

Automated Testing with Playwright



Automated Testing



<https://playwright.dev/>



Playwright

Testing framework from Microsoft
Support modern web browsers
Headless by default
Cross platform and language



History



2005

2010

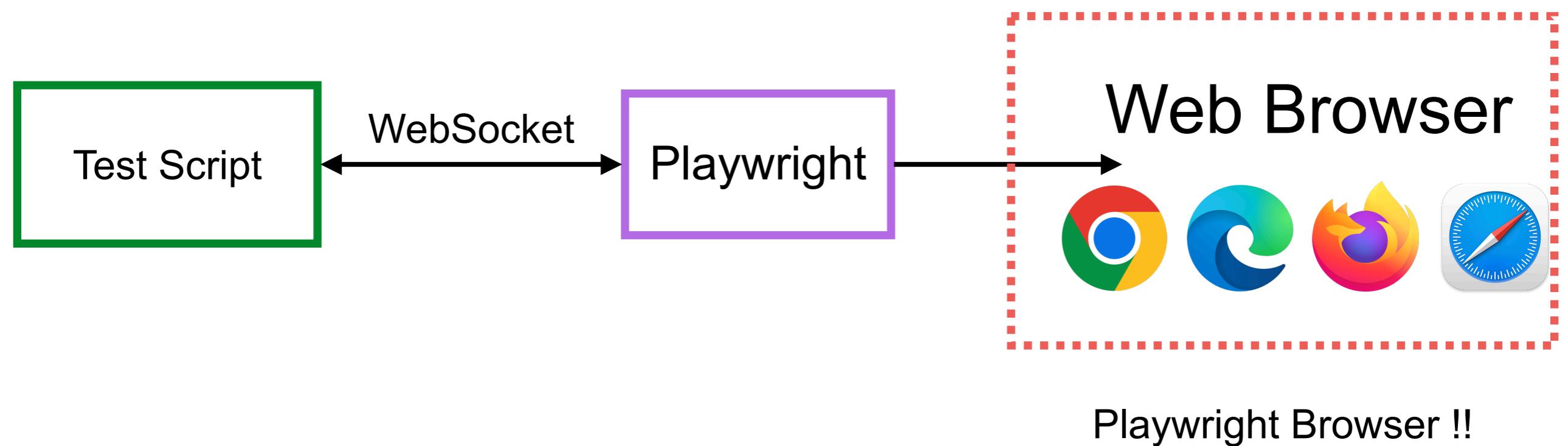
2017

2020

2023



Architecture



<https://playwright.dev/docs/browsers>



Playwright Testing

Web Browser
Test

Visual Test

API Test



Setup for Playwright



Start with Playwright

\$npm init playwright@latest

Node.js® is an open-source, cross-platform JavaScript runtime environment.

Download Node.js®

20.9.0 LTS
Recommended For Most Users

21.1.0 Current
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

<https://nodejs.org/en>



Run test

\$npx playwright test

```
Running 6 tests using 6 workers  
6 passed (11.9s)
```

To open last HTML report run:

```
npx playwright show-report
```



Test Report

\$npx playwright show-report

All 6	Passed 6	Failed 0	Flaky 0	Skipped 0
10/31/2023, 8:27:17 PM Total time: 11.6s				
✓ example.spec.ts				
✓ has title chromium				975ms
example.spec.ts:3				
✓ get started link chromium				1.1s
example.spec.ts:10				
✓ has title firefox				1.9s
example.spec.ts:3				
✓ get started link firefox				2.7s
example.spec.ts:10				
✓ has title webkit				2.4s
example.spec.ts:3				
✓ get started link webkit				2.4s
example.spec.ts:10				

<https://playwright.dev/docs/test-reporters>



Test configuration

Edit in file `playwright.config.ts`

Headless

Filter tests

Timeout

View port

Permission

Location

Global setup
and teardown

Recording

Retried

<https://playwright.dev/docs/test-configuration>



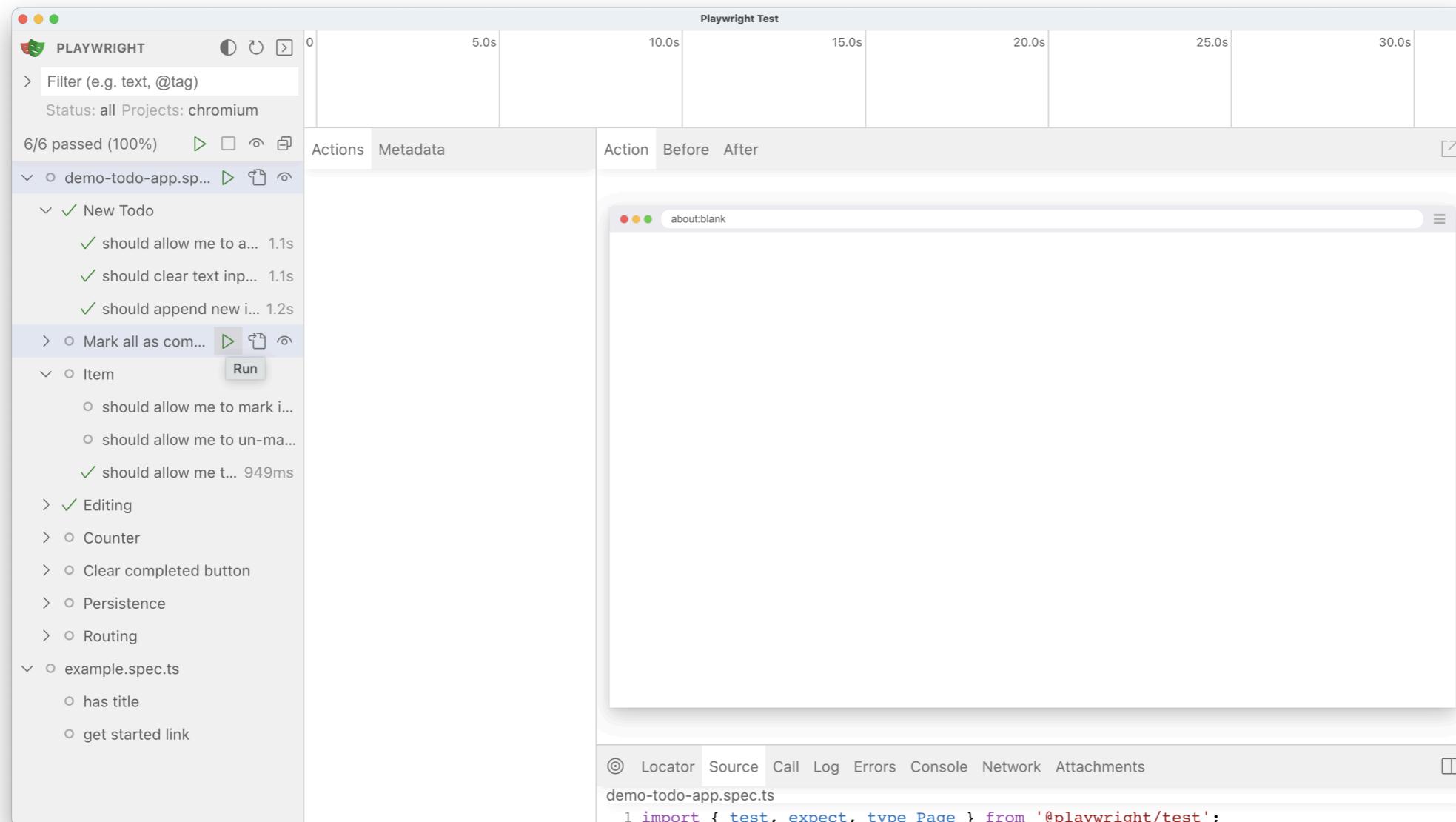
Example configuration

```
import { defineConfig } from '@playwright/test';
export default defineConfig({
  use: {
    headless: false,
    viewport: { width: 1280, height: 720 },
    ignoreHTTPSErrors: true,
    video: 'on',
  },
});
```



