# Automating Checking - Phase 2 - Test Automation Strategy

## Checking

### Testing Terminology

- Planning test activities
- Designing test cases
- Exploring a product
- Adapting an approach
- Evaluating the experience from an end-user's perspective
- Running tests and looking at the outcomes...
    - this one is deliberately kinda hard

a lot of this is what we have done before - with playwright

Automation can find it hard to do some things that exploratory testing can achieve

Is automation "thinking" or is it just following instructions it has been given?

### How is this different to 'checking'

since automation is not able to 'think' for itself, it is more favorable to refer to this as 'checking', in that the computer is checking certain things exist or match or whatever - it is primed by humans to know what to do in this case

A human following predefined steps is also 'checking'

### Testing vs checking

Below are some activities - for each, determine whether you think it's "testing" or "checking".

- Running unit tests with defined asserts
    - checking
- Creating a mind map of test areas for an application
    - 'testing' - planning

- Assessing whether a web page is accessible for colour blind users
  - Checking - maybe testing as well if you get someone to look at this
- Exploring a product's functionality
  - Testing
- Determining whether an observed behaviour is a bug
  - Testing
- Evaluating and assessing a risk
- Executing a planned test
- Running a script that looks for regressions

What other activities can you think of or that you've done in recent weeks that aren't listed above, and what labels would you give those?

## Automation

- this is a really good thing to have and can have many benefits - but has lots of setup and can lead to bad tests
- is it worth it? Will you need to keep retesting, or just a one-off?
- Just because you can, does that mean you should?
- Value is added over time

# Evaluating Tests for Automation

## How to evaluate?

maybe work out the time?

```
auto_time = Time to write automation + (Number of times executed * Time to execute)

E.g. auto_time = 60 minutes + (10 * 1 minutes) = 70 minutes

manual_time = Number of times tests run * Time to run tests and evaluate

E.g. manual_time = 10 * 20 = 200 minutes
```

This may work - may not, maybe they change the app and the autotests break - then the time ramps up

## Regression testing

Regression testing can performed at any time and can apply to things like components or the interfaces between components, up to entire systems with end-to-end tests. It's used as a way to check for regressions, which are essentially unintended side-effects of changes to any part of the system or even the environment in which it is run. When the regression tests are passing, that gives us an increased level of confidence about recent changes and the thing(s) against which we've run the tests.

With frequent changes to the code and/or as a system grows, checking for regressions becomes increasingly important and hence so does regression testing. While it's all well and good verifying that a fix made as a result of a bug being raised has addressed the original defect, if we've caused undesirable side-effects to other functionality elsewhere, we want to know about this as soon as possible.

When testing for regressions becomes frequent and time-consuming, as you might expect, it's wise to consider automation.

## Maintenance

usually automated
can slow down over time as stuff is added

Maintenance of regression test suites is, therefore, a frequent consideration and there are some things you can do to make things easier.

- When adding a test, add some justification or reasoning. Sure, it makes sense *now* but it might not do in a year's time or were someone else to be reviewing it wondering "Do we really need this test?"
- Regular review and action can help prevent regression test suites from becoming stale, unfamiliar and losing their value.
- Questioning whether a test should stay in if it has never failed can provoke some interesting debate. Sometimes, tests can be erroneously added (see previous sections and modules on human error) and have no value or aren't checking what they seem to be checking. Sometimes, a test can be perfectly valid but it's simply testing something that has never failed. Caution though: if it were to suddenly fail, how big of a problem would that be? (Hint: think about *risk assessment*.)
- Optimisation - is there a better way (like a new interface or environment or technology) that can be used which will speed up the tests? Optimisation of a regression test suite can postpone or remove difficult discussions around what to cut from a suite that's considered to be too slow or bloated.

