WEFix: Intelligent Automatic Generation of Explicit Waits for Efficient Web End-to-End Flaky Tests

Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, Weihang Wang

Published: 23 Jan 2024, Last Modified: 23 Jan 2024 TheWebConf24 Conference, Senior Area Chairs, Area Chairs, Reviewers, Authors Revisions BibTeX

Keywords: Web End-to-End Testing, Flaky Tests, Test Automation, Browser UI Changes, Testing Reliability, Software Testing, Web Development

TL;DR: WEFix introduces an intelligent, automated approach to generate explicit waits in web end-to-end testing, ensuring a high success rate in fixing flaky tests while maintaining low runtime overhead.

Abstract:

Web end-to-end (e2e) testing evaluates the workflow of a web application from beginning to end. It simulates real-world user scenarios to ensure the application flows behave as expected. Web e2e testing plays an indispensable role in the development of modern web applications. However, web e2e tests are notorious for being flaky, i.e., the tests can produce inconsistent results, passing or failing unpredictably, despite no changes to the code under test. One common type of flakiness in web e2e testing is caused by nondeterministic execution orders between the test code and the client-side code under test. In particular, UI-based flakiness emerges as a notably prevalent and challenging issue to mitigate. Such flaky tests are challenging to fix because the test code has limited knowledge about the client-side code execution.

In this paper, we propose WEFix, a technique that can automatically generate fix code for UI-based flakiness in web e2e testing. The core of our approach is to leverage browser UI changes to predict the client-side code execution and generate proper wait oracles. We evaluate the effectiveness and efficiency of WEFix against 122 web e2e flaky tests from seven popular real-world projects. Our results show that (1) UI-based flakiness is prevalent; (2) implicit waits introduce significant runtime overhead; (3) WEFix dramatically reduces the overhead (from 3.7× to 1.25×) while still achieving a high correctness (98%).

Track: Systems and Infrastructure for Web, Mobile, and WoT

Submission Guidelines Scope: Yes

Submission Guidelines Blind: Yes

Submission Guidelines Format: Yes

Submission Guidelines Limit: Yes

Submission Guidelines Authorship: Yes

Student Author: Yes

Serve As Reviewer: Weihang Wang, Weike Fang

Submission Number: 1888

Paper Decision

Decision Program Chairs 22 Jan 2024, 02:40 (modified: 23 Jan 2024, 01:40) Program Chairs, Senior Area Chairs, Area Chairs, Reviewers, Authors Revisions

Decision: Accept

Comment:

All the reviewers are quite positive about the paper, considering well-organized structure and clarity. The reviewers also highlight the real-world findings and fixings of flakiness in web e2e testing. Yet, the reviewers do concern some minor issues, which, if fixed, could enhance the overall quality and clarity of the work.

The authors are recommended to discuss the heuristic values set in the paper, which may significantly influence the performance. The authors may further clarify the evaluation settings, i.e., lack of comparison with other similar techniques, and limited evaluation on different browsers and platforms. Potential overhead/false positives/false negatives should also be discussed in the paper. The authors may also mention the feedback from developers about the reported flakiness.

Official Review of Submission1888 by Reviewer fBZd

Official Review Reviewer fBZd 25 Nov 2023, 05:14 (modified: 02 Dec 2023, 01:28) Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Reviewer fBZd, Authors Revisions

Review:

Summary

The authors design and develop an approach named WEFix to automatically fix the UI-based flakiness in web e2e testing. Specifically, WEFix leverages browser UI changes to predict the client-side code execution and generate proper wait oracles. By applying WEFix to 122 web e2e flaky tests from real-world web projects, the results show that WEFix can effectively find and fix flaky tests.

Strengths

- The authors propose a new approach to fix flaky web e2e tests.
- The authors implement a prototype of the proposed approach named WEFix.
- The effectiveness and performance of WEFix are evaluated on real-world web e2e tests. The tool does find several flaky tests and can fix most of these flaky tests.
- The artifacts of WEFix are open-source.

Weaknesses

- The confirmation on the fixed flaky tests has not been disclosed.

Detailed comments

I am quite positive about this paper. The approach proposed and the prototype implemented by the authors can effectively fix flaky web e2e tests. I just have a few suggestions and comments on the current version of the paper.

1. Evaluating and discussing the effectiveness of Algorithm 1.
   Since some of the values used in Algorithm 1 are set heuristically, the author should evaluate and prove these values are set properly.
2. Disclosing the uncovered flaky tests and including the feedback in the paper.
   The authors should follow the responsible disclosure policy to disclose their findings to the corresponding parties, such as developers and maintainers of web projects under evaluation. If the corresponding parties can help confirm the correctness of detection results, the authors could add the feedback in the paper to further show the effectiveness of WEFix.

