

Từ Script Dễ Vỡ Đến Agent Tự Phục Hồi



Tái định nghĩa
Kiểm thử Tự động

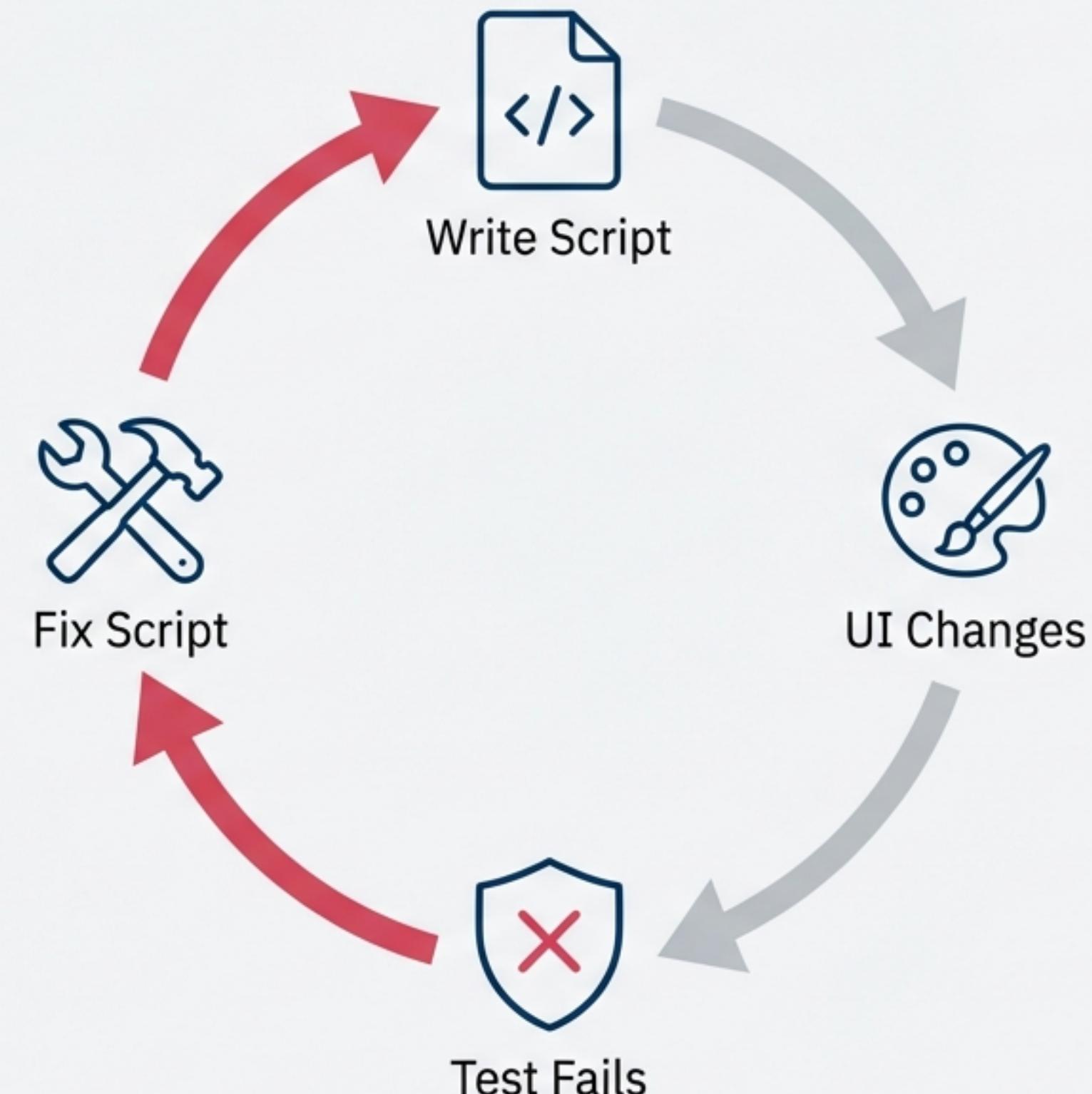
với Playwright MCP và Tư
duy dựa trên Chủ đích
(Intent-based Thinking)



Một phân tích về xu hướng công nghệ kiểm thử cho năm 2025 và xa hơn.