Start with Playwright in UI Mode

\$npx playwright test --ui

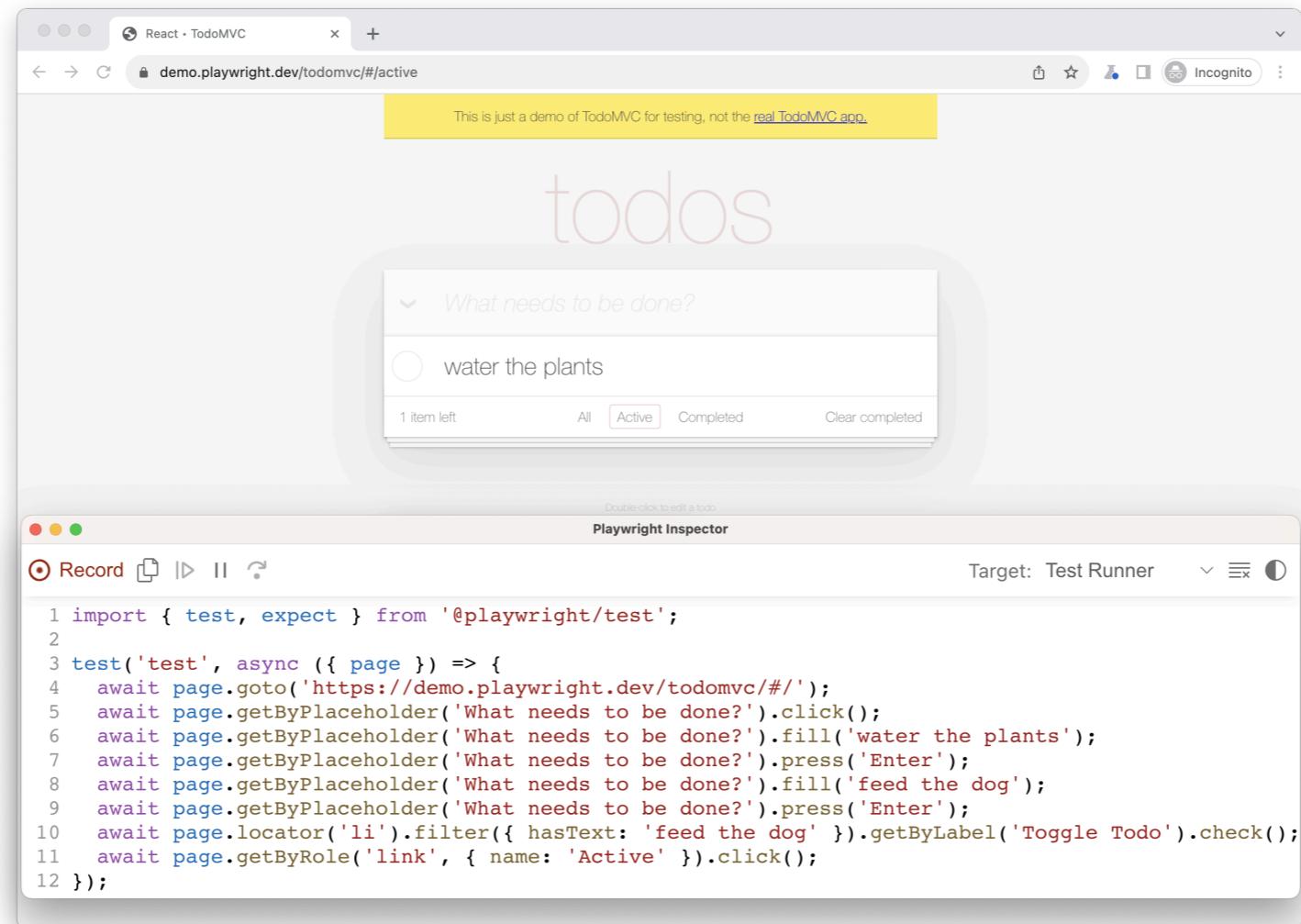


<https://playwright.dev/docs/test-ui-mode>



Record and Generate Test

\$npx playwright codegen demo.playwright.dev/todomvc



<https://playwright.dev/docs/codegen-intro>



Playwright Test for VSCode

Playwright Test for VS Code

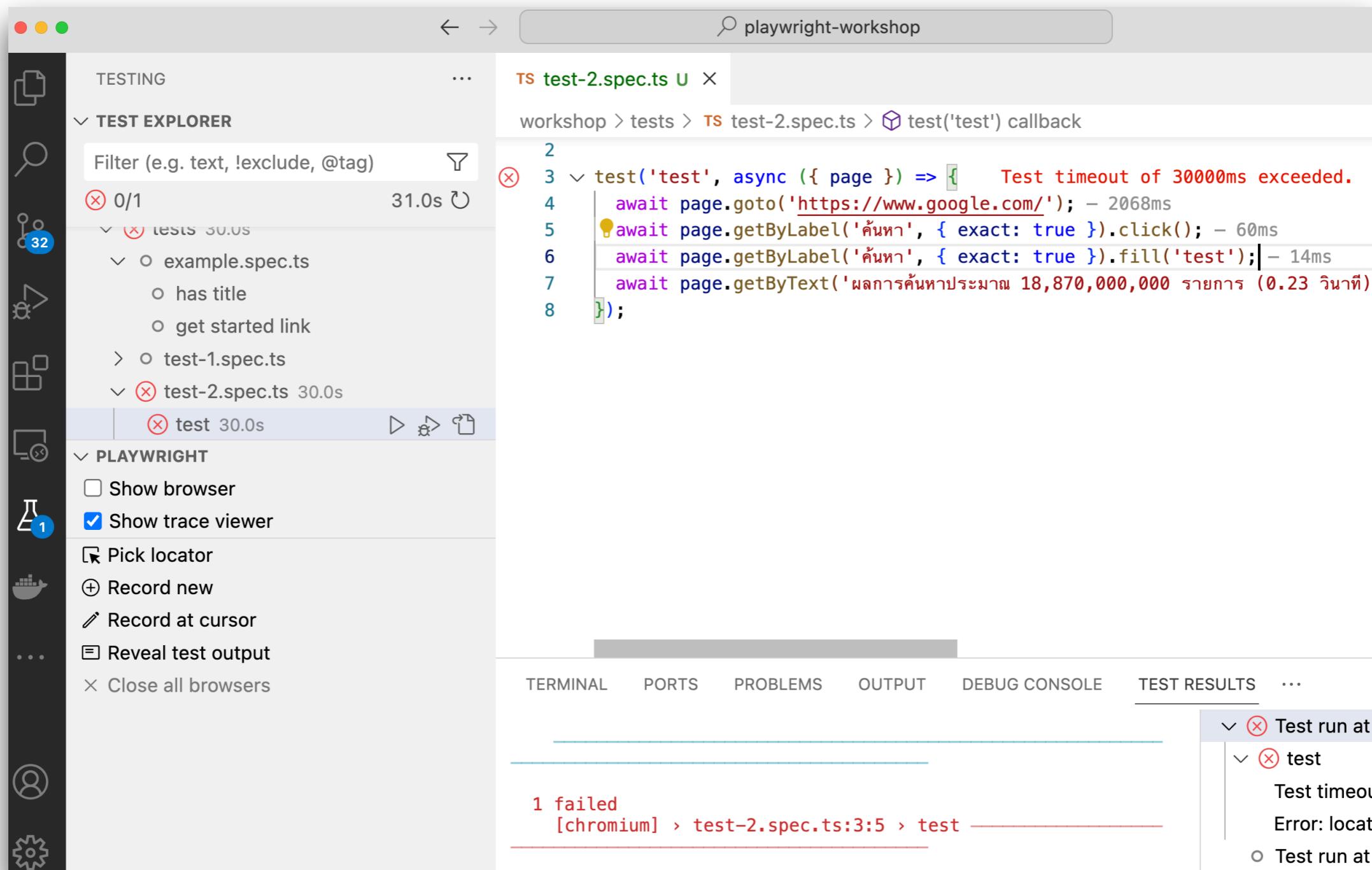
This extension integrates Playwright into your VS Code workflow. Here is what it can do:

- [Playwright Test for VS Code](#)
 - [Requirements](#)
 - [Install Playwright](#)
 - [Run tests with a single click](#)
 - [Run Multiple Tests](#)
 - [Show browsers](#)
 - [Pick locators](#)
 - [Debug step-by-step, explore locators](#)
 - [Tune locators](#)
 - [Record new tests](#)
 - [Record at cursor](#)

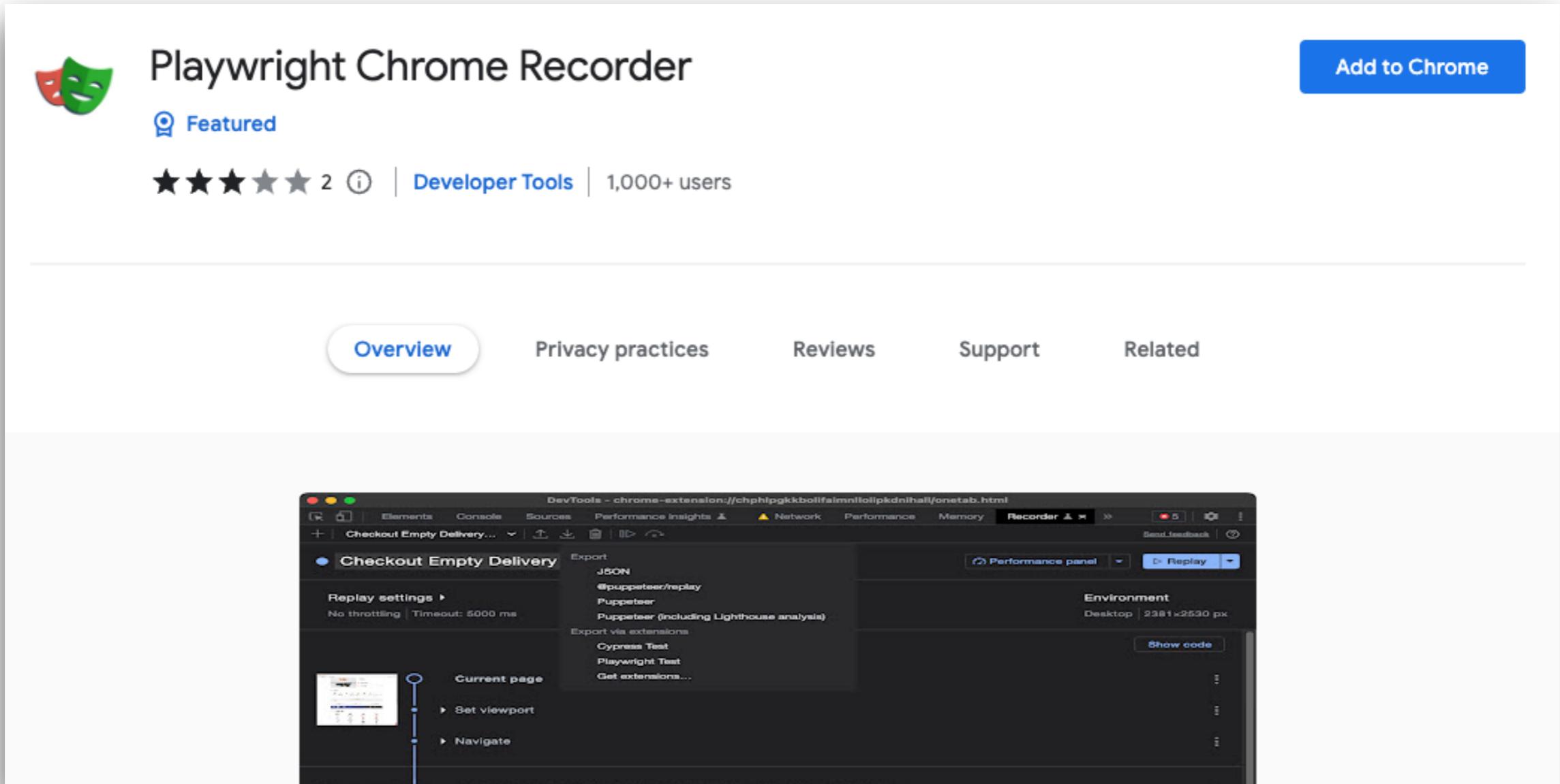
<https://marketplace.visualstudio.com/items?itemName=ms-playwright.playwright>