Questions:

N/A

Ethics Review Flag: No

Ethics Review Description: N/A

Scope: 3: The work is somewhat relevant to the Web and to the track, and is of narrow interest to a sub-community

Novelty: 5

Technical Quality: 5

Reviewer Confidence: 3: The reviewer is confident but not certain that the evaluation is correct

Author Response to Reviewer fBZd

Official Comment Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) 11 Dec 2023, 04:22

Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:

We thank the reviewer for providing constructive feedback. Please find the detailed response below.

> **[Suggestion] Evaluate and discuss the effectiveness of Algorithm 1.**

We agree. We have evaluated and proved the effectiveness of the heuristic values used in Algorithm 1. We will add the following discussion to the paper to highlight the effectiveness of these chosen values.

The main idea of this algorithm is to double the size of the listening window whenever a new mutation occurs, allowing sufficient time for subsequent mutations. In particular, we set the initial window size to 1 second and the maximum size to 20 seconds. This initial size is chosen to effectively record all mutations in most applications, and the upper limit is crucial for

handling extremely complicated applications. This algorithm dynamically adjusts the window size to suit diverse applications while aiming to minimize overhead.

Our evaluation demonstrates the effectiveness of these heuristic window sizes:

1. Figure 6 illustrates the cumulative distribution of Relative Time (RT) across seven projects. As shown in the figure, a 1-second window size proves sufficient to capture 90% of mutations in our e2e tests.
2. The 20-second upper limit aligns with common practices in widely-used industrial testing frameworks such as Selenium and Cypress [1,2,3]. Figure 6 shows that all mutations have an RT less than 3.5 seconds, confirming the adequacy of the 20-second upper limit.

> **[Suggestion] Disclose uncovered flaky tests and solicit feedback.**

Thank you for suggesting that we disclose our results to corresponding parties and solicit feedback. We agree that this would be very helpful in showcasing WEFix's effectiveness and demonstrating its practical utility in real-world scenarios. Our plan is to share our findings with the developers and conduct an in-depth comparison between the flaky tests detected by WEFix and their original tests, all while adhering to the responsible disclosure policy. We will validate the detected instances of flakiness and identify any additional flakiness that may not have been previously discovered. Following this comparative analysis, we plan to work closely with the developers to seamlessly integrate any necessary patches into their code base. This process, disclosing any uncovered flakiness and incorporating feedback from the developers, will contribute to potential further enhancements of WEFix's design, ensuring its reliability and effectiveness in addressing flaky tests.

References

[1] Selenium. 2023. Waiting Strategies. https://www.selenium.dev/documentation/webdriver/waits/

[2] Krishna Rungta. 2023. Selenium Wait – Implicit, Explicit and Fluent Waits. https://www.guru99.com/implicit-explicit-waits-selenium.html

[3] Cypress and Zach Bloomquist. 2019. Increase CDP timeout to 20 seconds. https://github.com/cypress-io/cypress/pull/5610

Replying to Author Response to Reviewer fBZd

Reply to Authors' Response

Official Comment · Reviewer fBZd · 13 Dec 2023, 10:58 · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:
Thanks for the detailed response. All of my concerns have been addressed. Thanks!

Official Review of Submission1888 by Reviewer ojvd

Official Review · Reviewer ojvd · 21 Nov 2023, 11:59 (modified: 02 Dec 2023, 01:29) · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Reviewer ojvd, Authors

Revisions

Review:
Quality:

High Quality: The paper is well-structured, presenting a clear problem statement, detailed methodology, thorough evaluation, and insightful discussion. The research design is rigorous, and the results are supported by extensive data and analysis.

Clarity:

Clear and Comprehensive: The paper is well-written and easy to follow. Technical concepts are explained clearly, making it accessible to readers with a basic understanding of web testing. The use of figures, tables, and code snippets enhances understanding.

Originality:

Highly Original: The approach of using a Mutation Recorder and Oracle Generator to automatically insert explicit waits is novel. The idea of leveraging browser UI changes to predict client-side code execution is innovative and shows a deep understanding of the problem domain.

Significance:

Highly Significant: The paper addresses a critical and long-standing issue in web development - UI-based flakiness in e2e tests. The solution has the potential to significantly improve the reliability and efficiency of web testing, which is crucial for modern web application development.

Pros:

1. Innovative Approach: The use of explicit waits to handle flakiness is a significant improvement over traditional implicit wait approaches.
2. Effective Solution: WEFix demonstrates high effectiveness in fixing flaky tests, with a correctness of 98%.
3. Efficiency: The approach significantly reduces the runtime overhead compared to implicit waits.
4. Practicality: The tool is implemented as an NPM package, making it easily integrable into existing web development workflows.
5. Strong Evaluation: The paper presents a comprehensive evaluation using real-world projects, enhancing the credibility of the results.

