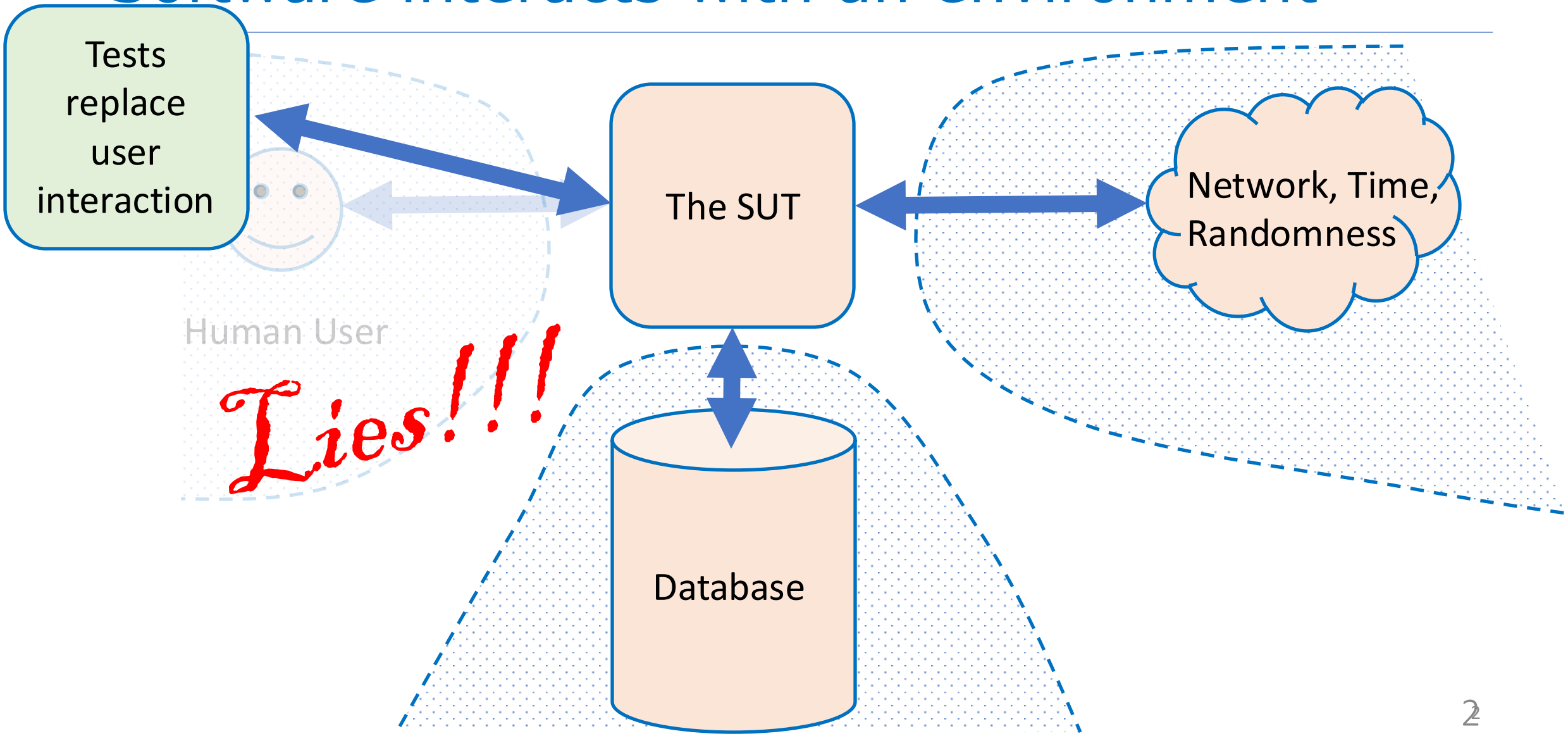
CS 4530: Fundamentals of Software Engineering
Module 5, Lesson 5
Testing Web Applications