Playwright Test for VSCode



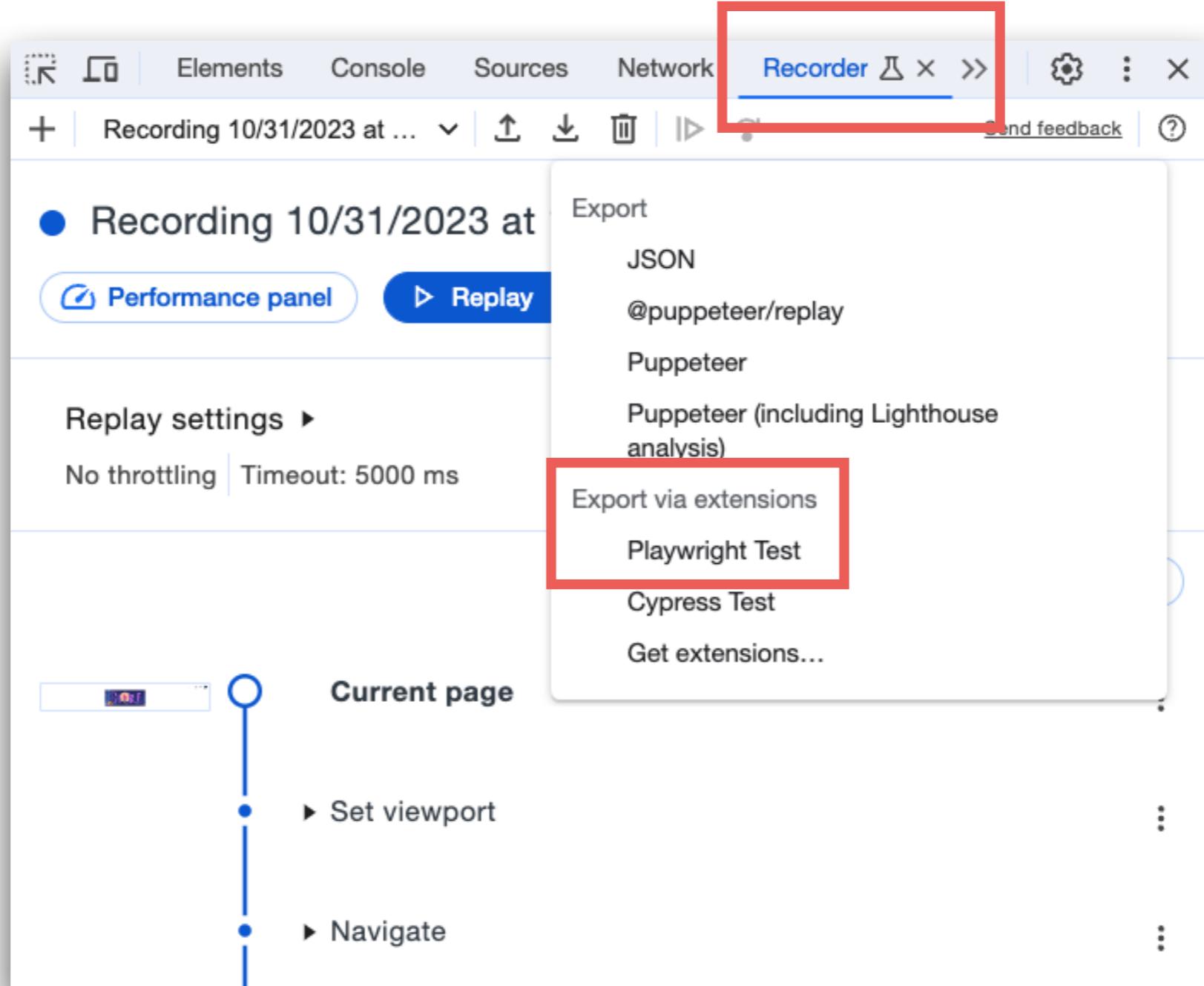
Playwright Chrome Recorder



<https://chrome.google.com/webstore/detail/playwright-chrome-recorde/bfnbgoehgplaehdceponclakmhlglpd>



Playwright Chrome Recorder



<https://chrome.google.com/webstore/detail/playwright-chrome-recorde/bfnbgoehgplaehdceponclakmhlglpd>



Example Code

```
await page.setViewportSize({
  width: 1579,
  height: 262
})
await page.goto("https://www.google.com/");
await page.goto("https://www.google.com/");
await page.locator("#APjFqb").click()
await page.locator("#APjFqb").type("test");
page.keyboard.down("{Enter}");
await page.locator("#cnt").click()
await page.locator("#result-stats").click()
});
```



Start to Write test case



Test Structure (AAA)

```
import { test, expect } from '@playwright/test';

test('get started link', async ({ page }) => {
    await page.goto('https://playwright.dev/');
    // Click the get started link.
    await page.getByRole('link', { name: 'Get started' }).click();
    // Expects page to have a heading with the name of Installation.
    await expect(page.getByRole('heading', { name: 'Installation' }))
        .toBeVisible();
});
```

<https://playwright.dev/docs/writing-tests>



Test class and functions

test.skip

test.only

test.fixme

test.fail

test.info

test.describe

test.setTimeout

test.slow

test.step

<https://playwright.dev/docs/api/class-test>



Test Life cycle

```
test.beforeAll(async ({ browser }) => {
  await browser.newContext();
});

test.afterAll(async ({ browser }) => {
  await browser.close();
});

test.beforeEach(async ({ page }) => {
  await page.goto('https://playwright.dev/');
});

test.afterEach(async ({ page }) => {
  await page.context().clearCookies();
});
```

<https://playwright.dev/docs/api/class-test>



Test Structure and Life cycle

```
test.beforeEach(async ({ page }) => {
  await page.goto('https://playwright.dev/');
});

test('has title', async ({ page }) => {
  // Assert
  await expect(page).toHaveTitle('/Playwright/');
});

test('get started link', async ({ page }) => {
  // Act
  await page.getByRole('link', { name: 'Get started' }).click();
  // Assert
  await expect(page.getByRole('heading', { name: 'Installation' }))
    .toBeVisible();
});
```

<https://playwright.dev/docs/best-practices>



Visual Testing

<https://playwright.dev/docs/test-snapshots>



Visual Testing

Compare with screenshots

```
import { test, expect } from '@playwright/test';

test('example test', async ({ page }) => {
  await page.goto('https://playwright.dev');
  await expect(page).toHaveScreenshot({ maxDiffPixels: 100 });
});
```

\$npx playwright test --update-snapshots



Playwright Best Practices



Playwright Best Practices

Test user-visible behavior

Make tests as isolated as possible

Avoid testing 3-party dependencies

Testing with database (control data)

<https://playwright.dev/docs/best-practices>



Finding and Using Elements

Isolated from UI !!



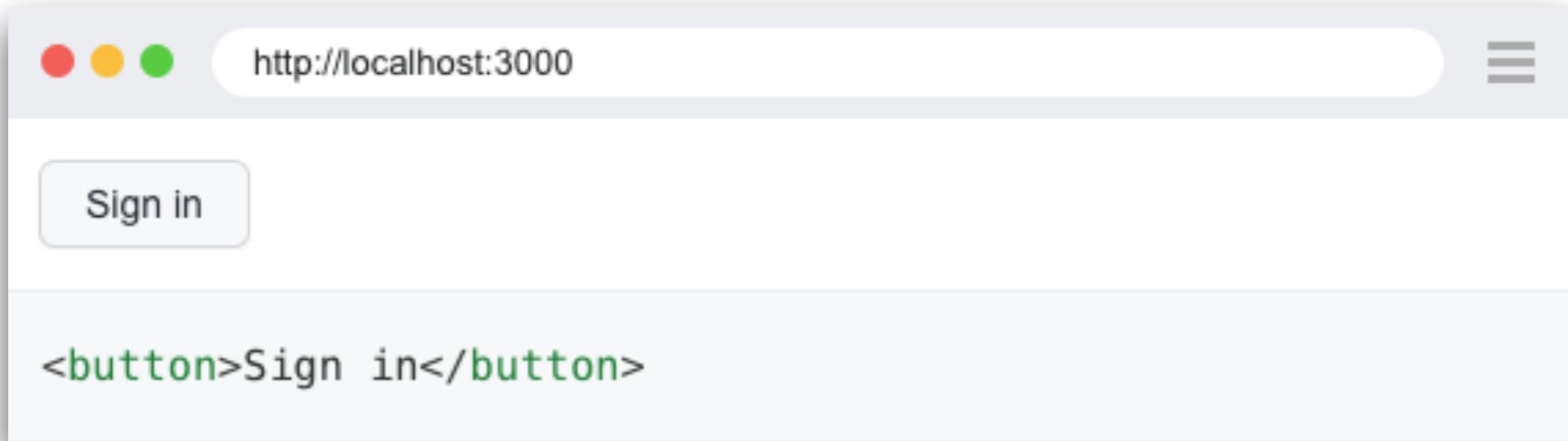
Building locators

Locators	Description
page.getByRole()	By data type of tag or input
page.getText()	From text content
page.getLabel()	From label's text
page.getTestId()	Use data-testid attribute
page.locator('xpath=//button')	XPath
page.locator('css=button')	CSS

<https://playwright.dev/docs/locators>



getByRole()



```
await page.getByRole('button', { name: 'Sign in' })
    .click();
```

<https://playwright.dev/docs/api/class-page#page-get-by-role>



Role Definitions

From ARIA guidelines

§ 5.4 Definition of Roles

Below is an alphabetical list of [WAI-ARIA roles](#) to be used by authors.