Cons:

1. The innovative use of cookies for information storage in your paper is commendable for its ingenuity. However, the paper could benefit from more detailed explanations of certain technical terms, such as 'mutation' and 'oracle.' These terms, while well-understood within the domain, might be confusing to readers not specialized in web testing or those familiar with similar terms in the context of fuzzing. Clarifying these terms would enhance the paper's accessibility and prevent potential misunderstandings among a broader audience.
2. Dependence on Accurate Mutation Recording: The effectiveness of WEFix relies heavily on the accurate recording of DOM mutations, which might not always be reliable in every testing scenario.
3. Potential for Overhead in Certain Cases: While generally efficient, there might be scenarios where the dynamic adjustment of the wait window could lead to performance overhead.

Questions:

1. In your paper, the effectiveness of WEFix is significantly dependent on the accurate recording of DOM mutations. Could you elaborate on how WEFix ensures the reliability of this mutation recording across various web testing scenarios? Additionally, are there any particular scenarios where this mutation recording might be less reliable, and if so, how does WEFix handle such situations?
2. The paper mentions that WEFix is generally efficient, but acknowledges the potential for performance overhead in certain cases due to the dynamic adjustment of the wait window. Could you provide more details on the scenarios where this overhead is most likely to occur? How does WEFix mitigate this issue to maintain overall efficiency in web testing?

Ethics Review Flag: No
Ethics Review Description: /
Scope: 4: The work is relevant to the Web and to the track, and is of broad interest to the community
Novelty: 5
Technical Quality: 6
Reviewer Confidence: 2: The reviewer is willing to defend the evaluation, but it is likely that the reviewer did not understand parts of the paper

Author Response to Reviewer ojvd

Official Comment · Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) · 11 Dec 2023, 04:25 · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:
We thank the reviewer for providing constructive feedback. Please find the detailed response below.

> **[Question] Elaborate on how WEFix ensures the reliability of mutation recording.**

To accurately record DOM mutations, WEFix relies on MutationObserver [1] as the core mechanism to monitor any changes in the DOM tree. MutationObserver is a standardized, native web API integrated into modern browsers, which can faithfully track DOM changes as long as a sufficiently large listening window is set to wait until all mutations are completed. However, an excessively large window will result in substantial overhead. Therefore, we dynamically adjust the window size, striving to minimize overhead while ensuring high reliability. As described in Algorithm 1 (lines 332-340), we set an upper limit of 20 seconds, which aligns with common practices in widely used industrial testing frameworks such as Selenium and Cypress [2,3,4]. Although a 20-second listening window is more than adequate for projects under test (Figure 6 shows all mutations have an RT of less than 3.5 seconds), it is possible that extremely complicated e2e tests may experience mutations beyond 20 seconds, potentially causing WEFix to fail under the current setting.

> **[Question] Provide more details on performance overhead.**

In this paper, we address two types of overhead within WEFix. The first is instrumentation overhead, where WEFix records DOM mutations triggered by a command, determines whether the command is flaky-prone, and accordingly inserts proper explicit wait statements. This overhead becomes more pronounced when the DOM mutations have a large Relative Time (RT), requiring the Mutation Recorder to allocate additional time (a larger listening window) to capture all relevant mutations for a command.

Initially, the Mutation Recorder was configured with a fixed-size listening window; however, we observed that the average RT of mutations varies dramatically among projects (Table 1). To address this variability and reduce overhead, we dynamically adjust the window size. This adjustment ensures that WEFix maintains a minimal instrumentation overhead. In contrast to previous approaches, which require several minutes to fix one single command [5], WEFix can generate the DOM mutation profile in a single run and can fix a flaky-prone command in approximately one second.

The second type is test runtime overhead, which is caused by the addition of wait statements. Since the runtime of these tests is compared to the original flaky tests without additional wait, some degree of overhead is inevitable. Such overhead is particularly evident in tests containing a large number of flaky-prone commands, as each command requires the addition of explicit

waits. To mitigate this overhead, WEFix intelligently identifies commands prone to flakiness and only inserts proper wait oracles after those commands, thereby maintaining overall efficiency in web testing.

> **[Suggestion] Clarify technical terms (such as "mutation" and "oracle").**

In the background and methodology sections, we provide examples and definitions for key technical terms. Specifically, we define the term "mutation" in section 2.3 (line 203) and elaborate on the term "oracle" in section 3.2 (line 367) with an accompanying example (lines 378-383) illustrating how an oracle works. However, we acknowledge that our explanations may not be easily accessible to a broader audience. We will clarify these terms by including the following formal definitions in the paper:

1. A *mutation* is defined as any change made to the DOM, which represents an HTML document with a tree structure. Examples of mutations include adding an element to the DOM or modifying an attribute of an existing element.
2. An *oracle* refers to the condition specified in the explicit wait statement added after a flaky-prone command, such as checking for the presence of an element on the DOM. It is used to determine if the expected condition is met.