‘Nỗi Đau Thầm Lặng’ của Mọi Kỹ Sư QA: Vòng Lặp Bảo Trì Vô Tận

- Các kịch bản kiểm thử (test scripts) của chúng ta cực kỳ ‘giòn’. Một thay đổi nhỏ về giao diện (UI) cũng có thể làm sụp đổ toàn bộ hệ thống kiểm thử.
- **Vấn đề cốt lõi:** Chúng ta đang dành nhiều thời gian để ‘sửa’ các bài test hơn là để ‘viết’ các bài test mới có giá trị.
- **Câu hỏi đặt ra:** Liệu có cách nào để các bài test của chúng ta đủ thông minh để ‘tự phục hồi’ khi giao diện thay đổi không?



Tại Sao Test Hay Hỗng? Vì Chúng Ta Đang Chỉ Đường Cho Người Bị Bịt Mắt

Cách cũ (Dựa trên DOM/XPath):

Chúng ta ra lệnh cho máy một cách máy móc, dựa trên cấu trúc kỹ thuật của trang web.

Phép ẩn dụ (Analogy):

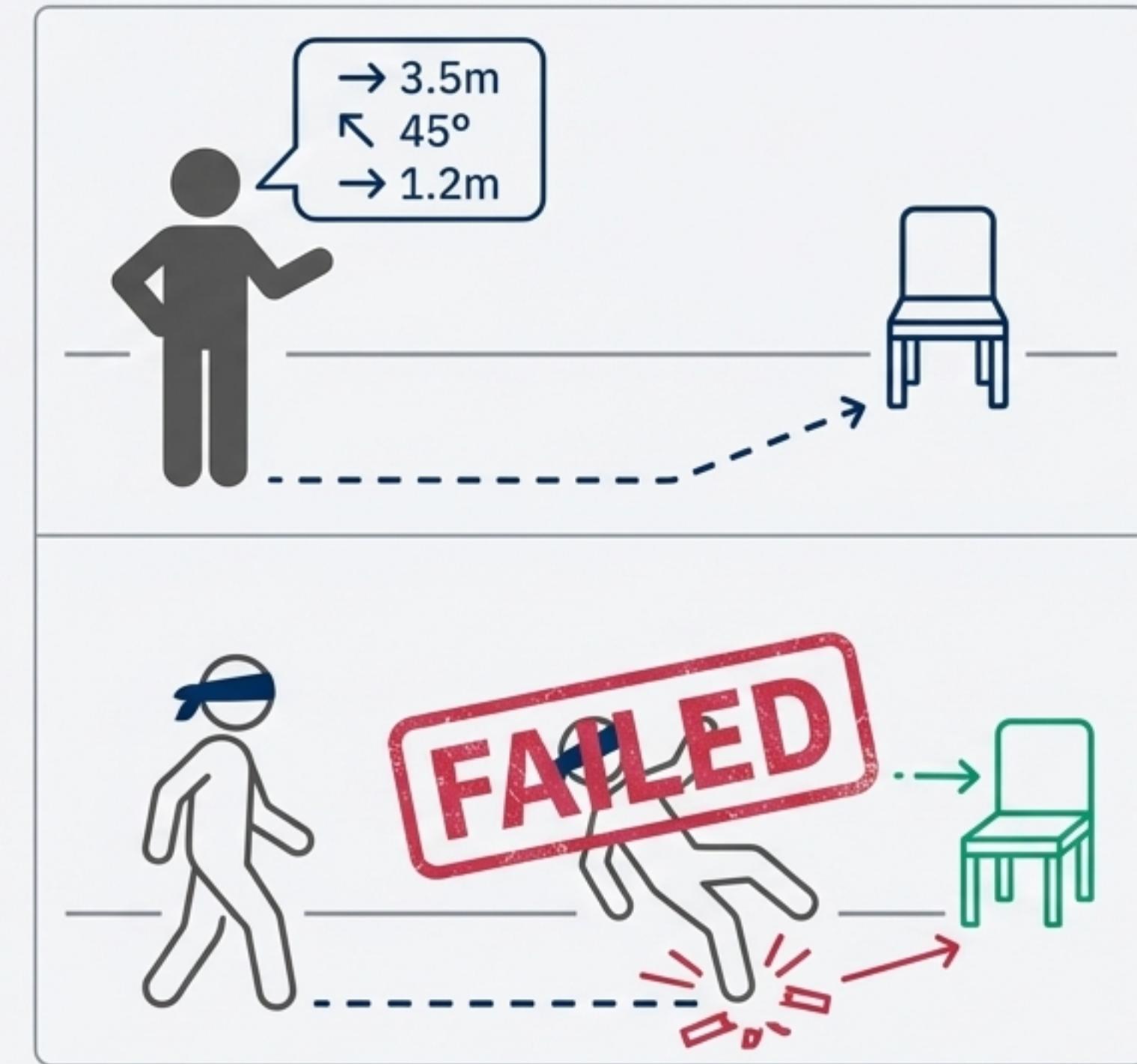
Giống như chỉ đường cho một người bị bịt mắt tìm ghế trong phòng tối.