Abstract roles are used for the ontology. Authors **MUST NOT** use abstract roles in content.

[alert](#)

A type of [live region](#) with important, and usually time-sensitive, information. See related [alertdialog](#) and [status](#).

[alertdialog](#)

A type of dialog that contains an alert message, where initial focus goes to an [element](#) within the dialog. See related [alert](#) and [dialog](#).

[application](#)

A [structure](#) containing one or more focusable elements requiring user input, such as keyboard or gesture events, that do not follow a standard interaction pattern supported by a [widget](#) role.

[article](#)

A section of a page that consists of a composition that forms an independent part of a document, page, or site.

[banner](#)

A [landmark](#) that contains mostly site-oriented content, rather than page-specific content.

https://www.w3.org/TR/wai-aria-1.2/#role_definitions



getByRole()



```
await expect(page.getByRole('heading', { name: 'Sign up' })).toBeVisible();

await page.getByRole('checkbox', { name: 'Subscribe' }).check();

await page.getByRole('button', { name: /submit/i }).click();
```



Locate by test id



```
await page.getByTestId('directions').click();
```



Change test id

Edit in file `playwright.config.ts`

```
import { defineConfig } from '@playwright/test';

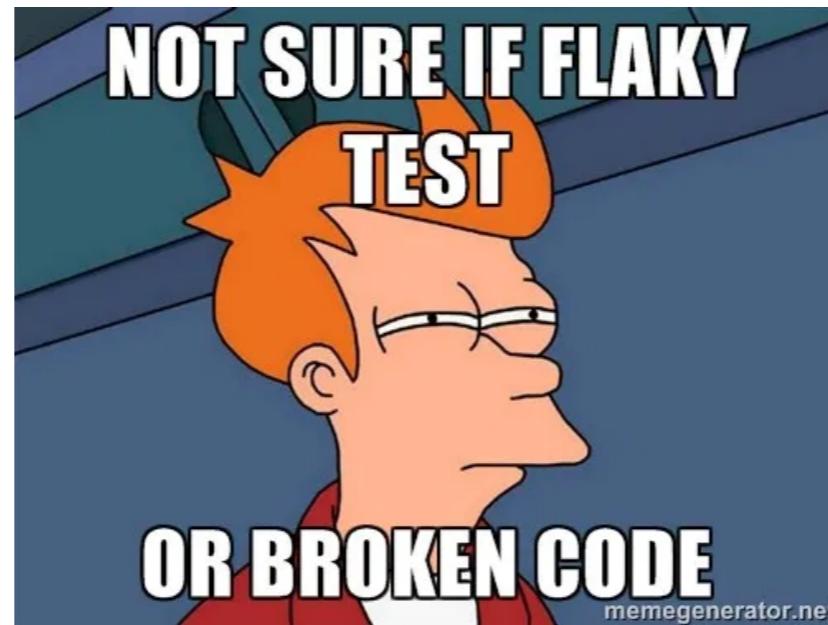
export default defineConfig({
  use: {
    testIdAttribute: 'data-pw'
  }
});
```



Good Locators

Unique
Descriptive
Resilient

Shorter in length (maintainability)



Playwright suggestion

Use locator over selector

```
await page.locator('text=Login').click()
```

```
await page.click('text=Login')
```



Playwright suggestion

Facing locator from user perspective

Use data-tested



Locate with XPath !!



XPath

CML Path Language
Query language used to identify tag in XML
Appium builds XML representation of app

XPath is powerful but very dangerous



Benefits of XPath

Find any element that exists

Find elements by using complex criteria



Cons of XPath

XPath selectors can be brittle

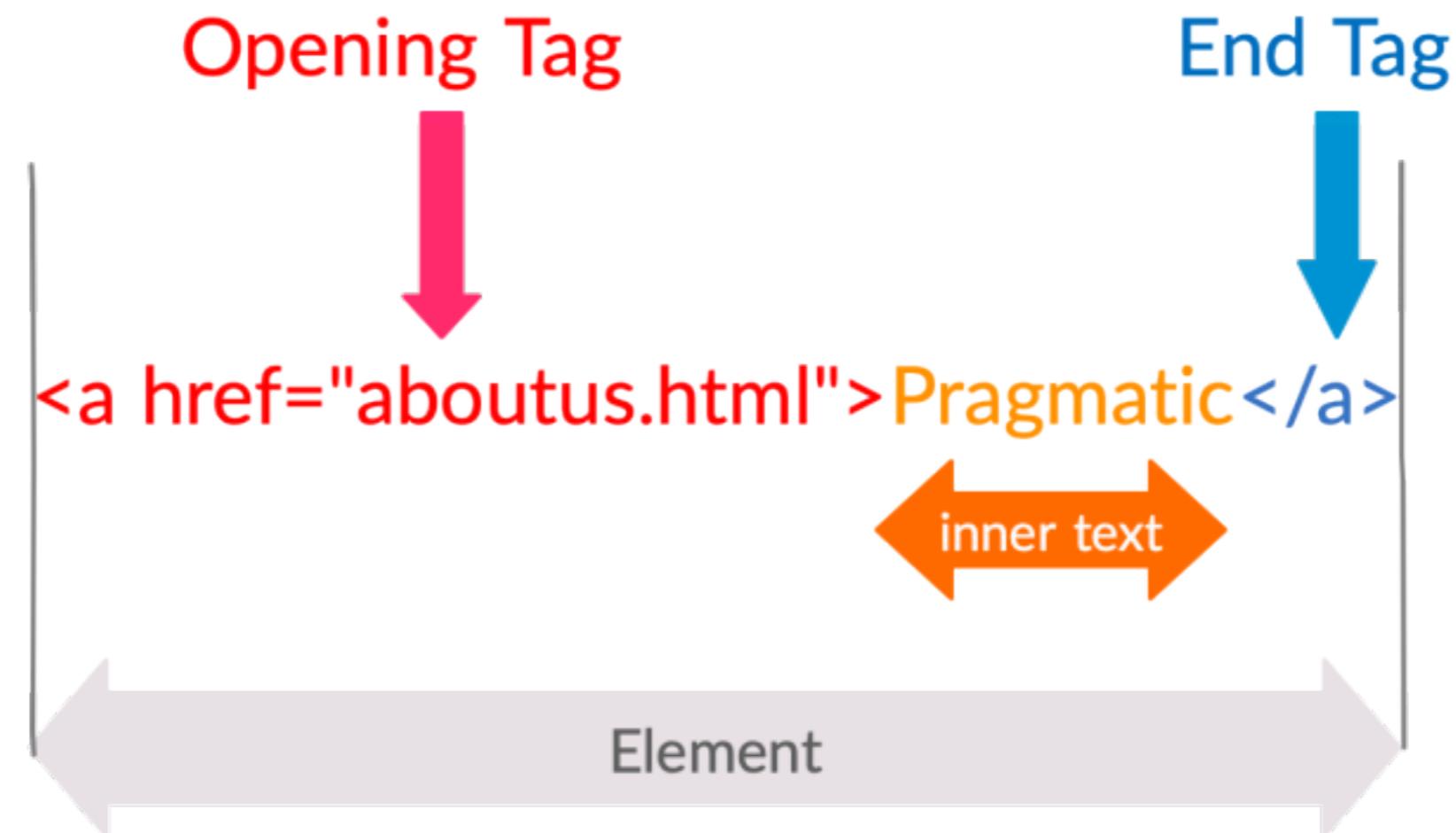
Slow

Broken test !!

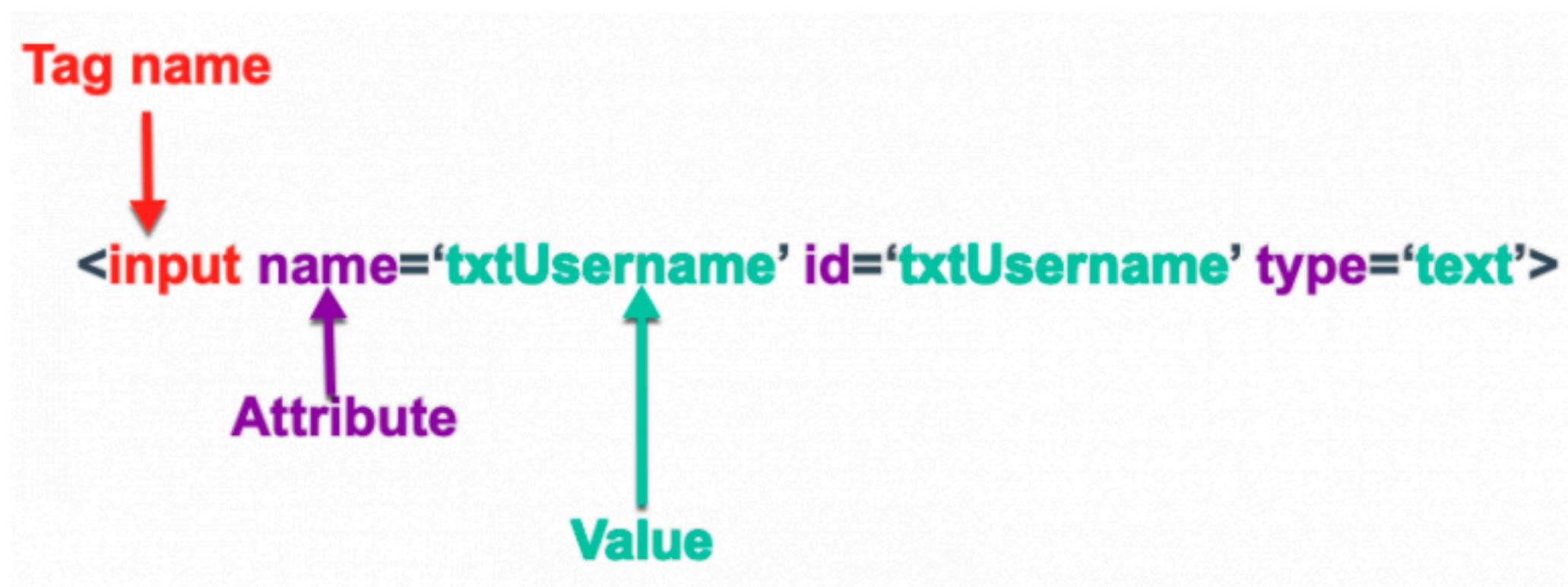
Avoid brittle selectors by rely on unique tag attribute



