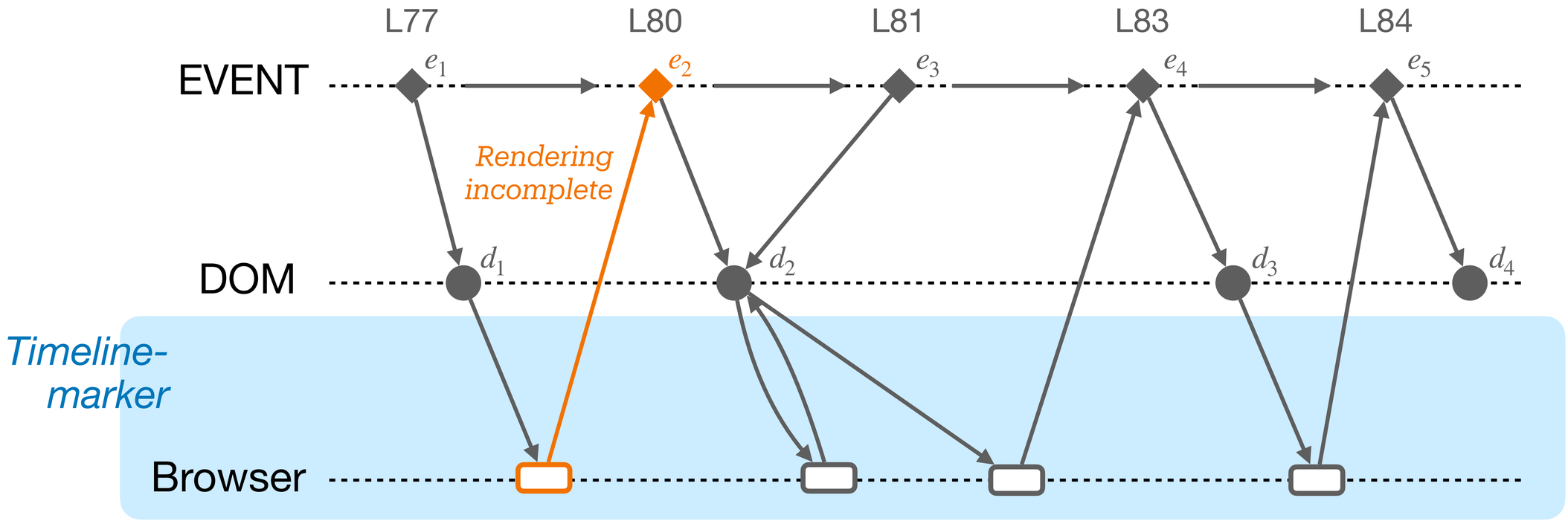
- **Goal**: analyse how DOM event interaction cause flakiness.



L80 | const secondAccountAddress = await driver.findElement(“.qr-code-address”);

DOM Event Interaction Model

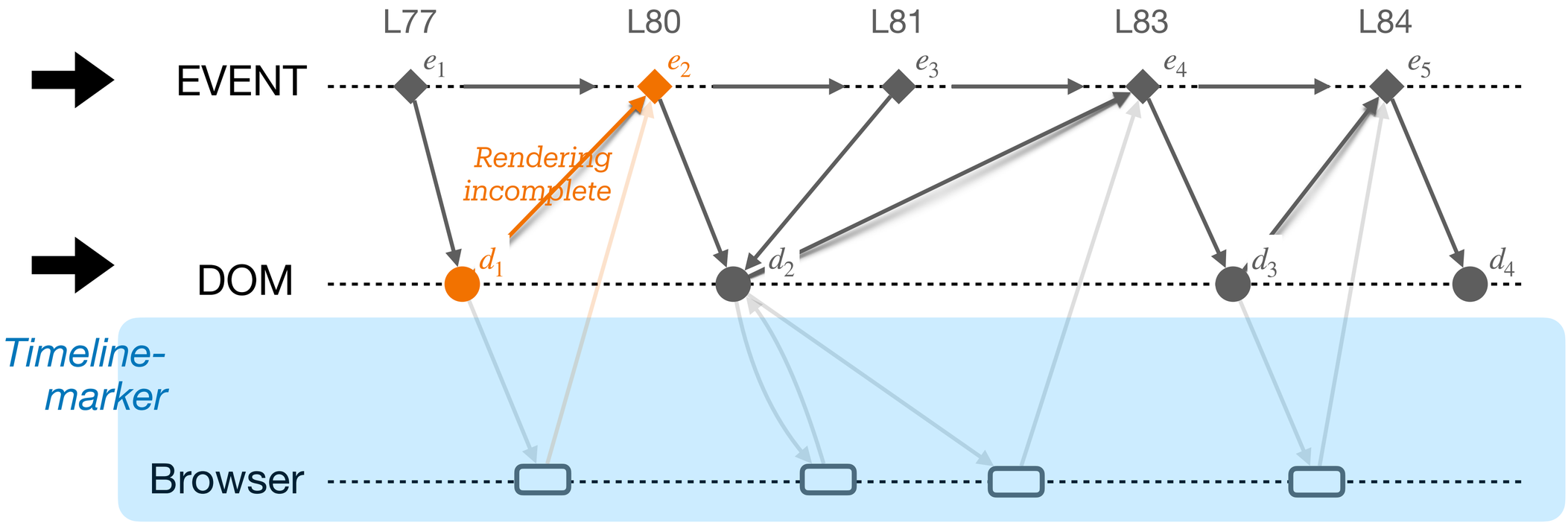
- **Goal**: analyse how DOM event interaction cause flakiness