Rob Simmons

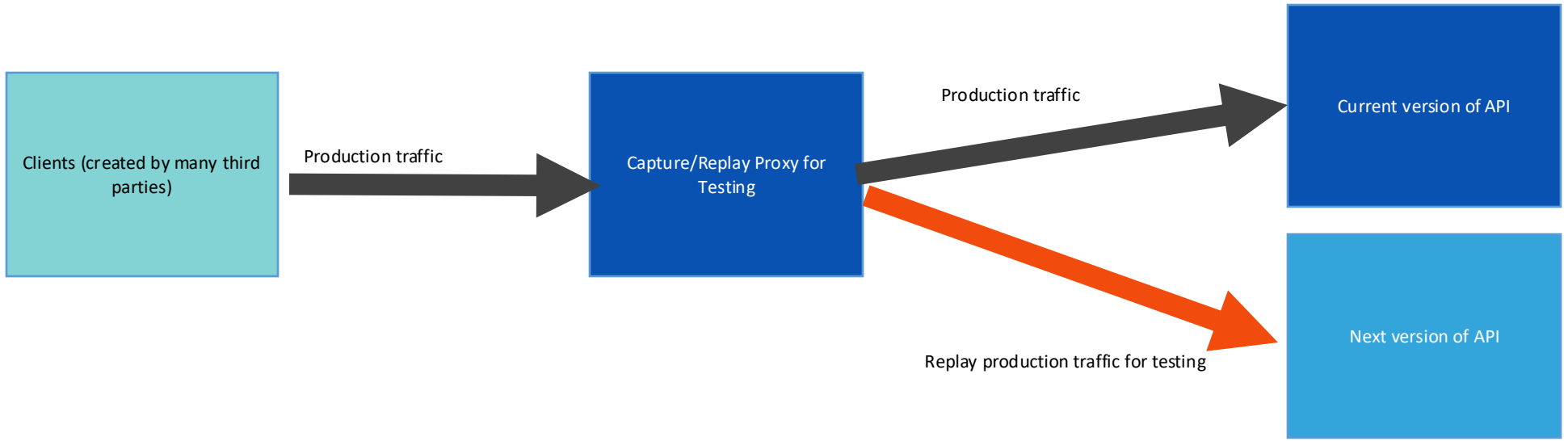
Khoury College of Computer Sciences

Software interacts with an environment



Mock System-Level Components with Capture/Replay

- Record the API requests and responses that clients make
- Test new versions of the API by identifying requests that result in different responses ("breaking changes")



https://www.tradeweb.com/our-markets/data--reporting/replay-service/

Acceptance Tests can be formulated as scenarios

- Acceptance tests are written to verify behavior from a user's perspective.
- The focus is on treating the application as a black-box
- Tests may be specified as given-when-then scenarios:

  given there's a logged in user
  and there's an article "bicycle"
  when the user navigates to the "bicycle" article's detail page
  and clicks the "add to basket" button
  then the article "bicycle" should be in their shopping basket

https://docs.cypress.io/guides/end-to-end-testing/writing-your-first-end-to-end-test

Deployed systems create testing challenges

- Clients believe “how it is now is right”,
  - Not “how the API intended it to be is right”
  - Writing thorough test suite is even harder, less useful
  - What is a “breaking change”?
- Still: vital to detect breaking changes
- Examples:
  - Detailed layout of GUIs
  - Side-effects of APIs, particularly under corner-cases

But how to make these human-readable scenarios into executable tests?

- Scenarios like the one above are readable by humans (e.g. customers)

But how to make these human-readable scenarios into executable tests?

- Automation is a *spectrum* — humans can automate tests!
  - Many of the same concerns apply: deterministic, not flaky…

6 **+10.0**

Testing:

- `npm ci && npm run build -w=client && npm start -w=server`
- load user0, user1, user2 in separate browsers
- user0 and user1 send chats in top game
- user0 leaves
- user2 enters and chats (should see chat history)
- user0 enters (should see chat history)

https://www.browserstack.com/guide/test-case-vs-test-s

Check for four text boxes to accomodate the 16 digit credit/debit card number	Enter 4 digits in a text box and check for aut movement of curson to next text box			
	Check with a 16 digit card number		digits in each text box.	each box. A	
	Check with an invalid 20 digit card number	4 text boxes to be displayed and only 16 digits are to be accomodated, 4 digits in each box. The user should not be able to enter the rest of the digits.	4 text boxes are displayed and 16 digits are accomodated, 4 digits in each box. The extra 4 digits are not recorded	Pass
		4 text boxes to be displayed, 13 digits are to be accomodated, 4 digits in first three boxes. The last	4 text boxes are displayed, first 4 text boxes accommodate 4 digits	

But how to make these human-readable scenarios into executable tests?