Câu lệnh của chúng ta:

Đi thẳng 3.5 mét. Rẽ trái 45 độ. Đi tiếp 1.2 mét. Ngồi xuống.

Kết quả tất yếu:

Chỉ cần ai đó xê dịch cái ghế 10cm, chương trình sẽ thất bại. Người đó sẽ ngã. Test sẽ rẽ báo 'FAILED'.



Cách Mới: Hãy Cứ Nói Cho AI Biết Bạn Muốn Gì

Cách mới (Dựa trên Ngữ nghĩa/Accessibility):

Thay vì ra lệnh từng bước, chúng ta mô tả mục tiêu cuối cùng.

Phép ẩn dụ (Analogy):

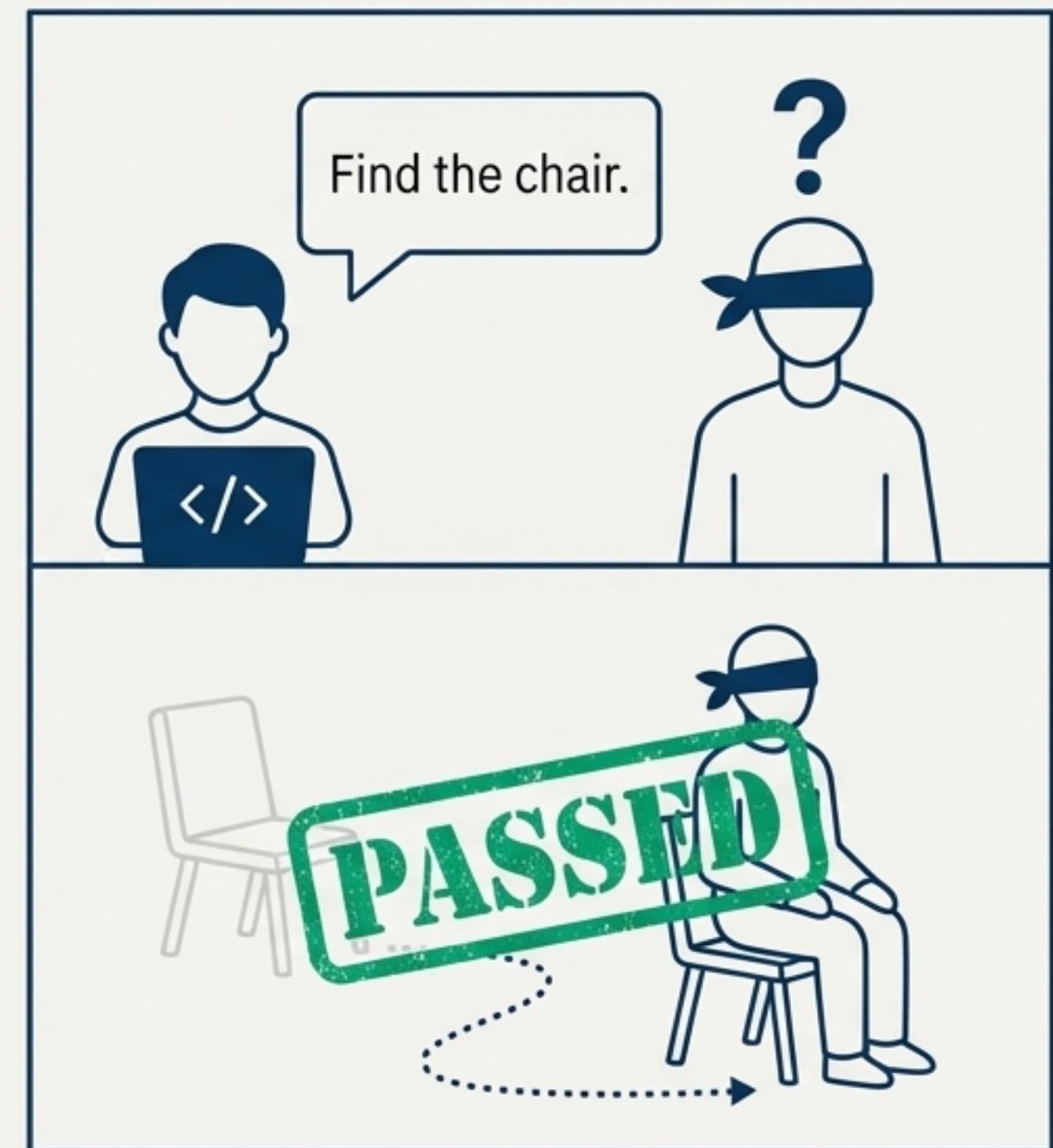
Vẫn là câu chuyện tìm ghế trong phòng tối.

