MACQUARIE
University

"Never Do User Testing Again

Automated UI Testing for Astronomical Web Tools

Mrunmayi Deshpande.

Mrunmayi.Deshpande@mq.edu.au

Australian Astronomical Optics

Macquarie University

Overview

- Automated UI testing:
  - why it matters?
  - what tools we used?
  - what we learned?

Astronomy Web's Unique Challenges

Testing web applications in an automated way is difficult.

- Diverse infrastructure
- Legacy + modern code mixed
- Small dev teams
- Long-lived software

Critical Failure Points

- Data Exploration
  - UIs for queries, cutouts, and analysis are complex. A bug here stops research.
- Pipeline Configuration
  - Errors in web forms, pipelines lead to failed jobs and wasted resources.
- Diverse Environments
  - Chrome, Firefox, Safari and platforms. Consistency is essential
- Increase User expectation
- Increasing web interface Complexity

Visual web testing does not work

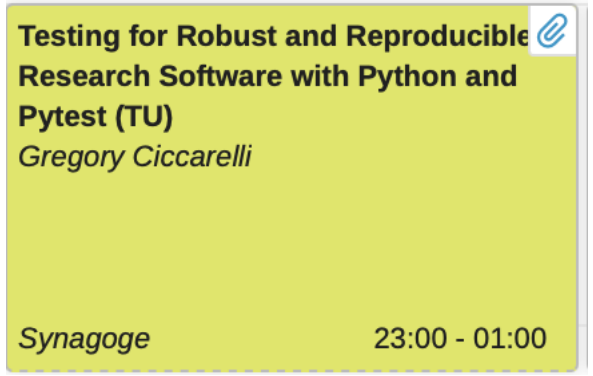
- “Blink" comparison tools and visual inspection cannot be used for QA.

UIs Are Critical for astronomy

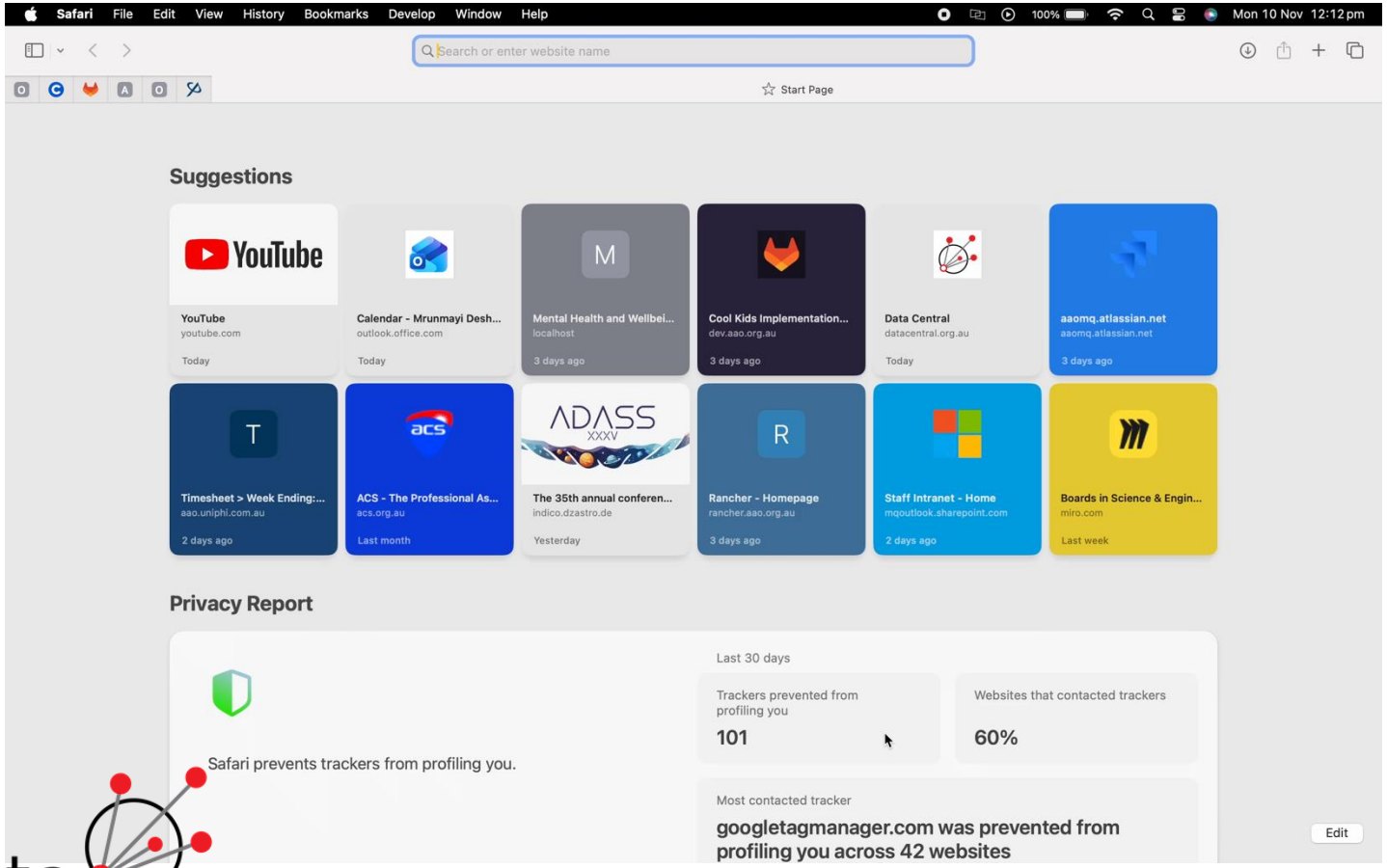
- Internal tools:
  - pipeline configuration, QA, dashboards
- Public-facing tools:
  - data portals, visual explorers, archives, VO services