Entire companies specialize in this endeavor (https://www.digitaldreamforge.com/our-services/)

DIGITAL DREAMFORGE

We provide these services…

Functionality Testing

- Test Plan Creation and Running
- Full-Cycle Testing
- Regression Testing
- Network Testing
- Launch Verification Testing

Entire sub-specialization within software engineering to push automation faster/simpler/more reliable to meet agile & CI/CD goals

QA Engineer Lead Responsibilities

- Plan, develop and execute product quality strategies for Meta's products
- Develop test strategies for business critical projects to ensure product correctness before launch
- Manage a team of in-house and offshore testers to conduct black box testing
- Spearhead initiatives that influences engineering organizations to build a quality-driven approach
- Partner with engineering and infrastructure teams to leverage automation for scalable solutions to prevent

But how to make these human-readable scenarios into executable tests?

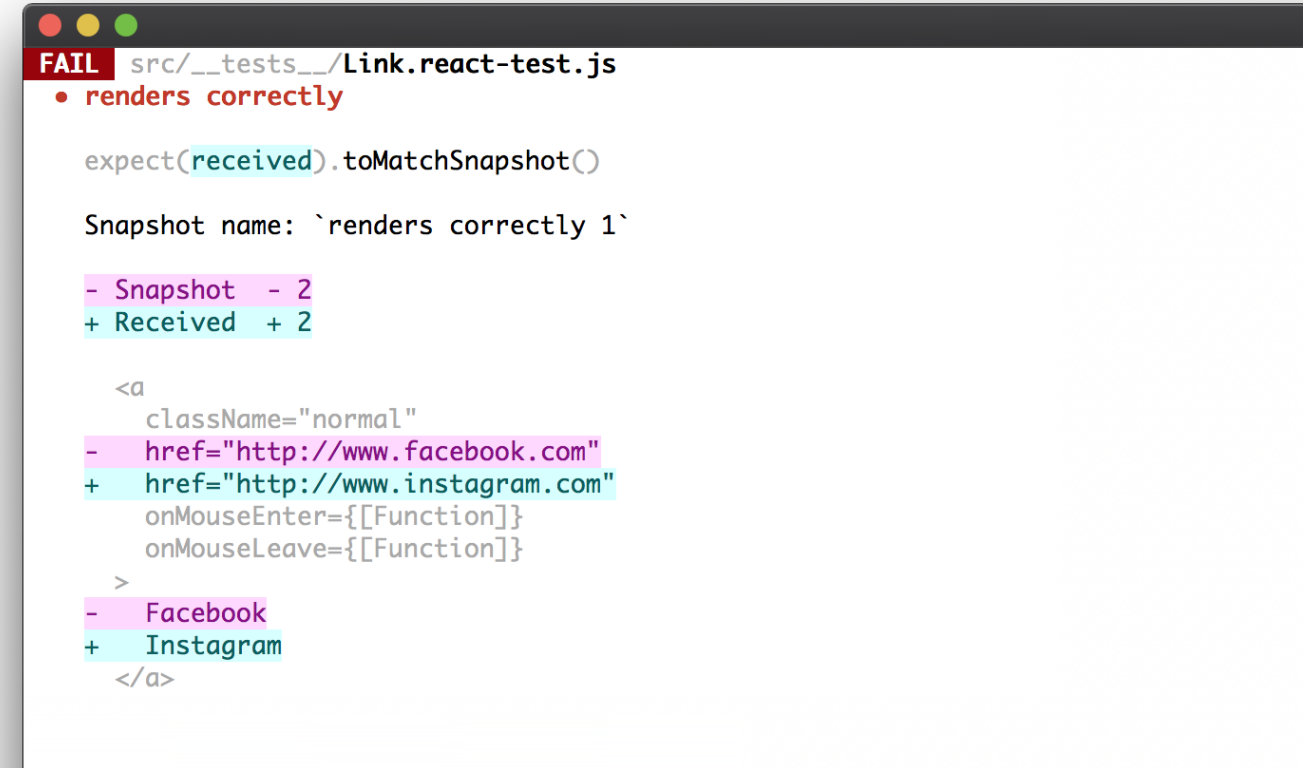
- They are not automatable are not automatable
- Lots of tools to fill the gap: I'll add links to the lesson page

Snapshot Tests Can Detect GUI Changes

- The first time the test runs, it saves a "snapshot" of the rendered GUI
- Subsequent runs will fail if the snapshot changes

