Reflect

Strategies to improve maintainability and reduce flakiness in automated regression tests

Many software teams struggle with automated end-to-end test suites that are flaky and take significant effort to maintain. When flakiness and poor maintainability are prevalent, the following issues often emerge:

- Tests for new features are built several sprints behind development, or not at all.
- Manual testing is required to make up for a lack of sufficient automated test coverage.
- A significant percentage of testers' time is spent updating existing tests.
- Testing becomes the bottleneck in the software release cycle.

In this whitepaper, we cover the common drivers of maintainability issues, and provide actionable advice on how to avoid these pitfalls in your organization.

Who We Are

Reflect (**https://reflect.run**) is an AI-assisted test automation platform for web applications. Our platform enables QA teams and engineering organizations to increase test coverage and catch more bugs without slowing down development velocity.

Being Resilient to Changes in the Application

If there's one constant in software, it's change. To compete in today's environment, software teams must be agile to customer needs and move fast, with quality. But what if every change made by a developer risks introducing breaking changes to your regression test suite?

This is a reality for many organizations. Below are common reasons why false positive failures occur, and how to mitigate them:

Lack of Stable Selectors

As the application changes, the selectors (also known as Locators) used in the automation framework become out-of-date.

Solutions:

- Standardize on an attribute like `data-test` or `data-testid` that is added to the application's HTML markup and is used specifically for test automation.
- Define `aria-*` attributes in your markup and use them as selectors. This provides a dual benefit of improving test resilience, while making your app more accessible.
- For component-based applications, adopt a policy of defining a `data-test` attribute at the root of each component definition whose value is the name of the component.
- Ensure your underlying testing tool or framework supports the concept of "self-healing" selectors, so that actions can be performed even when selectors become out of date.

REFLECT'S APPROACH

Reflect generates multiple selectors for each step in a test. Our selector generation algorithm generates a diverse set of selectors and ranks them according to specificity and semantic relevance. For React apps, it's even able to locate and target elements based on its underlying React component name.

Reflect supports self-healing via a Generative AI approach. When selectors become stale, the Reflect AI uses the description of the test step (e.g. 'Click on the Submit button') to determine what element on the page can fulfill that action, if any. This approach has proven to be resilient to even large-scale changes in an application.

Expressing Intent

In order to build automated tests that accurately verify the requirements of your application, it's imperative to use a tool or framework that is capable of expressing the full range of actions and assertions that fulfill your testing requirements.

Common Pitfalls:

- Interactions within a table or grid view are notoriously hard to automate and can introduce significant flakiness if not handled correctly. This is because tables and grids are dynamic by nature. Any change to the sorting, filtering, or contents of the table between test runs can cause tests to fail erroneously.
- Be wary of scenarios where elements are accessed in relation to other elements. *e.g. "Click the edit button associated with the row 'John Doe'"*
- Negative tests can be difficult to accurately test in an automation tool or framework. For example, a common negative test is to verify that no errors are displayed. But how can you be sure your automated test is looking for all possible errors?
- Ensure your testing tool can easily extract dynamic data from the application under test for future use. Not all data can be controlled in an AUT, and having the ability to be resilient to changes to things like pricing and availability helps keep maintenance costs low.

REFLECT'S APPROACH

Reflect is unique in that test steps can be created via **record-and-playback features**, **JavaScript-based steps**, or **"prompts" to the Reflect AI** (ala ChatGPT).

Using Reflect's record-and-playback features, users can quickly record virtually any action within a web browser. Code-based steps allows for interactions and assertions against the underlying state of the page. AI Prompts enable users to perform actions (*"Fill out the form fields with realistic values")* and assertions (*"Verify that the amount for this Opportunity is greater than $10,000"*) by writing plain English sentences that do not need to conform to any specific syntax.

Handling the Complexities of the Web Browser

The web browser is one of the most important, and most complex, pieces of software that's ever been developed. It also poses significant challenges for test automation due to the dynamic nature of a typical web application.

Web application performance can vary significant from run to run. These variations are a major cause of test flakiness unless the automated tests are purposefully built to be resilient to these changes.

Common Pitfalls:

- Hard-coded waits within test automation code (e.g. `Thread.sleep(1000)`) make bad assumptions about the performance of the page. In the best case, hard-coded waits cause the tests to run longer than necessary. In the worst case, they don't wait long enough and cause false failures that are difficult to debug.
- Network calls will vary between runs, and especially in situations where UI behavior is dependent on a network response, these variations can cause tests to have unexpected errors. Consider a button on an application that is disabled until an AJAX request has completed. For the automated test to be resilient to variations in network timings, it must wait for the AJAX call to complete, or for the button to be re-enabled, before interacting with the button.
- Single-Page Applications (SPAs) render content without a full page load, making it difficult for some testing tools and frameworks to determine when the page is interactable.
- Applications that utilize animations for page transitions and loading states necessitate that automated tests are aware of when animations are executing and wait for them to complete before continuing.
- The most complex interactions supported within a web browser are often the most difficult to automate. If your application supports complex interactions, it's important to ensure that your testing tool can accurately test them. These interactions include:
  - Drag-and-drop
  - File uploads
  - Workflows that require interacting with an email or SMS message
  - Workflows that require verifications against a third-party system

REFLECT'S APPROACH

Reflect does the heavily lifting under-the-covers to ensure tests are resilient to the differences in behavior that can occur between test runs. Each test step in Reflect uses *smart waits* that wait for an element to be fully visible and interactable before proceeding with the next action. Reflect also detects when network calls are invoked as a result of an action, and will wait for those calls to complete before proceeding.

For testing complex interactions, Reflect has best-in-class support for file uploads, drag-and-drop, and email/SMS testing. Third party integrations can be easily tested as part of a larger workflow via Reflect's built-in API testing support.

Conclusion

To move fast with quality, software teams must ensure that the pace of testing can keep up with the speed of development.

A successful test automation strategy requires not only the right approach, but the right tools. When evaluating your test strategy, pay close attention to whether the issues and pitfalls outlined in this webinar are occurring, and implement changes in process or tooling.

For more testing strategy and tips, visit our website at **https://reflect.run**

About Reflect
Reflect is an AI-assisted test automation platform for web applications. Reflect enables QA teams and engineering organizations to increase test coverage and catch more bugs without slowing down development velocity.