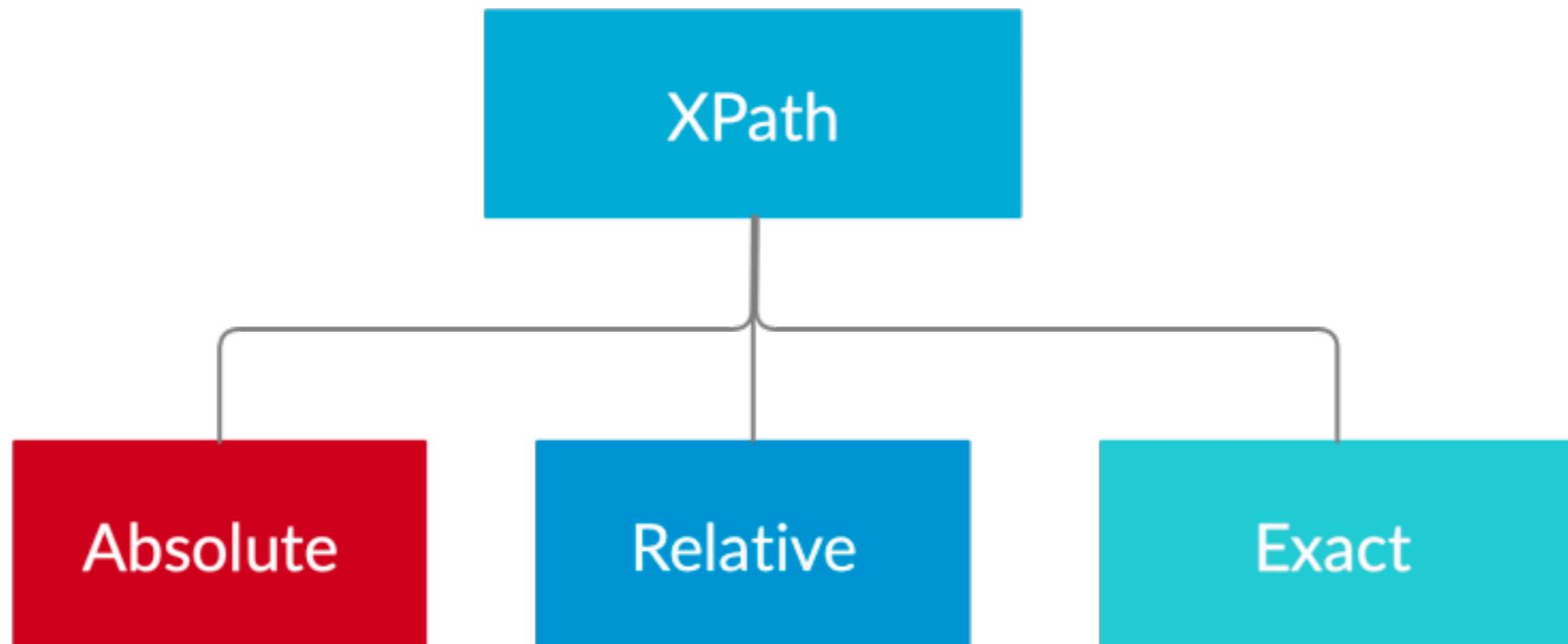
XPath



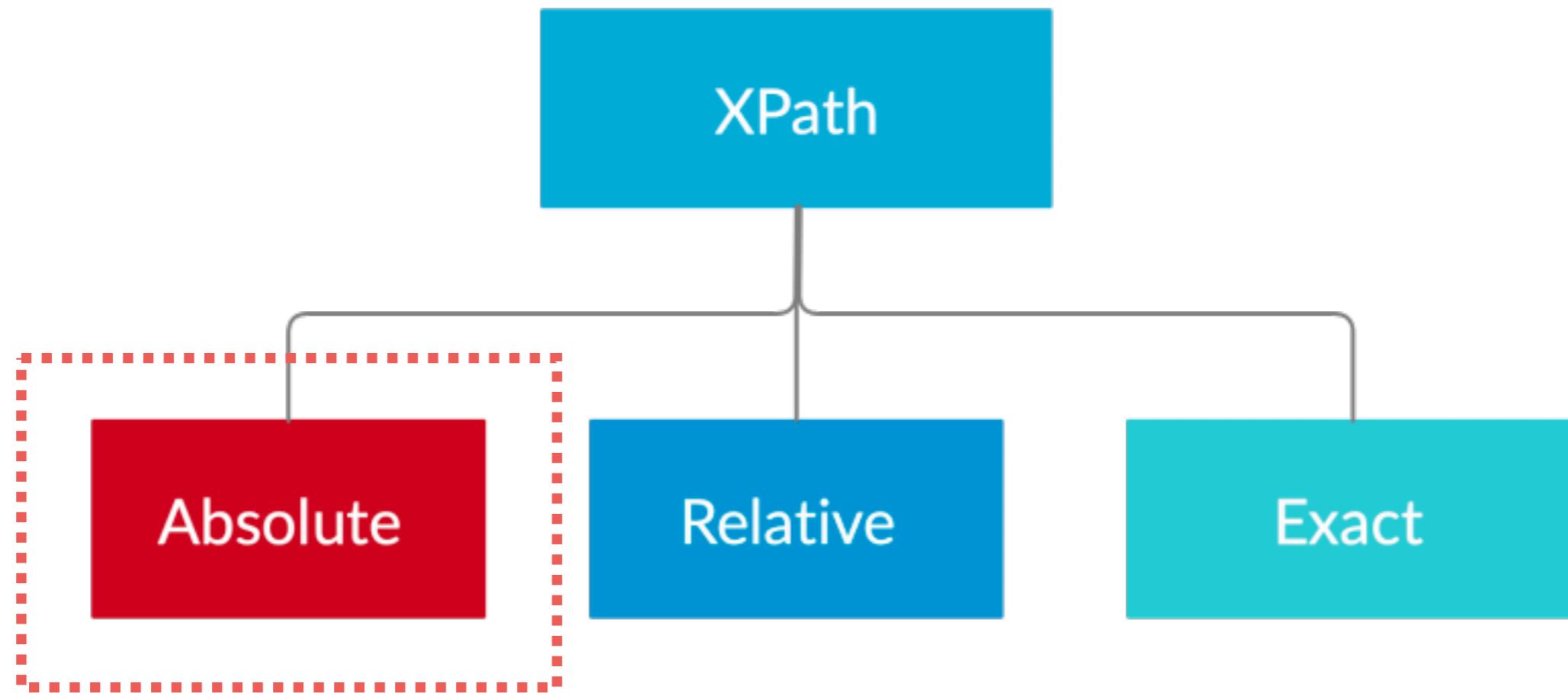
XPath



Types XPath



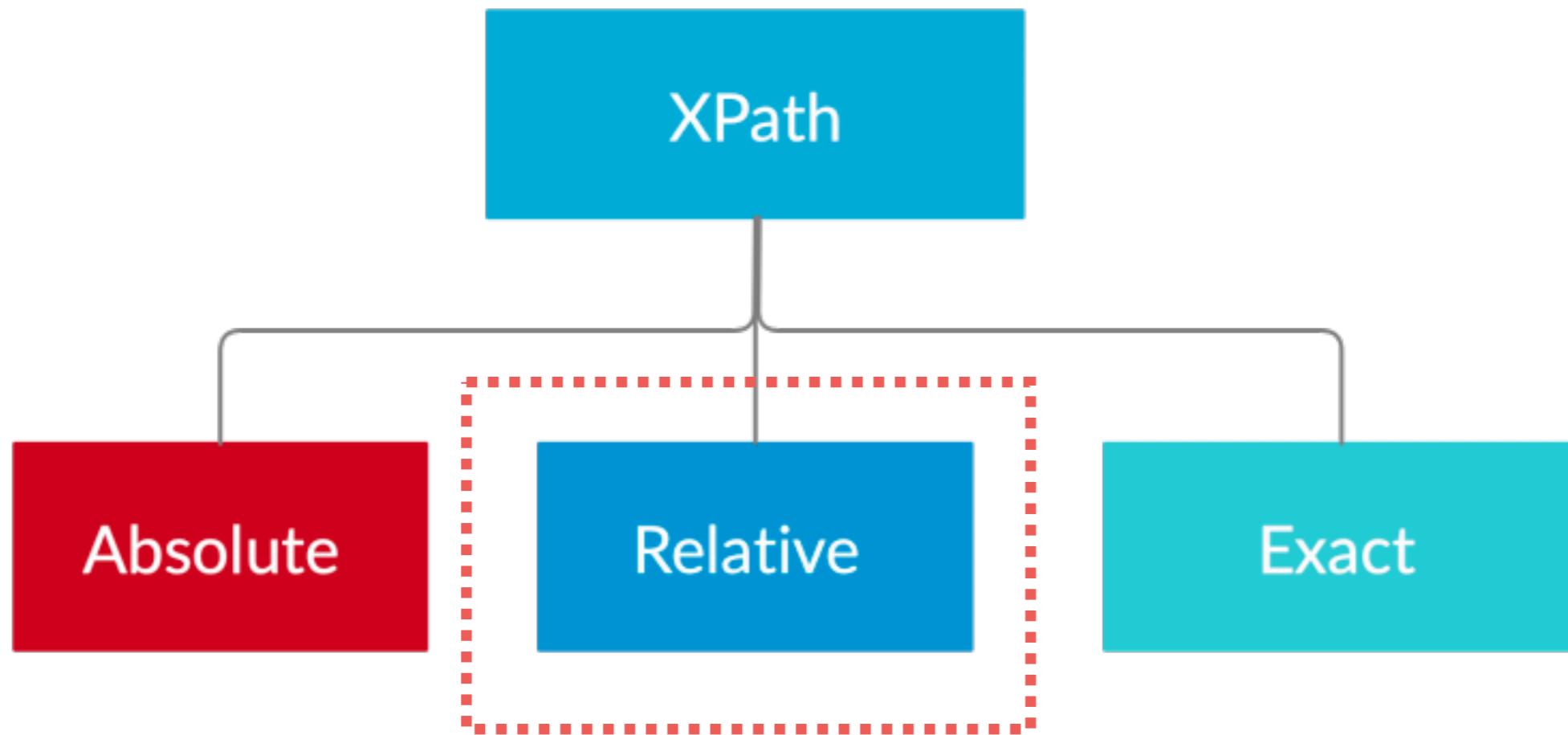
Don't use !!