References

[1] Mozilla. Mozilla MutationObserver Document. https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver

[2] Selenium. 2023. Waiting Strategies. https://www.selenium.dev/documentation/webdriver/waits/

[3] Krishna Rungta. 2023. Selenium Wait – Implicit, Explicit and Fluent Waits. https://www.guru99.com/implicit-explicit-waits-selenium.html

[4] Cypress and Zach Bloomquist. 2019. Increase CDP timeout to 20 seconds. https://github.com/cypress-io/cypress/pull/5610

[5] Dario Olianas, Maurizio Leotta, and Filippo Ricca. 2022. SleepReplacer: A novel tool-based approach for replacing thread sleeps in selenium webdriver test code. Software Quality Journal 30, 4 (2022), 1089–1121.

Replying to Author Response to Reviewer ojvd

Official Comment by Reviewer ojvd

Official Comment · Reviewer ojvd · 14 Dec 2023, 07:31 · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:
Thank you for your comprehensive rebuttal in response to the queries raised during the review process. Your detailed explanations have effectively addressed my concerns and have provided clearer insights into your research.

Official Review of Submission1888 by Reviewer Pugt

Official Review · Reviewer Pugt · 19 Nov 2023, 20:36 (modified: 02 Dec 2023, 01:29) · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Reviewer Pugt, Authors
Revisions

Review:
Overview:

This paper proposes a framework for mutating flaky tests into reliability by automatically adding waits on explicit events (such as key values and so on).

Clarity:

The paper is well written and I could follow their arguments throughout. The literature review seems comprehensive and all figures and tables can be understood.

Qualities/Strengths:

This work seems extremely useful, and I am pleased to see that the authors have also released it as open source. The evaluation methodology over a number of open source projects seems sound, and the results are convincing.

Weaknesses:

No comparison with other approaches than just manually inserted delays.

Questions:
1. Why did the two failing cases fail? This is a small number and so the root cause could have been provided.
2. Are there cases where WEFix added unnecessary delays?

Ethics Review Flag: No
Ethics Review Description: NA
Scope: 4: The work is relevant to the Web and to the track, and is of broad interest to the community
Novelty: 5
Technical Quality: 6
Reviewer Confidence: 3: The reviewer is confident but not certain that the evaluation is correct

Author Response to Reviewer Pugt

Official Comment · Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) · 11 Dec 2023, 04:27 (modified: 11 Dec 2023, 05:09)
Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors · Revisions

Comment:
We thank the reviewer for providing constructive feedback. Please find the detailed response below.

> **[Question] Analyze root causes of the two failing cases.**

WEFix failed to fix two test files due to flakiness stemming from inconsistent internal states introduced by third-party tools, which is unrelated to DOM element changes. WEFix cannot capture these internal updates in third-party tools, preventing it from generating a proper fix for the flakiness.

For example, consider the UI test in the file *react-beautiful-dnd/reorder-virtual.spec.js*. It tests the drag-and-drop reordering of a virtual list, which is managed and rendered by an external virtualization library. This test involves updating the internal representation of the list's order and other state information within the third-party code. Although WEFix can effectively track all DOM mutations, it cannot access these internal changes in third-party tools.

This scenario lies beyond WEFix's scope to fix. The most practical solution in this case is to implement implicit waits, allowing sufficient time for all potential changes to conclude. This approach has been successfully used by developers to address such flakiness. We leave it as a future work to address third-party internal flakiness effectively and efficiently.

> **[Question] Discuss if there are cases where WEFix added unnecessary delays.**

Adding unnecessary delays will lead to an increase in runtime overhead. WEFix seeks to reduce overhead by intelligently identifying flaky-prone commands and adding proper oracles only after such commands. When a command is flagged as flaky-prone, WEFix selects three properties from the final DOM state with the latest mutation time and uses them to generate corresponding explicit waits. The number of properties chosen is a trade-off between stability and complexity: too few may be inadequate to eliminate flakiness, while too many could produce excessive, unnecessary delays, significantly increasing runtime overhead. Choosing three properties strikes a balance, achieving a high fix rate of 98.4% with a low overhead increase of 16%.

It is important to note that the presence of flakiness can differ based on factors such as device specifications and network conditions. Thus, some introduced delays may be unnecessary in certain e2e test environments. It's nearly impossible to determine if an inserted wait is redundant across all circumstances. To address this, we plan to inform developers of tested projects to discuss the necessity of any injected delays.