Câu lệnh mới của chúng ta:

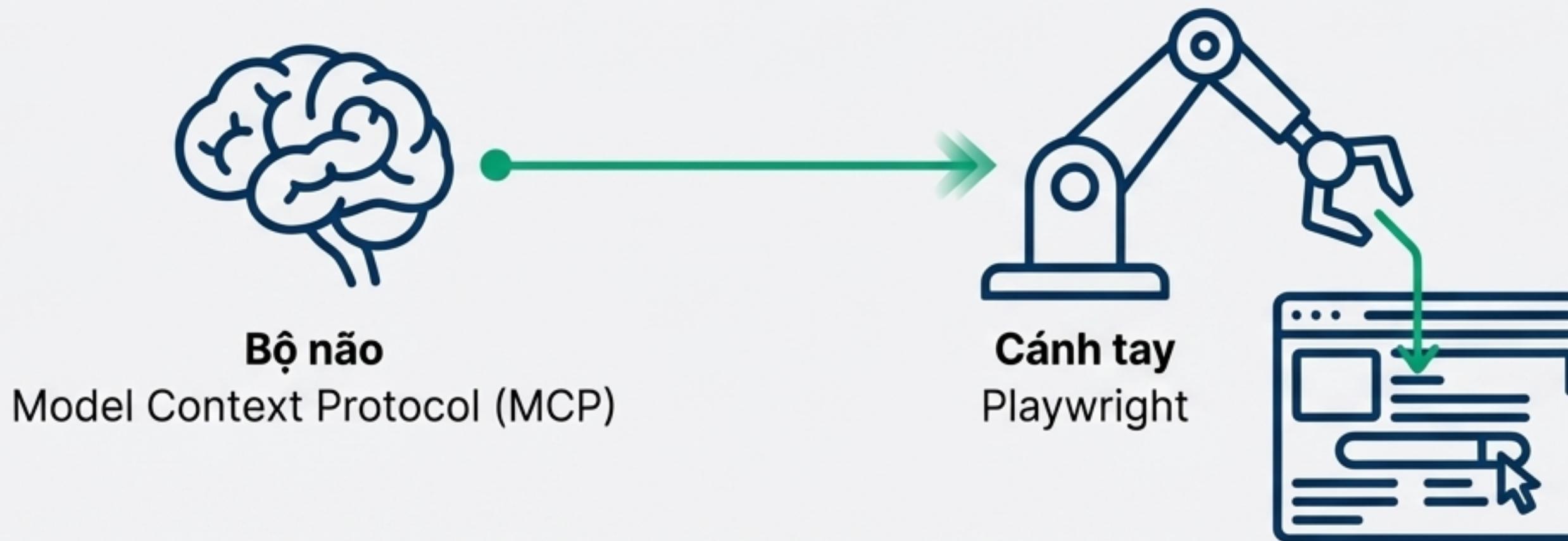
Tìm vật thể có hình dáng ‘cái ghế’. Tiến lại gần nó. Ngồi xuống.

