

1. Retry-ability & Test Flakiness Prevention

Cypress has built-in retry-ability, meaning it automatically retries commands and assertions until they pass or timeout.

Flakiness usually comes from timing issues, animations, dynamic elements, or network delays.

Goal: write tests that are **consistent, stable, and deterministic**.

2. Handling Dynamic Elements & Race Conditions

Modern UIs load content asynchronously, causing tests to fail if elements aren't ready.

Cypress helps by:

- Using `.should()` for automatic retries
- Waiting for **API intercepts** instead of fixed `wait()`
- Ensuring stable selectors (`data-testid` attributes)
- Avoiding chained commands before elements appear

This prevents tests from interacting with elements too early.

3. Implementing Retries & Timeouts

Cypress supports both:

Built-in Retries

- Added automatically for commands like `cy.get()`, `cy.contains()`, and assertions.
- Configurable in `cypress.config.js`:
 - **retries for failed tests**
 - **default command timeouts**

Timeouts

You can extend timeouts for slow pages or operations:

```
cy.get('.item', { timeout: 10000 }).should('be.visible')
```

Retries + timeouts = stable, predictable tests.

4. Debugging Flaky Tests

Common flakiness debugging techniques:

- Review **screenshots & videos** captured by Cypress.
- Use `cy.pause()` and `cy.debug()` to inspect live behavior.
- Add `cy.intercept()` waits for network stability.
- Replace positional selectors (`:nth-child`) with stable test IDs.
- Log messages using `cy.log()` to trace sequence of actions.
- Confirm backend test data is reset properly before tests.

A systematic approach helps isolate whether the issue is **UI timing**, **network delay**, or **test data inconsistency**.