> **[Suggestion] Compare WEFix with other automatic approaches.**

While end-to-end (e2e) flaky tests have been found to be prevalent, there have been few efforts employing automatic explicit waits. WEFix stands out in this regard, distinctly different from the few existing solutions. To the best of our knowledge, the only existing method that attempts to automatically fix e2e flaky tests with explicit waits is SleepReplacer [1] by Olianas *et al.* However, as detailed in the Related Work section, WEFix and SleepReplacer have fundamentally different designs and use scenarios. In particular, SleepReplacer, which relies on the availability of implicit waits, passively replaces pre-existing implicit waits with explicit waits. By contrast, WEFix is designed to proactively insert explicit waits for flaky commands even when no delays are originally present. This proactive approach is pivotal as it automates the manual efforts to select proper explicit waits, which closely resembles real-world development practices. Consequently, SleepReplacer does not fit inside our comparison scope, as it is limited to flaky tests with pre-existing wait statements. Therefore, we compare WEFix against various implicit wait baselines, which represent the traditional approach to automatically resolving test flakiness.

Although the fix rates are not directly comparable, we can still compare the efficiency of the two approaches. For example, SleepReplacer requires several minutes (up to 10 minutes) to replace a single implicit wait with an explicit one. In comparison, WEFix is notably more efficient, taking only around one second to instrument each command. This stark contrast underscores the efficiency enhancement in addressing flaky tests compared to SleepReplacer.

References

[1] Dario Olianas, Maurizio Leotta, and Filippo Ricca. 2022. SleepReplacer: A novel tool-based approach for replacing thread sleeps in selenium webdriver test code. Software Quality Journal 30, 4 (2022), 1089–1121.

Replying to Author Response to Reviewer Pugt

Official Comment by Reviewer Pugt

Official Comment · Reviewer Pugt · 13 Dec 2023, 22:54 · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:
Thank you for your response.

Official Review of Submission1888 by Reviewer nrSK

Official Review · Reviewer nrSK · 12 Nov 2023, 05:44 (modified: 02 Dec 2023, 01:29) · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Reviewer nrSK, Authors
Revisions

Review:
This paper presents WEFix to automatically generate fix code for UI-based flakiness. It inserts explicit waits, which wait until certain condition is met, to mitigate flaky instances. The paper reads smoothily and I can follow its idea even though I am not an expert in web e2e test. The evaluation shows that the solution can significantly improve the efficiency of implicit wait solutions.

pros:

- it automatically inserts fix to mitigate flakiness
- the fix can adaptably adjust to web events

cons:

- values determined by experience (e.g., three properties and window size) may have significant influence on performance

Questions:

- about the originality: Is there any existing solutions that use explicit waits? In section 3.2 (page 4), it mentions some existing explicit waits. What about their performancce? why not compare them?
- How to differentiate between flaky events and random events? some events may contain random values.
- Will the finite state machine be too large? If yes, what's the influence on the performance?

Ethics Review Flag: No
Ethics Review Description: N/A
Scope: 4: The work is relevant to the Web and to the track, and is of broad interest to the community
Novelty: 5
Technical Quality: 5
Reviewer Confidence: 3: The reviewer is confident but not certain that the evaluation is correct

Author Response to Reviewer nrSK (1/2)

Official Comment · Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) · 11 Dec 2023, 04:31
Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:
We thank the reviewer for providing constructive feedback. Please find the detailed response below.

> **[Question] Compare WEFix's performance with existing solutions using explicit waits.**

While e2e flaky tests are prevalent, there have been few efforts using automatic explicit waits. To the best of our knowledge, WEFix distinguishes itself from existing solutions, with SleepReplacer [1] being the only existing method using explicit waits. However, as detailed in the Related Work section, WEFix and SleepReplacer have fundamentally different designs and use scenarios. SleepReplacer relies on the presence of implicit waits and passively replaces pre-existing implicit waits with explicit waits. In contrast, WEFix proactively inserts explicit waits for flaky commands, even when no delays were initially present. This proactive approach is pivotal as it automates the manual efforts to select proper explicit waits, which closely resembles real-world development practices. SleepReplacer does not fit inside our comparison scope, as it is limited to flaky tests with pre-existing waits. Therefore, we compare WEFix against implicit wait baselines, which represent the traditional approach to automatically resolving flakiness.

While fix rates are not directly comparable, we can still compare the efficiency of the two approaches. SleepReplacer requires several minutes (up to 10 minutes) to replace a single implicit wait with an explicit one, whereas WEFix is notably more efficient, taking only around one second to instrument each command. This substantial difference underscores WEFix's efficiency in addressing flaky tests compared to SleepReplacer.

> **[Question] Differentiate flaky events from random events.**

We distinguish between two types of randomness in e2e testing. The first is randomness or uncertainty in the environment, such as network delays and server loads, which WEFix can effectively address. In such scenarios, "async wait" flakiness may arise if no proper waits were provided to ensure prior mutations have completed before proceeding to the next command. WEFix can effectively deal with this kind of randomness and fix such flakiness that may arise.

The second type of randomness is inherent randomness in test code, such as the usage of a random number generator. WEFix does not address this unless it causes UI-based "async wait" flakiness. As our paper details, an e2e test is flaky if the test outcome, i.e., pass or fail, varies across multiple runs. The e2e test should be designed to behave consistently throughout different runs, regardless of the randomness in test code.