Ưu điểm vượt trội:

Dù cái ghế bị dời đi đâu, miễn là nó vẫn là ‘cái ghế’, người đó sẽ tìm và ngồi được. Test sẽ ‘PASSED’.



Công Nghệ Đằng Sau Phép Màu: Playwright MCP và Kỷ Nguyên ‘Agentic Automation’



Agentic Automation (Tự động hóa dựa trên tác tử): Một **mô hình kiểm thử mới**, nơi một “**tác tử**” AI thông minh tương tác với ứng dụng dựa trên ngữ nghĩa và mục tiêu, thay vì các script cứng nhắc.

Sự kết hợp:

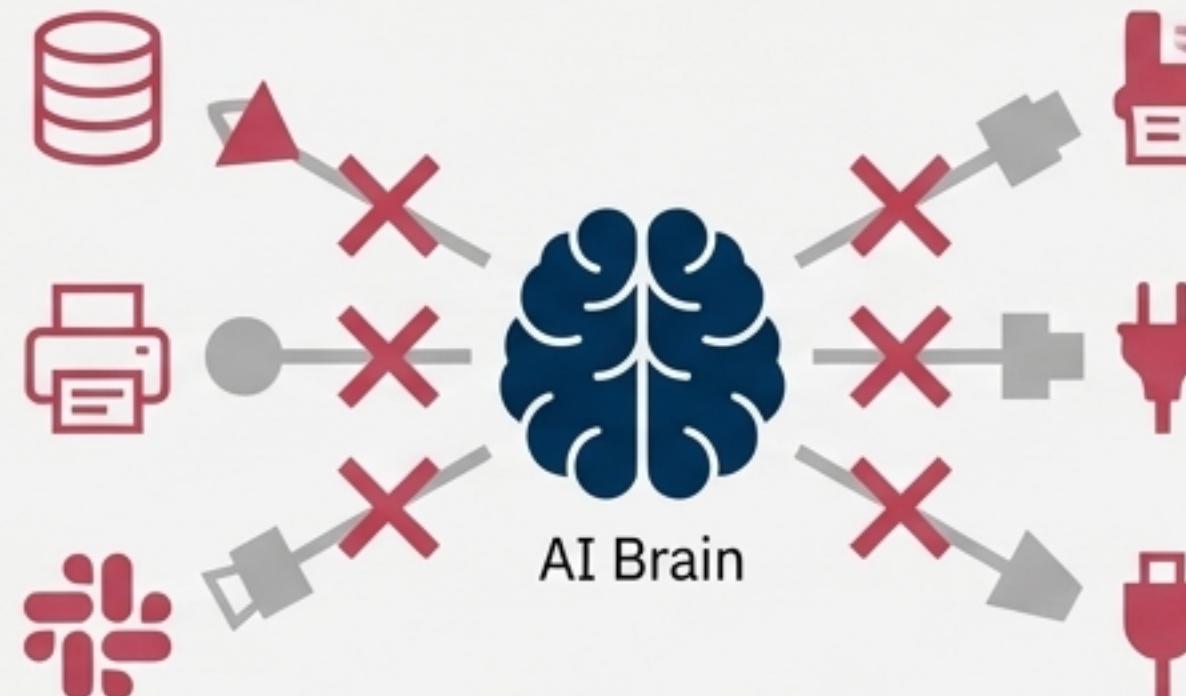
- **Playwright:** Cung cấp ‘cánh tay’ để tương tác với trình duyệt.
- **Model Context Protocol (MCP):** Cung cấp ‘bộ não’ AI để hiểu và ra quyết định.

MCP Là Gì? Hãy Tưởng Tượng Nó Là Cổng USB-C Dành Cho AI

Định nghĩa Kỹ thuật (Technical Definition): MCP là một tiêu chuẩn mở (open standard) giúp các mô hình ngôn ngữ lớn (LLM) kết nối và điều khiển các công cụ bên ngoài một cách thống nhất.