# Test Pyramid

## Unit tests

- testing one - we know this stuff - usually down to the dev

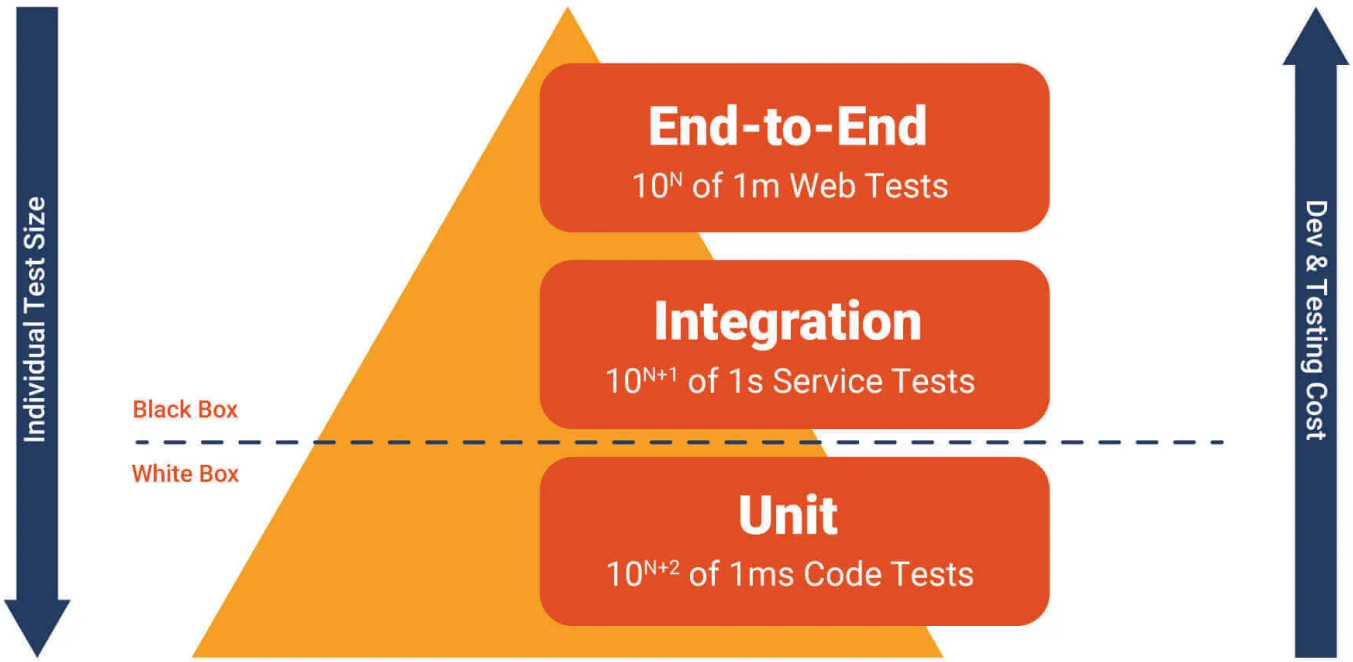
## Integration tests

- again we kinda know this - testing multiple components with eachother - sometimes the dev sometimes testers - depends on the company

## End to end/ui/etc tests

They tend to include operations which a real user might do (hence the name 'user acceptance tests') and could be browser-driven, for example, if the product is browser-based. This type of test can be run using a tool like Playwright or Selenium.

At this top level, tests tend to be harder to set up, because you have to get the application running, and slower to execute. They do, however, have the benefit of being closer to real use situations and they also check how the system works as a whole.




- The **width** of the pyramid roughly represents the amount of tests at each level i.e. more unit tests, fewer end-to-end/UI tests
- The **time** to run tests decreases as you move from the top of the pyramid to the bottom, from slow and more expensive to run end-to-end/UI tests, down to quick and cheaper unit tests

- Tests at the bottom (unit tests) are more isolated than those above which require more integration in order to be able to run as you move up the pyramid

## Reviewing for Automation

### Challenge part 1

For each of the hypothetical tests below, decide whether they are suitable candidates for automation, rather than running the tests manually and having a human checking the outcomes and behaviours.

Checking whether...

- (1) An addition function handles adding a positive integer to a negative integer
- (2) The "happy path" through the product which the CTO plans to demo in an hour is working as expected
- (3) A program returns the first 1000 digits of pi correctly
- (4) A website renders appropriately in Safari, Chrome, Firefox, Edge, Vivaldi and Brave
- (5) A workaround for some functionality is effective, knowing it's to be rewritten and replaced with something proper in two weeks time
- (6) An old, deprecated (and soon to be removed) bit of functionality still works, when a newer method for that functionality already exists
- (7) A website's dark/light theme can be applied and changes the site
- (8) The website's background colour is pure white (i.e. in hex `#FFFFFF`) not off-white/etc.
- (9) A website's colour contrast is appropriate for users with different types of colour blindness
- (10) The system fails gracefully when put under extreme load, such as having an excessive number of users in parallel
- (11) The Copyright symbol and version number are printed in the About dialog of each daily build of the product
- (12) A mobile phone displays a helpful number of WiFi signal bars as the phone moves nearer to/further from the router
- (13) The system is still responsive under a normal/expected load of users using it at the same time
- (14) The accompanying manual pages for the newly added feature accurately describe how to use it

Write down a heading for each group (e.g. "Automate" & "Don't Automate"), with the hypothetical tests, and/or their Ids, below the appropriate heading. Are all the decisions clear cut or are some trickier than others? Add annotations where appropriate e.g. if decisions are tricky, if you've added more than two group categories, or any other things you particular feel are noteworthy in some way.

You'll submit this write-up of groupings to your coach, along with materials from the second part of the challenge below.