> **[Question] Discuss the impact of the finite state machine on performance.**

The finite state machine (FSM) representing DOM tree transitions may grow in size as e2e tests become more complex. However, it's important to note that the FSM is constructed during the Mutation Recorder phase, which impacts only the instrumentation time needed to create explicit waits for flaky-prone commands.

In terms of time complexity, the Mutation Recorder is designed to record all DOM mutations in a single run, making the time comparable to a standard test execution. Notably, WEFix only uses the end state for generating test oracles, so the time complexity for oracle generation is constant.

Regarding space complexity, during FSM construction, the Oracle Generator processes the mutation list in a temporal sequence, ensuring a linear FSM that effectively prevents exponential growth in size. Additionally, mutations are stored locally in a JSON format, addressing any memory concerns associated with exponential growth in state transitions.

Therefore, WEFix effectively minimizes the performance impact of the FSM size, ensuring both low time and space complexity.

Replying to Author Response to Reviewer nrSK (1/2)

Author Response to Reviewer nrSK (2/2)

Official Comment · Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) · 11 Dec 2023, 04:32 (modified: 11 Dec 2023, 04:37)
Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors · Revisions

[Deleted]

Author Response to Reviewer nrSK (2/2)

Official Comment · Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) · 11 Dec 2023, 04:37
Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:

> **[Suggestion] Discuss the selection of heuristic values and their influence on performance.**

We acknowledge that setting heuristic values may affect performance for various applications and testing scenarios. We aim to reduce such impacts by following common practices and adopting empirical evidence.

Our choice of the number of reruns follows common practice. When reproducing flaky tests, we followed an existing work on flaky tests [2] to set the rerun times to be 10 to decide whether a test is flaky. This threshold proves effective in determining flakiness.

Our choice of the last three properties to generate corresponding explicit waits balances the trade-off between stability and complexity. Too few may be inadequate to eliminate flakiness, while too many could produce excessive, unnecessary delays, significantly increasing runtime overhead. Choosing three properties strikes a balance, achieving a high fix rate of 98.4% with a low runtime overhead increase of 16%.

Our decision for the window size (Algorithm 1) is supported by empirical evidence. The main idea of this algorithm is to double the size of the listening window whenever a new mutation occurs, allowing sufficient time for subsequent mutations. In particular, we set the initial window size to 1 second and the maximum size to 20 seconds. This algorithm dynamically adjusts the window size to suit diverse applications while aiming to minimize overhead.

Our evaluation demonstrates the effectiveness of these heuristic window sizes:

- Figure 6 illustrates the cumulative distribution of Relative Time (RT) across seven projects. As shown in the figure, a 1-second window size proves sufficient to capture 90% of mutations in our e2e tests.
- The 20-second upper limit aligns with common practices in widely-used industrial testing frameworks such as Selenium and Cypress [3,4,5]. Figure 6 shows that all mutations have an RT less than 3.5 seconds, confirming the adequacy of the 20-second upper limit.

References

[1] Dario Olianas, Maurizio Leotta, and Filippo Ricca. 2022. SleepReplacer: A novel tool-based approach for replacing thread sleeps in selenium webdriver test code. Software Quality Journal 30, 4 (2022), 1089–1121.

[2] August Shi, Wing Lam, Reed Oei, Tao Xie, and Darko Marinov. 2019. iFixFlakies: 1024 A framework for automatically fixing order-dependent flaky tests. In Proceedings 1025 of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 545–555.

[3] Selenium. 2023. Waiting Strategies. https://www.selenium.dev/documentation/webdriver/waits/

[4] Krishna Rungta. 2023. Selenium Wait – Implicit, Explicit and Fluent Waits. https://www.guru99.com/implicit-explicit-waits-selenium.html

[5] Cypress and Zach Bloomquist. 2019. Increase CDP timeout to 20 seconds. https://github.com/cypress-io/cypress/pull/5610

Replying to Author Response to Reviewer nrSK (2/2)

Thanks

Official Comment · Reviewer nrSK · 11 Dec 2023, 18:00 · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:
Thanks for your clarification.

Official Review of Submission1888 by Reviewer 6eJn

Official Review · Reviewer 6eJn · 30 Oct 2023, 06:38 (modified: 02 Dec 2023, 01:29) · Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Reviewer 6eJn, Authors

Revisions

Review:

Summary

This paper proposes WEFix, a technique that can automatically generate explicit waits for web end-to-end (e2e) tests that are flaky due to UI changes. WEFix records the DOM mutations triggered by each command and generates proper wait oracles based on them. WEFix also uses a finite state machine to model the DOM mutation events and ensure the correctness and efficiency of the wait oracles. The paper evaluates WEFix on seven real-world web projects and shows that it can fix 98% of the UI-based flaky tests with low overhead.