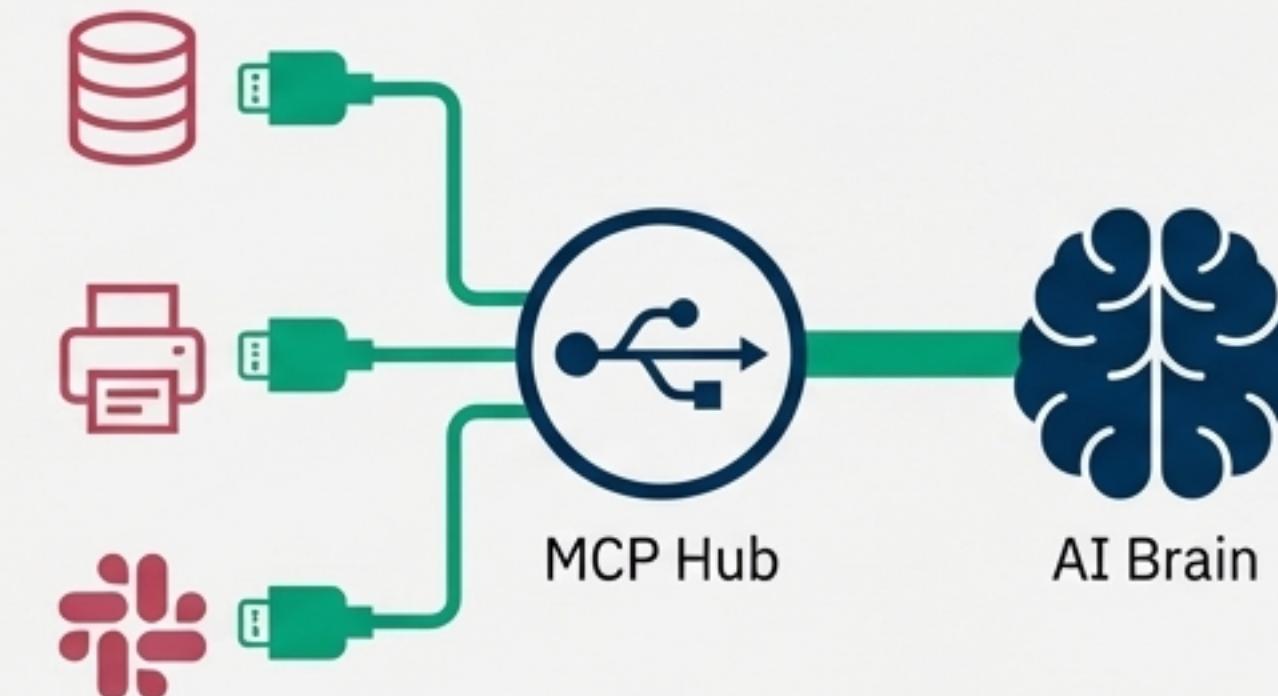
Trước kia (Không có MCP)

Giống như mỗi thiết bị cần một cổng riêng (máy in cổng LPT, chuột cổng PS/2). Mỗi AI phải viết code riêng để kết nối với từng công cụ.