`/html/body/div[1]/div/div[2]/form/div[2]/input`



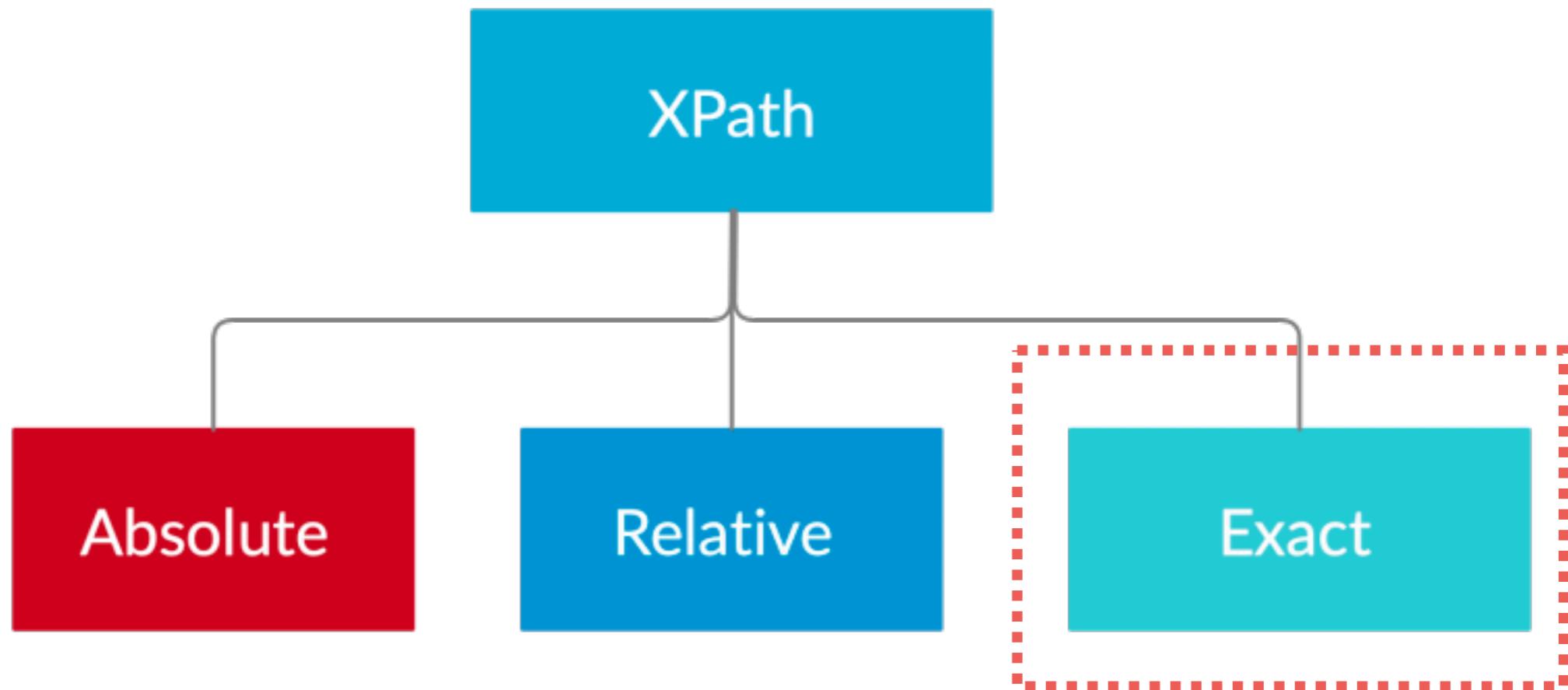
Relative



```
//div[@id='divUsername']/input  
//form/div[@id='divUsername']/input  
//form/*/input
```



Exact



```
//div[@class='datevalue currmonth']//span[./text()='2']
```



**Absolute faster than Relative
But shortest is better**



Waiting for Elements



Auto waiting with timeout

<https://playwright.dev/docs/actionability>



Config Timeout

Edit in file `playwright.config.ts`

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  timeout: 5 * 60 * 1000,
});
```

<https://playwright.dev/docs/test-timeouts>



Tag tests

`@tag_name` in your tests
Run by tag name

`@Fast`

`@Sprint1`

`@Feature1`

`@Medium`

`@Sprint2`

`@Feature2`

`@Slow`

`@Sprint3`

`@Feature3`

<https://playwright.dev/docs/test-annotations#tag-tests>



Tag tests

```
test('Upload 3 files @upload', async ({ page }) => {  
});
```

\$npm playwright test --grep @upload

<https://playwright.dev/docs/test-timeouts>



Workshop with Upload file

<http://nervgh.github.io/pages/angular-file-upload/examples/simple/>



Workshop with page.waitForEvent

<https://efiling.rd.go.th/rd-cms/>

Popup

Request

Response

Filechooser

<https://playwright.dev/docs/api/class-page#page-wait-for-event>



```
import { test, expect } from '@playwright/test';

test('Try with RD CMS', async ({ page }) => {

    await page.goto('https://efiling.rd.go.th/rd-cms/');
    await page.getByRole('heading', { name: 'แจ้งข่าวสาร' }).click();

    const page1Promise = page.waitForEvent('popup');
    await page.getByRole('link', { name: 'รายละเอียด' }).click();
    const page1 = await page1Promise;
    await page1.getByRole('heading',
        { name: 'ช่วงเวลาการให้บริการของแต่ละช่องทาง' }).click();
    await page.getByLabel('Close').click();

    const page2Promise = page.waitForEvent('popup');
    await page.getByRole('button',
        { name: 'ยื่นแบบออนไลน์' }).getByRole('link').click();
    const page2 = await page2Promise;
    await page2.getByRole('button', { name: 'เข้าสู่ระบบ' }).click();

});
```



```

test('Try with RD CMS', async ({ page }) => {

  await test.step('Step 1', async () => {
    await page.goto('https://efiling.rd.go.th/rd-cms/');
    await page.getByRole('heading',
      { name: 'แจ้งข่าวสาร' }).click();
  });

  await test.step('Step 2', async () => {
    const page1Promise = page.waitForEvent('popup');
    await page.getByRole('link', { name: 'รายละเอียด' }).click();
    const page1 = await page1Promise;
    await page1.getByRole('heading',
      { name: 'ช่วงเวลาการให้บริการของแต่ละช่องทาง' }).click();
  });

  await test.step('Step 3', async () => {
    await page.getByLabel('Close').click();
    const page2Promise = page.waitForEvent('popup');
    await page.getByRole('button',
      { name: 'ยืนยันออนไลน์' }).getByRole('link').click();
    const page2 = await page2Promise;
    await page2.getByRole('button',
      { name: 'เข้าสู่ระบบ' }).click();
  });
});

```

<https://playwright.dev/docs/api/class-test#test-step>



Good Test ?



Good Tests ?

Fast
Isolate
Repeatable
Self-verify
Timely

<https://playwright.dev/docs/browser-contexts>



Test Isolation in Playwright

```
import { test } from '@playwright/test';

test('example test', async ({ page, context }) => {
    // "context" is an isolated BrowserContext, created for this specific test.
    // "page" belongs to this context.
});

test('another test', async ({ page, context }) => {
    // "context" and "page" in this second test are completely
    // isolated from the first test.
});
```

<https://playwright.dev/docs/browser-contexts>



Create multiple context in test

```
import { test } from '@playwright/test';