L80 | const secondAccountAddress = await driver.findElement(“.qr-code-address”);

DOM Event Interaction Model

- **Goal**: analyse how DOM event interaction cause flakiness



L80 | const secondAccountAddress = await driver.findElement(“.qr-code-address”);

The impact of DOM event interactions

Statement categorisation

- Define **six new statement** categories based on their roles in DOM interactions and event flows:

Type	Definition	Example
T_{ED} (Event-DOM)	An event that explicitly or implicitly triggers querying or modification of DOM	await driver.clickElement({text: "Approve", tag: “button”}); await driver.hoverElement({ id: "menu" });
T_{E} (Event)	Contain (high-level) user or system events without DOM involvement	cy.visit(“/“); await scrollToBottom(mainViewer, page);
T_{DE} (DOM-Event)	Querying or modification of a DOM causes or triggers an event	await page.locator(editPublicLink).click();
T_{D} (DOM)	Querying or modification of a DOM without triggering events	const fileList = await getContentBySelector(".attach-name not(.attach-size)”);
T_{R} (Response)	Check, verify, or assert to the state of the DOM (i.e., response validation)	e2e.components.LoadingIndicator.icon().should(“Invisible”);
T_{O} (Other)	Operations that are not directly related to DOM manipulation or events	var getExplorationElements = function(explorationTitle);

The impact of DOM event interactions

Study Design.

- Manual analyses on 123 flaky tests from 49 open-source web projects (keyword-based filtering)
- **Four key factors**
  - **F1**—Current statement type (the type of interaction that is currently investigated for it being directly involved in the flaky behavior)
  - **F2**—Previous statement type
  - **F3**—Same DOM context (whether all elements referenced in the current statement are within **the same DOM context** (Y) or **span across different contexts** (e.g., main page + modal, iframe) (**N**)).
  - **F4**—Event interaction level (the scope at which the event is triggered: page level (P), element level (E), or both (PE)).

The impact of DOM event interactions

Example.

```
   it("should add same account addresses", async() {
     ...
75   await driver.clickElement(
76       '[data-testid="account-options-menu__account-details"]');
77   const detailsModal = await driver.findVisibleElement("span .modal”);
78
79
80   const secondAccountAddress = await driver.findElement(“.qr-code-address");
81   const secondAccountPublicAddress = await secondAccountAddress.getText();
82
83   await driver.clickElement(“.account-modal__close");
84   await detailsModal.waitForElementState(“hidden");
     ...
   }
```

#	Action (Simplified)	F1	F2	F3	F4
77	findVisibleElement('span .modal')	ED (find-element)	ED (click-element)	N	E
80	findElement('.qr-code__address')	ED (find-element)	ED (find-element)	Y	E
81	getText() on .qr-code__address	ED (find-element)	ED (find-element)	Y	E
83	clickElement('.account-modal__close')	ED (click-element)	ED (find-element)	Y	E
84	waitForElementState('hidden')	ED (wait-element)	ED (click-element)	Y	E

The impact of DOM event interactions

Current statement interaction type among Flaky and Non-flaky cases

F1 (# number (percentage))		
Type	**Flaky**	**Non-flaky**
T_{ED} (Event-DOM)	36 (29%)	38 (33%)
- T_E (Event)	18 (15%)	**34 (30%)**
T_{DE} (DOM-Event)	18 (15%)	20 (18%)
+ T_D (DOM)	**16 (13%)**	4 (4%)
+ T_R (Response)	**35 (28%)**	18 (15%)
T_O (Other)	-	-
Total	123 (100%)	114 (100%)

- T_D and T_R are more frequent in flaky cases: DOM operations are prone to instability and flakiness often manifest during result verification.
  - D operations directly depend on page state, making them more sensitive to rendering delays or missing elements
  - Developers often skip adding explicit wait/checks before operations
- T_E (i.e., event interactions alone) are less likely to cause flakiness
  - Doesn't fail unless the target is missing or not ready (less tied to dynamic content)— their fissures often stem form preceding DOM issues

The impact of DOM event interactions

Current statement interaction type among Flaky and Non-flaky cases

F1 (# number (percentage))		
Type	**Flaky**	**Non-flaky**
T_{ED} (Event-DOM)	36 (29%)	38 (33%)
- T_E (Event)	18 (15%)	**34 (30%)**
T_{DE} (DOM-Event)	18 (15%)	20 (18%)
+ T_D (DOM)	**16 (13%)**	4 (4%)
+ T_R (Response)	**35 (28%)**	18 (15%)
T_O (Other)	-	-
Total	123 (100%)	114 (100%)