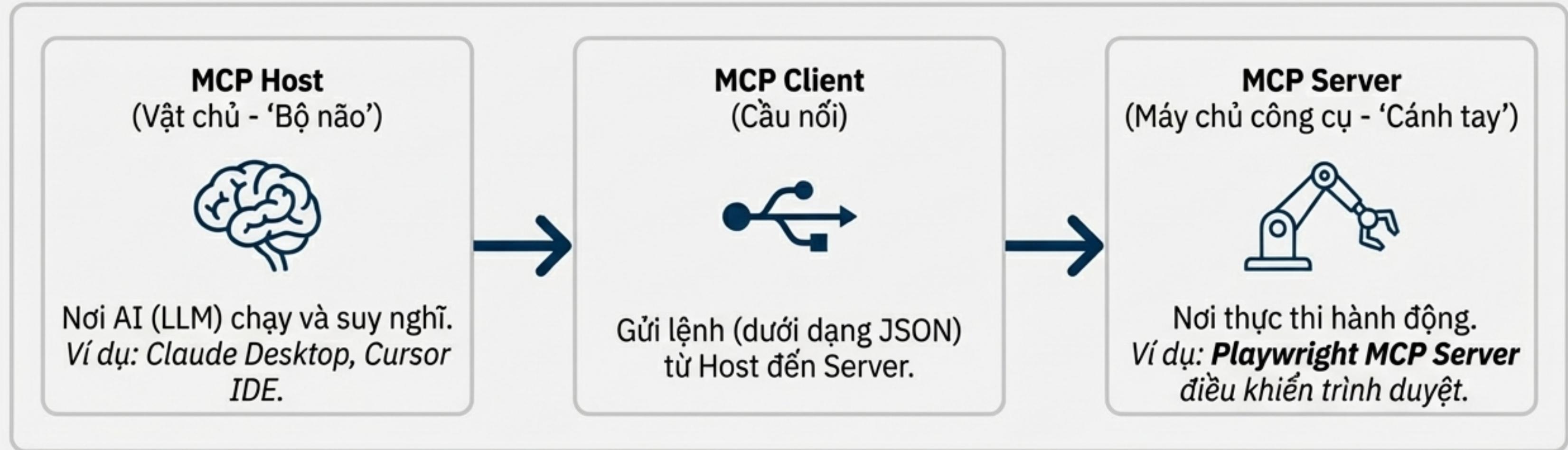


Bây giờ (Với MCP)

Tất cả đều cắm vào một cổng USB-C duy nhất. MCP là cổng chung đó cho AI, cho phép nó điều khiển Playwright, Database, Slack,... một cách dễ dàng.



Kiến Trúc Hoạt Động: Bộ Não, Cánh Tay, và Cầu Nối



Luồng hoạt động (Workflow):



AI ‘Nhìn’ Giao Diện Bằng Cách Nào? Cuộc Đổi Đầu Giữa DOM và Accessibility Tree

DOM (Document Object Model)

Là bản vẽ kỹ thuật chi tiết của trang web. Chứa đầy các thẻ `<div>`, ``, `class='wrapper-v2-flex'`. Rất phức tạp và dễ thay đổi.

```
<div class="wrapper-v2-flex">
  <div class="container-456">
    <span class="text-muted">Name:</span>
    <div class="input-group">
      <input type="text" class="form-control-3" id="userName">
    </div>
    <div class="actions-row">
      <button type="submit" class="btn-primary-99">Submit</button>
      <a href="/help" class="link-secondary">Help</a>
    </div>
  </div>
</div>
```



Accessibility Tree (A11y Tree)

Là bản tóm tắt ý nghĩa dành cho người khiếm thị (hoặc AI). Loại bỏ các chi tiết không cần thiết, chỉ giữ lại những gì người dùng quan tâm: Nút bấm (Button), Liên kết (Link), Ô nhập liệu (Input).

- Input: "Name"
- Button: "Submit"
- Link: "Help"



AI không ‘đọc’ code, nó ‘hiểu’ ý nghĩa của giao diện thông qua A11y Tree. Đây là chìa khóa của sự ổn định.

Tình Huống Thực Tế: Kịch Bản ‘Mua Vé Xem Phim Mai’

- **Bối cảnh (Context):** Chúng ta cần viết một bài test tự động cho chức năng đặt vé phim ‘Mai’ trên một trang web.
- **Cách làm cũ (Script-based Approach):** Dựa vào vị trí và thuộc tính cố định của các phần tử.

```
// Click phim ở vị trí thứ 3 trong danh sách  
await page.click('#movie-list > div:nth-child(3)');
```

```
// Chọn ghế B4 có sẵn  
await page.click('.seat-available[data-id="B4"]');
```

```
// Nhấn nút thanh toán  
await page.click('#btn-pay-final');
```



- **Điểm Yếu Chí Mạng (The Critical Flaw):** Ngày mai, rạp chiếu phim cập nhật, đẩy phim ‘Mai’ lên vị trí số 1. Script sẽ click nhầm phim khác.

Cách Làm Mới: Agent Tự Tìm Đường

Phương pháp (Method):

Ra lệnh bằng ngôn ngữ tự nhiên, mô tả chủ đích.
IBM Plex Sans

Prompt đưa cho Agent:

> “Hãy tìm phim tên là ‘Mai’. Chọn một ghế trống bất kỳ ở hàng giữa. Sau đó nhấn thanh toán.”

Quá trình Agent thực thi (Agent's Execution Process):



Quan sát (Observe): ‘Tôi sẽ quét Accessibility Tree để tìm văn bản có chứa chữ ‘Mai’.’