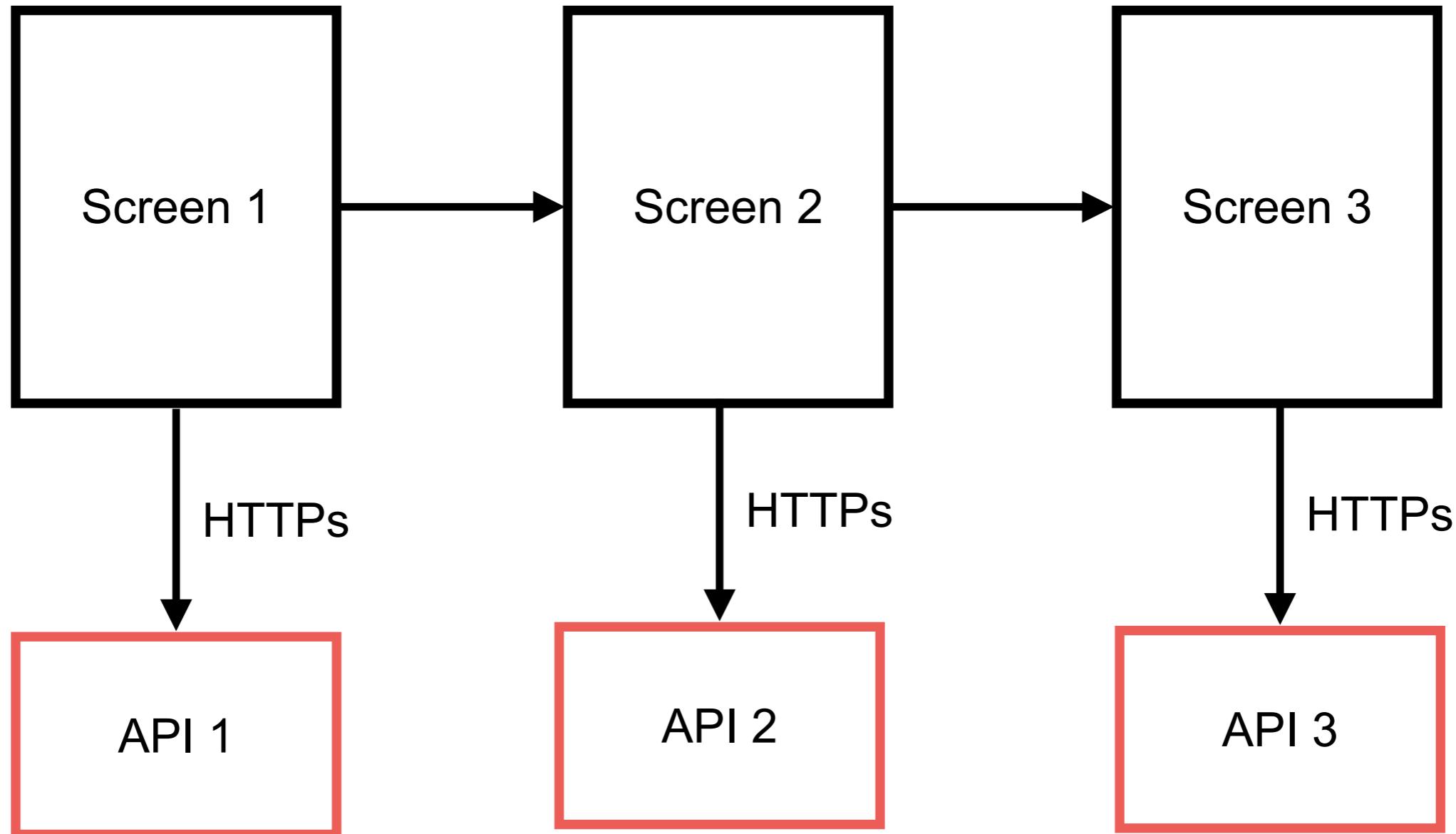
test('admin and user', async ({ browser }) => {
  // Create two isolated browser contexts
  const adminContext = await browser.newContext();
  const userContext = await browser.newContext();

  // Create pages and interact with contexts independently
  const adminPage = await adminContext.newPage();
  const userPage = await userContext.newPage();
});
```

<https://playwright.dev/docs/browser-contexts>



Integration Testing



Mock Server (HTTPs)

Stubby4Node

<https://github.com/mrak/stubby4node>

MountBank

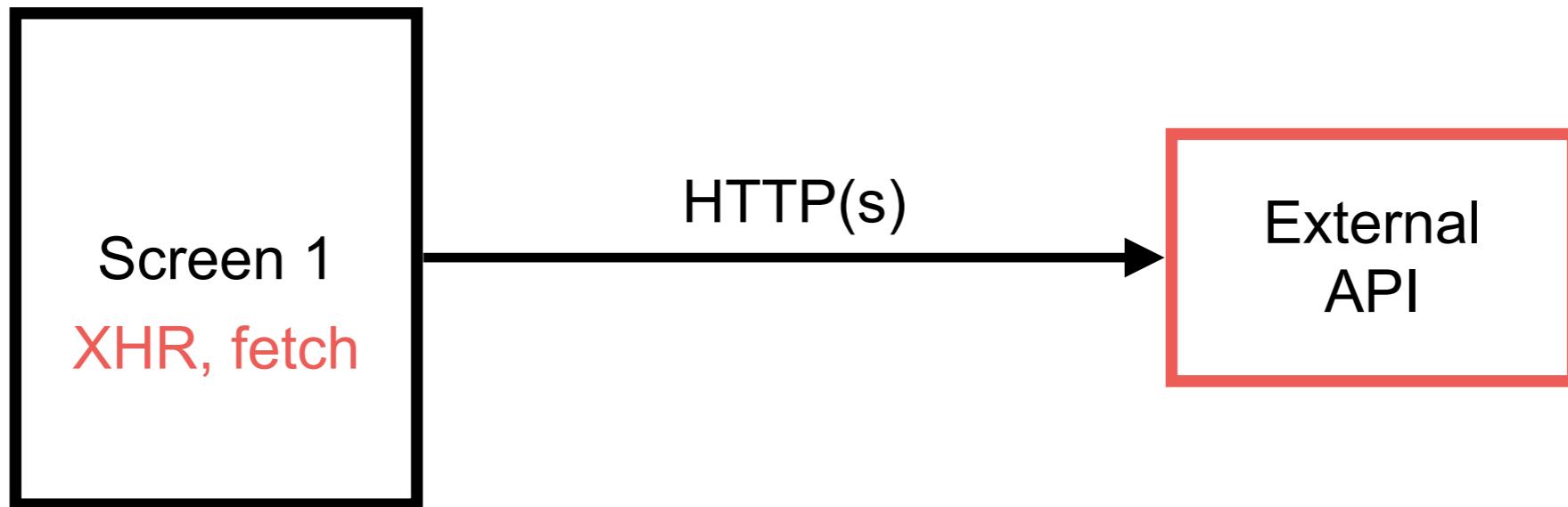
<https://www.mbttest.org/>

WireMock

<https://wiremock.org/>



Mock API with Playwright



<https://playwright.dev/docs/network>



Inspect Fetch/XHR

The screenshot shows a browser window with the URL `demo.playwright.dev/api-mocking/`. The main content area displays a heading "Render a List of Fruits" and a list of fruits: Strawberry, Banana, Tomato, Pear, Blackberry, Kiwi, Pineapple, Passionfruit, Orange, Raspberry, Watermelon, Lemon, Mango, Blueberry, and Apple. Below the list are two links: "Check out the tests for this repo" and "Learn more on API mocking in Playwright".

The browser's developer tools Network tab is open, with the "Network" tab selected. A red box highlights the "Fetch/XHR" filter button. Another red box highlights the "Response" tab in the table header, which is currently active. The table lists a single request named "fruits". The response body is shown as a JSON array:

```
[{"name": "Strawberry", "id": 3}, {"name": "Banana", "id": 1}, {"name": "Tomato", "id": 5}, {"name": "Pear", "id": 4}, {"name": "Blackberry", "id": 64}, {"name": "Kiwi", "id": 66}...]
```



Write test with mock api

```
import { test, expect } from '@playwright/test';

test("Mock API", async ({ page }) => {

    // Mock the api call before navigating
    await page.route('*/**/api/v1/fruits', async route => {
        const json = [{ name: 'Mock Data', id: 21 }];
        await route.fulfill({ json });
    });

    await page.goto('https://demo.playwright.dev/api-mocking');
    await expect(page.getByText('Mock Data')).toBeVisible();
});
```

<https://playwright.dev/docs/mock>



Test in UI Mode

\$npx playwright test --ui

The screenshot shows the Playwright UI test runner interface. On the left, a sidebar displays a tree view of test actions and metadata. A specific action, 'expect.toBeVisible getByText("Fruit")', is highlighted with a blue selection bar. The main area features a browser window displaying a demo application titled 'Render a List of Fruits'. Below the browser is a network tab showing a list of requests and responses. One request, a GET to '/fruits', is highlighted with a red box and labeled 'fulfilled'.

Status	Method	Request	Content Type	Duration	Size	Route
690ms	301	GET /api-mocking	text/html	194ms	-	
717ms	200	GET /	text/html	27ms	550	
764ms	200	GET /index-6d10c910.css	text/css	42ms	2.2K	
765ms	200	GET /index_e9b211ff.js	application/javascript	68ms	46.0K	
818ms	200	GET /fruits	application/json	6ms	95	fulfilled
819ms	-1	GET /playwright-logo.svg	x-unknown	-1ms	-	



Web Server in Playwright

<https://playwright.dev/docs/test-webserver>



Web Server

Start dev server before run your tests
Config in file **playwright.config.ts**

```
export default defineConfig({
  // Run your local dev server before starting the tests
  webServer: {
    command: 'npm run start',
    url: 'http://127.0.0.1:3000',
    reuseExistingServer: !process.env.CI,
    stdout: 'ignore',
    stderr: 'pipe',
  },
});
```

<https://playwright.dev/docs/test-webserver>



Demo

<https://github.com/up1/workshop-playwright/tree/main/webserver>



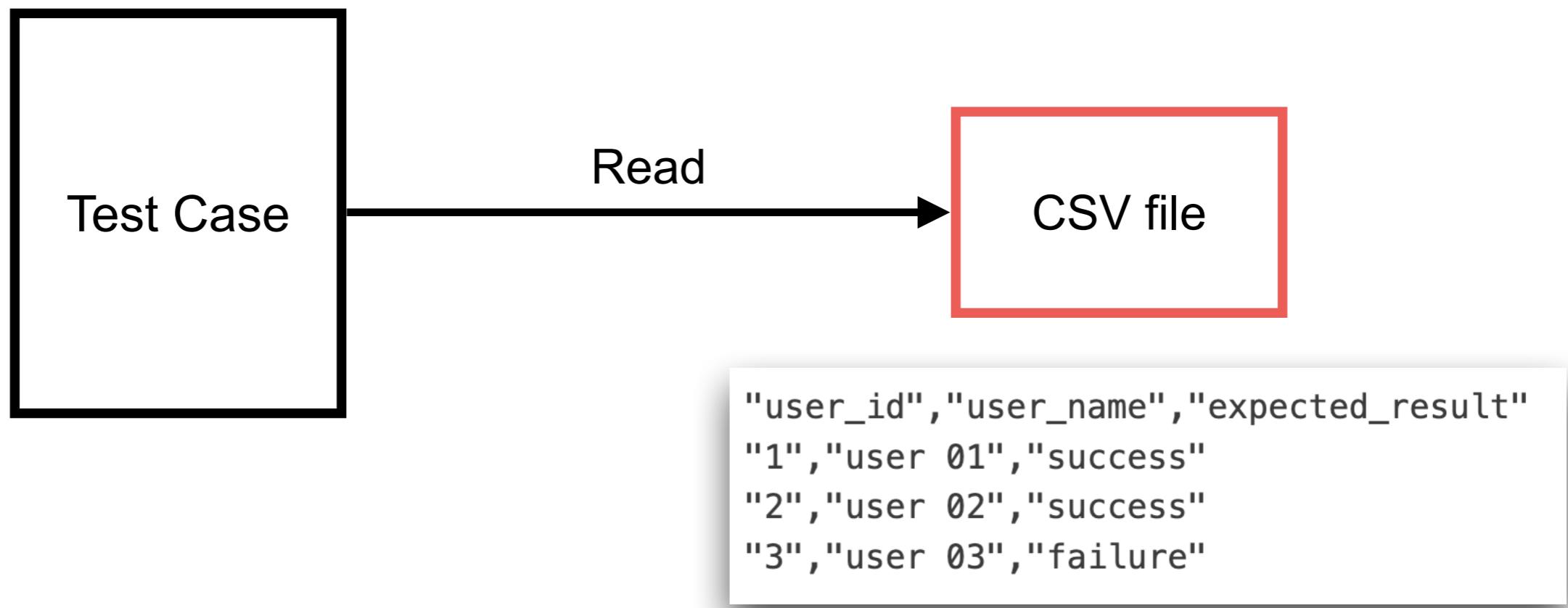
Working with CSV file

<https://playwright.dev/docs/test-parameterize#create-tests-via-a-csv-file>



Data test in CSV file

Use **csv-parse** library to read css file



<https://www.npmjs.com/package/csv-parse>



Test case with CSV