- ED/DE interactions (T_{ED} and T_{DE}): flakiness correlates with event-DOM coupling:
  - Tightly coupled (wait-element) → stable (non-flaky)
  - Moderately coupled (click-element) → balanced
  - Loosely coupled (find-element, hover-element) → more flaky

The impact of DOM event interactions

DOM event interaction combinations among Flaky and Non-flaky cases

	Flaky		Example	Non-flaky		Example
Top	F2:F1	#	Statement detail	F2:F1	#	Statement detail
1	T_{ED}:T_{ED}	19 (15%)	click-ele:find-ele	T_{ED}:T_{ED}	23 (20%)	find-ele:click-ele
2	T_{ED}:T_{R}	12 (10%)	click-ele:reponse	T_{E}:T_{E}	14 (12%)	e-update:e-update
3	T_{E}:T_{ED}	11 (9%)	e-load:click-ele	T_{ED}:T_{E}	9 (7%)	find-ele:e-wait
4	T_{E}:T_{R}	9 (7%)	e-update:response	T_{DE}:T_{DE}	8 (3%)	element-click:element-wait
5	T_{DE}:T_{DE}	8 (7%)	element-click:element-click	T_{E}:T_{DE}	7 (6%)	e-load:wait-ele

- Flakiness often arises from improper sequencing of event–DOM operations
  - e.g., click-ele:find-ele vs. find-ele:click-ele.
  - T_{ED}:T_{ED} interactions are high-risk—*multiple DOM-event sequences in a row amplify timing issues*
- Non-flaky sequences often involve **natural or stable orders** or **explicit waits.**
  - e.g., find-ele:e-wait, e-load:wait-ele

The impact of DOM event interactions

Impact of DOM Consistency & Interaction Level

- 76% of flaky tests involve different DOM contexts.
- 81% relate to element-level interactions; page-level alone is rare (2%).
- Flakiness is more tied to element-level operations and cross-DOM interactions than to page-level events.

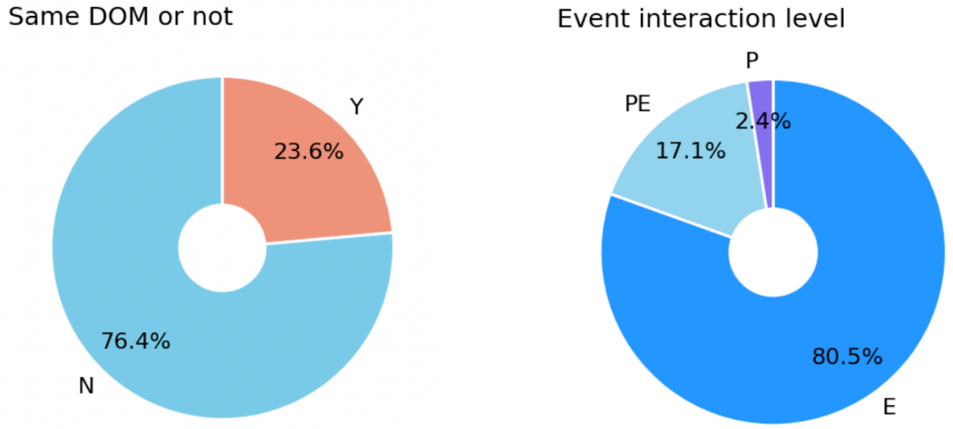


Fig. 5: The distribution of the same DOM element and event interaction levels associated with flakiness. *Y* and *N* are *yes* and *no* to whether the elements belong to the same DOM; *P*, *E*, and *PE* refer page level, element level, both.

The impact of DOM event interactions

Common developers' fix strategies

Fix strategy	Description	Example	# number
DOM interaction synchronization mechanism	Ensure elements are fully loaded/visible before interaction (e.g., waitUntil, waitForSelector)	await driver.waitForSelector('. qr-code__address');	62 (50.4%)
Conditional event completion waits	Add dynamic delays (sleep, timeout) to accommodate async operations.	cy.wait(500);	47 (38.2%)
Consistent DOM state transitions	Wait for all async callbacks/data updates to complete to avoid partial DOM states	await pWaitFor(()=> connection. streams.length === 0);	14 (11.4%)

- Framework support (e.g., Cypress, Jest) already integrates many synchronization methods, which developers should actively leverage.
- T_{ED} (Event-DOM) is the most frequently fixed type, while T_D (DOM) takes the longest to resolve (153 days on avg.).
- **Take-away:** these fix strategies highlight the importance of synchronization, timing, and state management in reducing flakiness.

Flakiness Localization

향후 연구 방향

- DOM event interaction flakiness repair
- Web UI test script generation and the robustness study