Pro

1. This paper discusses an interesting yet important problem
2. The paper evaluates WEFix on seven popular real-world web projects and shows that it can fix 98% of the UI-based flaky tests with low overhead
3. The dataset is publicially available.

Cons

1. Lack of comparison with other techniques
2. Limited evaluation on different browsers and platforms
3. Potential false positives and false negatives
4. Lack of user study or feedback

Detailed Comments

1. **Lack of comparison with other techniques**: The paper does not compare WEFix with other existing techniques for fixing UI-based flakiness, such as Flakiness Diagnosis and Fixing or Flakiness Detection and Localization. A comparison with these techniques could show the advantages and disadvantages of WEFix in different scenarios.
2. **Limited evaluation on different browsers and platforms**: The paper only evaluates WEFix on Chrome browser and CircleCI platform. However, UI-based flakiness may vary across different browsers and platforms, such as Firefox, Safari, or GitHub Actions. A more comprehensive evaluation on different browsers and platforms could demonstrate the robustness and generalizability of WEFix.
3. **Potential false positives and false negatives**: The paper does not discuss the possibility of false positives and false negatives in WEFix. False positives refer to cases where WEFix adds unnecessary explicit waits that do not fix any flakiness but increase the runtime overhead. False negatives refer to cases where WEFix fails to add explicit waits that are needed to fix the flakiness. A discussion on how to measure and reduce the false positives and false negatives could improve the reliability and accuracy of WEFix.
4. **Lack of user study or feedback**: The paper does not report any user study or feedback on WEFix. User study or feedback could provide valuable insights into how developers perceive and use WEFix, what are the benefits and challenges of using WEFix, and what are the potential improvements or extensions of WEFix.
5. **Lack of scalability evaluation**: The paper does not evaluate the scalability of WEFix on large-scale web applications or test suites. It is possible that WEFix may encounter performance issues or limitations when dealing with complex web applications or test suites that involve a large number of commands, mutations, elements, or properties. A scalability evaluation could show the practicality and applicability of WEFix in real-world scenarios.
6. **Lack of in-depth discussion on results**: This paper could benefit from a more in-depth discussion of the results.

Questions:
1. it's unclear how this algorithm deals with complex scenarios where multiple mutations occur simultaneously or in a rapid sequence. Could you elaborate on how your

Ethics Review Flag: No
Ethics Review Description: /
Scope: 4: The work is relevant to the Web and to the track, and is of broad interest to the community
Novelty: 4
Technical Quality: 4
Reviewer Confidence: 3: The reviewer is confident but not certain that the evaluation is correct

Author Response to Reviewer 6eJn (1/2)

Official Comment · Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) · 11 Dec 2023, 04:34

Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:
We thank the reviewer for providing constructive feedback. Please find the detailed response below.

> **[Question] Elaborate on how WEFix handles complex scenarios involving simultaneous or rapid sequences of mutations.**

WEFix uses MutationObserver [1] to record all DOM mutations triggered by each flaky-prone command. MutationObserver, a standard web API integrated into modern browsers, can faithfully record all mutation events, even when multiple mutations occur simultaneously or rapidly. As long as the mutation sequence is causally consistent, meaning that mutations are recorded in order, WEFix guarantees robustness and accurate final DOM states for generating proper wait oracles.

While not observed in our testing, we acknowledge the potential for MutationObserver to deviate from strict causal ordering, especially as we expand our testing to include more complex e2e tests in collaboration with organizations managing large-scale web applications. We recognize this as an area for potential improvement.

> **[Suggestion] Compare WEFix with other approaches.**

While e2e flaky tests are prevalent, there have been few efforts using automatic explicit waits. To the best of our knowledge, WEFix distinguishes itself from existing solutions, with SleepReplacer [2] being the only existing method using explicit waits. However, as detailed in the Related Work section, WEFix and SleepReplacer have fundamentally different designs and use scenarios. SleepReplacer relies on the presence of implicit waits and passively replaces pre-existing implicit waits with explicit waits. In contrast, WEFix proactively inserts explicit waits for flaky commands, even when no delays were initially present. This proactive approach is pivotal as it automates the manual efforts to select proper explicit waits, which closely resembles real-world development practices. SleepReplacer does not fit inside our comparison scope, as it is limited to flaky tests with pre-existing waits. Therefore, we compare WEFix against implicit wait baselines, which represent the traditional approach to automatically resolving flakiness.

While fix rates are not directly comparable, we can still compare the efficiency of the two approaches. SleepReplacer requires several minutes (up to 10 minutes) to replace a single implicit wait with an explicit one, whereas WEFix is notably more efficient, taking only around one second to instrument each command. This substantial difference underscores WEFix's efficiency in addressing flaky tests compared to SleepReplacer.

> **[Suggestion] Evaluate WEFix on different browsers and platforms.**

Thank you for pointing out the importance of evaluating our method across different browsers and platforms.