Mistakes cost time, trust, and scientific accuracy

So how do we test these?



Manual UI testing

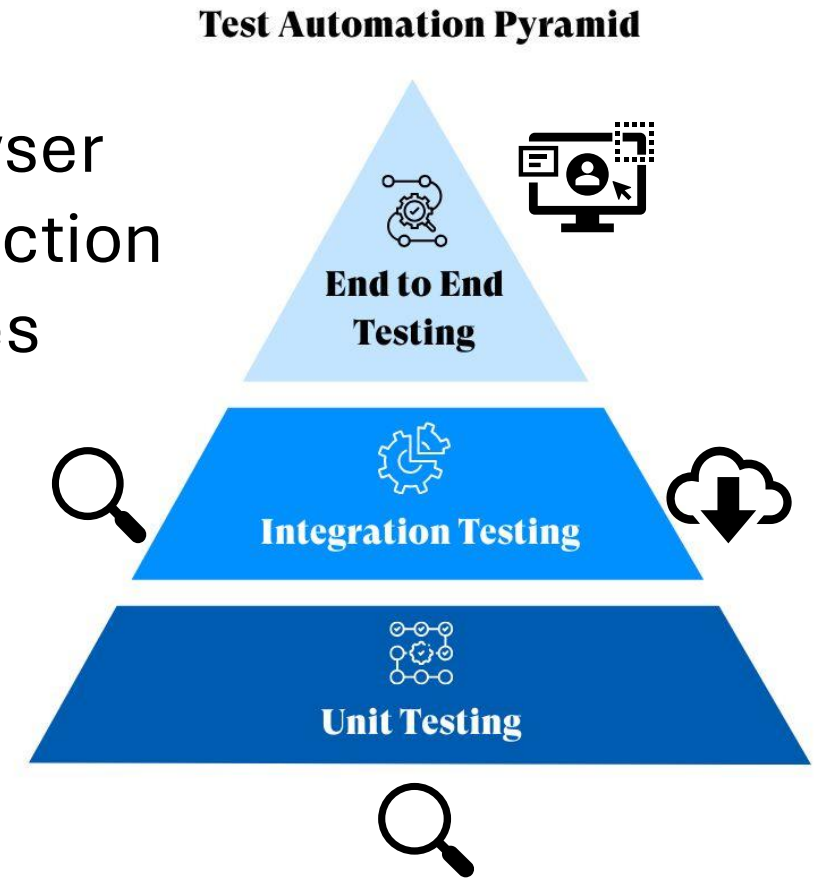


- Essential
- Slow
- Inconsistent
- Not scalable

We need a systematic, automated approach

What Is E2E Testing?

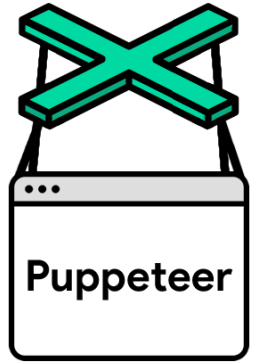
- Simulates real user interactions in a browser
- Detects regressions before they hit production
- Scales across browsers, OSs, screen sizes
- Critical for CI/CD pipelines



Which test automation tool should you choose?

Features:

- Architecture
- Speed & Stability
- Setup
- Parallel Testing for efficiency
- Network Mocking
- Debugging
- Cross-Browser support



Which UI automation for DataCentral?