Hành động 1 (Act 1): ‘Aha, tìm thấy rồi. Tôi sẽ click vào Link có tên ‘Mai’, bất kể nó nằm ở vị trí nào.’



Quan sát tiếp (Observe Again): “Bây giờ tôi thấy sơ đồ ghế. Tôi sẽ tìm các Nút (Button) có trạng thái ‘Available’ và tên chứa ‘E’ hoặc ‘F’.”



Hành động 2 (Act 2): ‘Chọn một ghế phù hợp và nhấn nút ‘Thanh toán’.’

ROBUST

Kết quả: Test luôn **PASS** dù thứ tự phim hay mã ghế thay đổi.

Mã Nguồn Minh Họa: Xây Dựng Một ‘QA Agent’

Đoạn mã Python sau mô phỏng tư duy của một Agent đơn giản sử dụng các nguyên tắc của Playwright MCP.

```
# Giả lập tư duy của MCP Agent
async def run_qa_agent(goal):
    print(f"🎯 MỤC TIÊU: {goal}")

    # Bước 1: Agent quan sát (Dùng A11y Tree)
    page_content = await page.evaluate("document.body.innerText")

    # Bước 2: Agent suy luận & Hành động
    if "Đăng nhập" in page_content:
        print("🤖 AI: Tôi thấy nút Đăng nhập. Tôi sẽ click nó.")
        # Sử dụng getByRole - Chìa khóa của sự ổn định
        await page.get_by_role("button", name="Đăng nhập").click()

    # Bước 3: Tự kiểm tra (Verification)
    if await page.get_by_text("Chào mừng quay lại").is_visible():
        print("✅ KẾT QUẢ: Test Passed!")
    else:
        print("❌ KẾT QUẢ: Test Failed!")
```

Chú ý việc sử dụng `get_by_role` và `get_by_text`. Đây là cách giao tiếp với trình duyệt dựa trên vai trò và nội dung mà người dùng nhìn thấy, không phải cấu trúc DOM ẩn bên dưới.



Phân Tích Hiệu Năng: Liệu ‘Thông Minh Hơn’ Có Đồng Nghĩa Với ‘Chậm Hơn’?

☰ Tiêu chí	Dùng DOM Selector (Cũ)	Dùng A11y Tree (Mới)	Đánh giá
🔍 Tốc độ tìm	~0.01 mili-giây	~0.05 mili-giây	Chậm hơn không đáng kể
🛡 Độ ổn định	Thấp (Dễ vỡ)	Cao (Tự thích nghi)	**Vượt trội**
💼 Chi phí Bảo trì	Cao (Tốn hàng giờ)	Gần như bằng 0	**Vượt trội**

Sự đánh đổi vài mili-giây không thể so sánh được với lợi ích khổng lồ về độ ổn định và chi phí nhân sự để bảo trì hệ thống.

Đây Không Chỉ Là Công Cụ Mới, Mà Là Một Sự Thay Đổi Về Tư Duy

Từ IMPERATIVE sang DECLARATIVE



Cũ: ‘Dạy máy làm thế nào.’
(Click `div` này, nhập text
vào `input` kia.)



Mới: “Bảo máy làm cái gì.”
(Đăng nhập với tài khoản
X.)

Từ IMPLEMENTATION sang USER EXPERIENCE



Cũ: Bắt lỗi dựa trên chi
tiết kỹ thuật (code).



Mới: Bắt lỗi dựa trên trải
nghiệm thực của người
dùng.

Lộ Trình Áp Dụng: Bắt Đầu Với Agentic Automation Như Thế Nào?

Bắt đầu ngay hôm nay

Ứng dụng: Sử dụng Playwright MCP cho Exploratory Testing (Kiểm thử Thăm dò).

Mục tiêu: Để các agent tự do ‘khám phá’ ứng dụng và tìm ra những lỗi ngẫu nhiên mà kịch bản cố định thường bỏ sót.



Giai đoạn 1

Tích hợp sâu hơn

Ứng dụng: Dùng agent để tự động sinh dữ liệu test (Test Data Generation) hoặc triển khai cơ chế ‘Tự sửa lỗi’ (Self-healing) cho các test case truyền thống bị thất bại.



Giai đoạn 2