We acknowledge that browser-specific behaviors can impact flakiness, given differences in how browsers handle asynchronous events and DOM updates. To mitigate such impacts, we adhere to common practices for testing frameworks. For example, we set a 20-second limit for the dynamic listening window, following industry standards in widely-used testing frameworks such as Selenium and Cypress [3,4,5]. When reproducing flakiness, we use guidelines from previous work on flaky tests [6], setting rerun times to 10 to determine if a test is flaky. These heuristic thresholds have been proven to be effective and are expected to be robust across different browsers.

We also recognize the importance of verifying WEFix's effectiveness across different CI/CD platforms. Such evaluations will help us determine if WEFix is robust and adaptable. Although our current evaluation is on CircleCI, the testing frameworks (Selenium and Cypress) are compatible with various CI/CD platforms, thus WEFix is expected to function seamlessly within these environments.

We leave these evaluations as future work to test WEFix's performance across different browsers and platforms to identify any discrepancies or robustness issues.

Author Response to Reviewer 6eJn (2/2)

Author Response to Reviewer 8ejh (2/2)

Official Comment · Authors (Xinyue Liu, Zihe Song, Weike Fang, Wei Yang, +1 more) · 11 Dec 2023, 04:35

Program Chairs, Senior Area Chairs, Area Chairs, Reviewers Submitted, Authors

Comment:

> **[Suggestion] Discuss potential false positives and false negatives.**

To reduce false positives, WEFix intelligently identifies flaky-prone commands and adds proper oracles only after such commands. The number of properties chosen for generating oracles strikes a balance between stability and complexity, achieving a 98.4% fix rate with a 16% overhead increase.

The presence of flakiness can differ based on factors such as device specifications and network conditions. Thus, some introduced delays may be unnecessary in certain e2e test environments. It's nearly impossible to determine if an inserted wait is redundant across all circumstances. To address this, we plan to inform developers of tested projects to discuss the necessity of any injected delays.

Regarding false negatives, WEFix failed to fix two test files due to flakiness stemming from inconsistent internal states introduced by third-party tools, which is unrelated to DOM element changes. WEFix cannot capture these internal updates in third-party tools, making it unable to provide a proper fix. The most practical solution is to implement implicit waits, a method successfully used by developers to handle such flakiness, which allows sufficient time for all potential changes to conclude. Addressing third-party internal flakiness is beyond WEFix's scope, and we leave it as future work.

> **[Suggestion] Include a user study and solicit feedback on WEFix.**

We agree that including a user study and developer feedback would be beneficial and enhance our understanding of WEFix's usability and potential areas of improvement.

Recognizing that participants must be knowledgeable in e2e testing and adept at spotting and fixing flakiness, we plan to collaborate with major industry enterprises for a comprehensive user study as a follow-up work. The collaboration will involve integrating WEFix into their existing e2e test workflows and collecting detailed feedback from developers. This feedback will be comprehensive, delving into various aspects of WEFix's usability, including perceived benefits and challenges of using WEFix, efficiency in reducing manual efforts, ease of integration into existing workflows, and potential extensions and improvements. The insights gained from the study will be instrumental in reporting on WEFix's usability and guiding its future enhancements.

> **[Suggestion] Evaluate the scalability of WEFix on large-scale web applications or test suites.**

In Section 4.1, we discussed our evaluation of seven large-scale, real-world projects, which have an average of 36.3k stars, 486 contributors, 4329 forks, and 1546 files. These projects are widely used by major enterprises. For example, Storybook, a UI development tool, is adopted by prominent companies including Microsoft, Shopify, Airbnb, and Salesforce. Our evaluation proves WEFix's effectiveness on these large-scale, real-world projects with a high fix rate and a low overhead.

To further evaluate scalability, we plan to collaborate closely with industry partners for an in-depth analysis. This analysis will specifically focus on the practicality and applicability of WEFix on large and complex web applications.

References

[1] Mozilla. Mozilla MutationObserver Document. https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver

[2] Dario Olianas, Maurizio Leotta, and Filippo Ricca. 2022. SleepReplacer: A novel tool-based approach for replacing thread sleeps in selenium webdriver test code. Software Quality Journal 30, 4 (2022), 1089–1121.

[3] Selenium. 2023. Waiting Strategies. https://www.selenium.dev/documentation/webdriver/waits/

[4] Krishna Rungta. 2023. Selenium Wait – Implicit, Explicit and Fluent Waits. https://www.guru99.com/implicit-explicit-waits-selenium.html

[5] Cypress and Zach Bloomquist. 2019. Increase CDP timeout to 20 seconds. https://github.com/cypress-io/cypress/pull/5610

[6] August Shi, Wing Lam, Reed Oei, Tao Xie, and Darko Marinov. 2019. iFixFlakies: 1024 A framework for automatically fixing order-dependent flaky tests. In Proceedings 1025 of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 545–555.