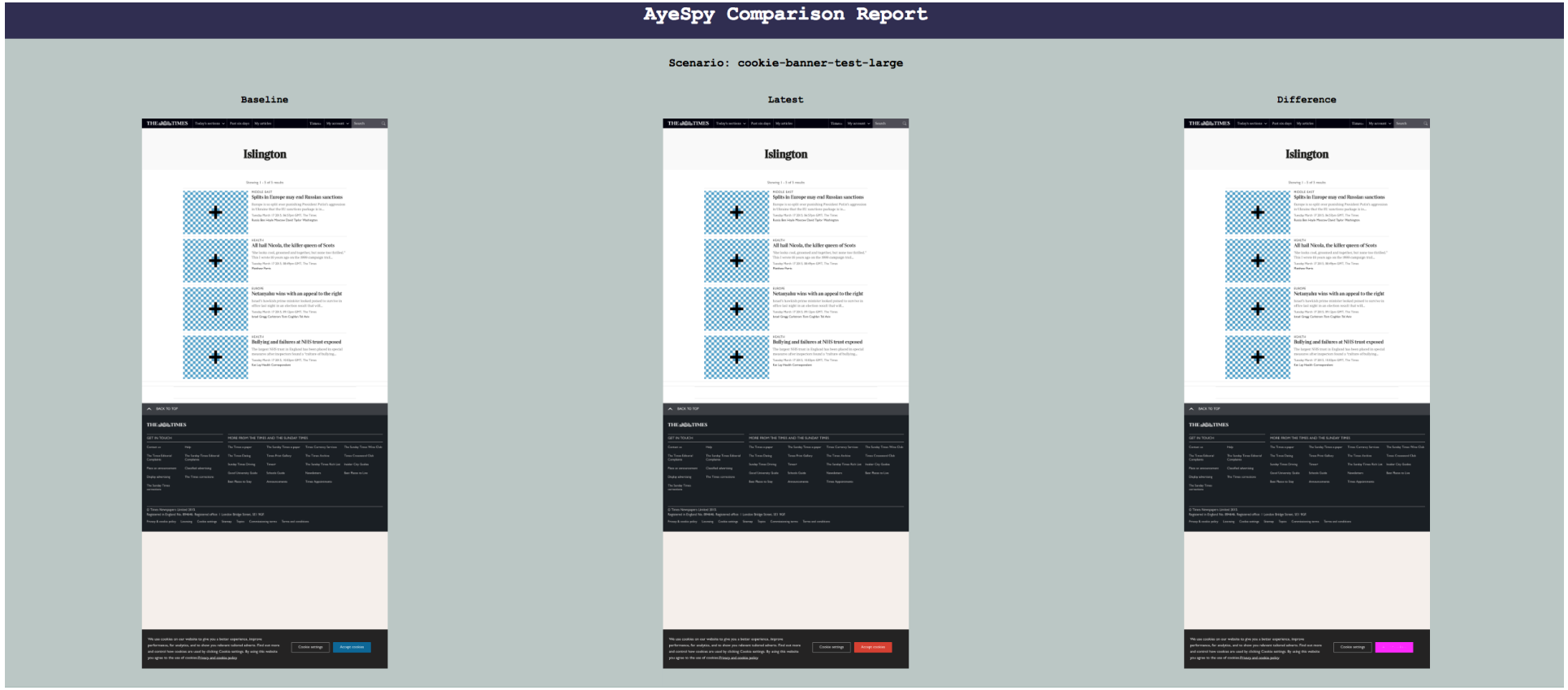
```
import renderer from 'react-test-
renderer';
import Link from '../Link';

it('renders correctly', () => {
  const tree = renderer
    .create(<Link
page="http://www.facebook.com">Facebook</L
ink>)
      .toJSON();
  expect(tree).toMatchSnapshot();
});
```



Product Owners can Assess Visual Snapshot Tests

- Capture a visual snapshot of an application under a state
- If that snapshot changes, produce a visual report for manual sign-off



https://github.com/newsuk/AyeSpy

Terms worth knowing!

- **Snapshot testing** — doing particular actions and storing the rendered HTML/DOM
- **Visual testing** — doing particular actions and storing the rendered pixels

How to perform actions? No good answers only tradeoffs

- Click these particular coordinates
- Search for a button with this user-visible text in it
- Search for a button with this id or user-invisible quality