Faster & more reliable:
No WebDriver layer → less maintenance

Smart auto-waits:
Handles dynamic pages

Unified cross-browser API
Test Chrome, Firefox & Safari easily

Built-in mocks:
Simulate API responses for datasets

Better debugging:
Visual trace, screenshots, and logs

Perfect for CI/CD:
Headless mode, runs in containers

Points	Selenium	Playwright
Features	WebDriver API and HTTP requests	WebSocket connection
	Layered Architecture based on JSON Wire Protocol but also support Headless Browsers	Headless Browser with event-driven architecture
Prerequisites	Java, Eclipse IDE, Selenium Standalone Server, Client Language Bindings, and Browser Drivers should be installed	NodeJS should be installed
Browser Support	Chrome, Firefox, Safari, Edge, Opera	Chromium, Firefox, and WebKit
Developer Experience		modern developer experience with features like automatic waits, built-in reporting, and debugging tools, making it easier to write and maintain tests
Speed	Comparitavely slower	WebSocket-based communication and optimized faster test execution
Supported Languages	Java, Python, C#, Ruby, PHP, JavaScript	JavaScript, TypeScript, Python, C#, and Java
Real device Testing	supports real device testing through Appium. Native mobile emulation (and experimental real Android support)	Real device clouds and remote servers supports mobile emulation and experimental real Android device support
Community Support	larger and more established community	growing backed by Microsoft
OS supported	Windows, Mac OS and Linux, Solaris	Windows, Mac OS and Linux
Features		TRACE VIEWER

Code and demo

```
4   test('navigate to VO → TAP and verify correct page (with video)', async () => {
20
21    // Test flow
22    await page.goto('https://datacentral.org.au/');
23
24    // Accept cookies if they appear
25    const agree = page.getByRole('button', { name: /I agree/i });
26    if (await agree.isVisible({ timeout: 5000 })) {
27      await agree.click();
28    }
29    // Hover or click the VO dropdown
30    const voLink = page.locator('a.navbar-link.has-text-weight-bold.is-arrowless', { hasText: 'VO'
31    await expect(voLink).toBeVisible({ timeout: 10000 });
32    await voLink.hover();
33    // Click TAP inside the dropdown
34    const tapLink = page.locator('div.navbar-dropdown a.navbar-item', { hasText: 'TAP' });
35    await expect(tapLink).toBeVisible({ timeout: 10000 });
36    await tapLink.click();
37
38    // Assert URL
39    await expect(page).toHaveURL('https://datacentral.org.au/vo/tap', { timeout: 10000 });
40
```

```
65
66    const path = await download.path();   // Verify download happened
67    expect(path).not.toBeNull();
68    const filename = await download.suggestedFilename();
69    console.log('Downloaded filename:', filename);
70    expect(filename).toBeTruthy();
71
72    await context.close();    // Close context and browser so video is finalized
73    await browser.close();
74  });
75
```

```
45    // Assert URL
46    await expect(page).toHaveURL('https://datacentral.org.au/vo/tap', { timeout: 10000 });
47
48    // Locate the textarea by name and check attributes + default value
49    const queryTextarea = page.locator('textarea[name="QUERY"][cols="80"][rows="5"]');
50    await expect(queryTextarea).toBeVisible({ timeout: 10000 });
51
52    // Check the value exactly as it appears
53    await expect(queryTextarea).toHaveValue(
54    `SELECT *
55  FROM TAP_SCHEMA.tables;`
56    );
57    // Click the Execute button (using the exact selector)
58    const executeBtn = page.locator('input#submit[type="submit"][value="Execute!"]');
59    await expect(executeBtn).toBeVisible({ timeout: 10000 });
60
61    const [ download ] = await Promise.all([
62      page.waitForEvent('download'),
63      executeBtn.click()
64    ]);
```

Wins from Automation at DataCentral

Playwright in Our CI/CD Pipeline

- Every pull request and commit triggers **backend + UI tests** in **headless browsers**.
- Runs automatically in **GitLab CI** across **Chromium, Firefox, and WebKit**.
- Failures produce **screenshots and video traces** for quick debugging.
- **Mocked services and lightweight fixtures** handle authentication and data setup

Why It Matters

- **Catches regressions early** — before merge
- **Fewer bugs** reach production
- **Developers trust the pipeline** and refactor confidently
- **Faster feature cycles** and more stable releases

Why Should You Care?

"If Data Central's complex, data-rich UI and web services can be reliably tested with Playwright — any web platform can benefit from it."

- **faster test runs**, **increased realibility**, and **easier maintenance**.
- Unified testing for UI + APIs
- Developer and Tester efficiency
- High reliability and maintainability
- Low cost and learning curve

Never Do Manual User Testing Again Takeaways

- Need some manual test and add automated UI tests on top
- Playwright is an important tool kit but not everything
- Start Now
- Build User Trust
- Doesn’t matter which web testing tool kit you use.
- UI testing of development and QA -> time to do other work.

Questions

-Thank you

- Mrunmayi Deshpande.
- Australian Astronomical Optics, Macquarie University

MACQUARIE University