```
import fs from 'fs';
import path from 'path';
import { test } from '@playwright/test';
import { parse } from 'csv-parse/sync';

const records = parse(fs.readFileSync(path.join(__dirname, 'user.csv')), {
  columns: true,
  skip_empty_lines: true
});

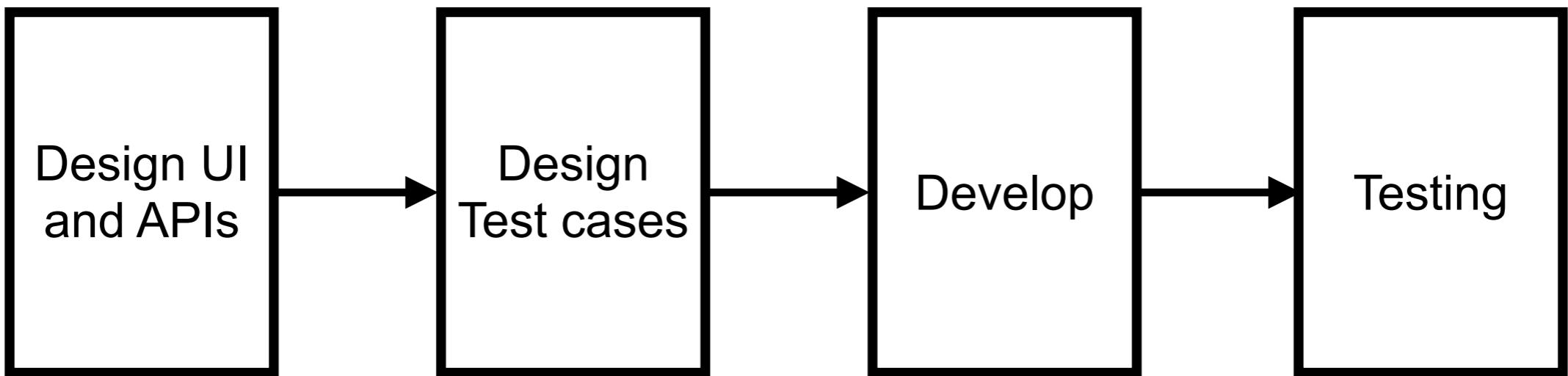
for (const record of records) {
  test(`Test case: ${record.test_case}`, async ({ }) => {
    console.log(record.user_id, record.user_name, record.expected_result);
  });
}
```



Test-First vs Test-Last ?



Improve Process



data-testid ?
Design element locator ?



BDD Style

<https://github.com/vitalets/playwright-bdd>





BDD

Scenario

Given
When
And
When
Then



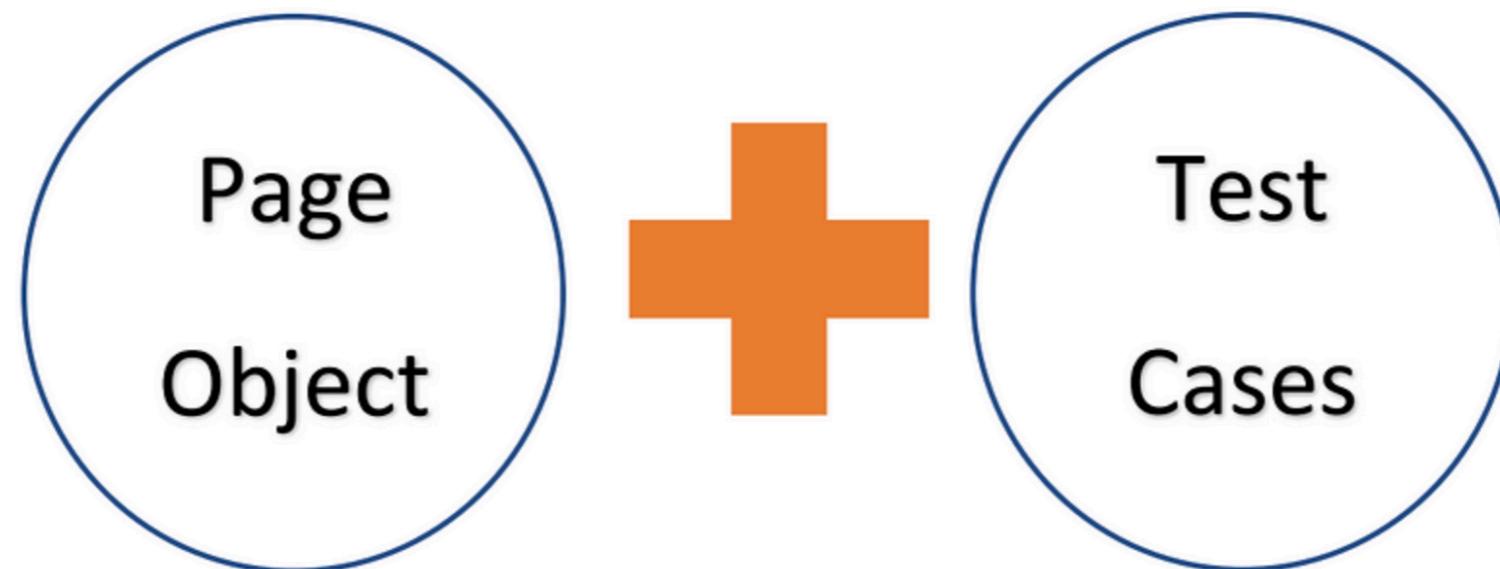
Page Object Pattern

<https://playwright.dev/docs/pom>

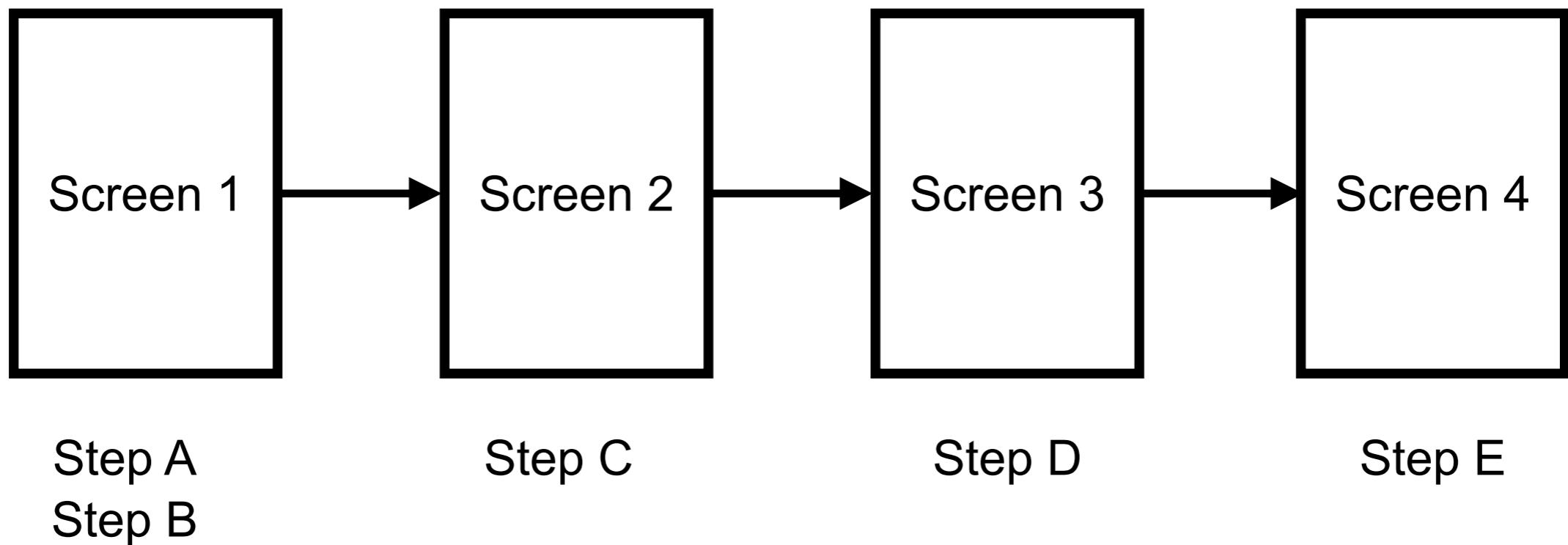


Page Object Pattern

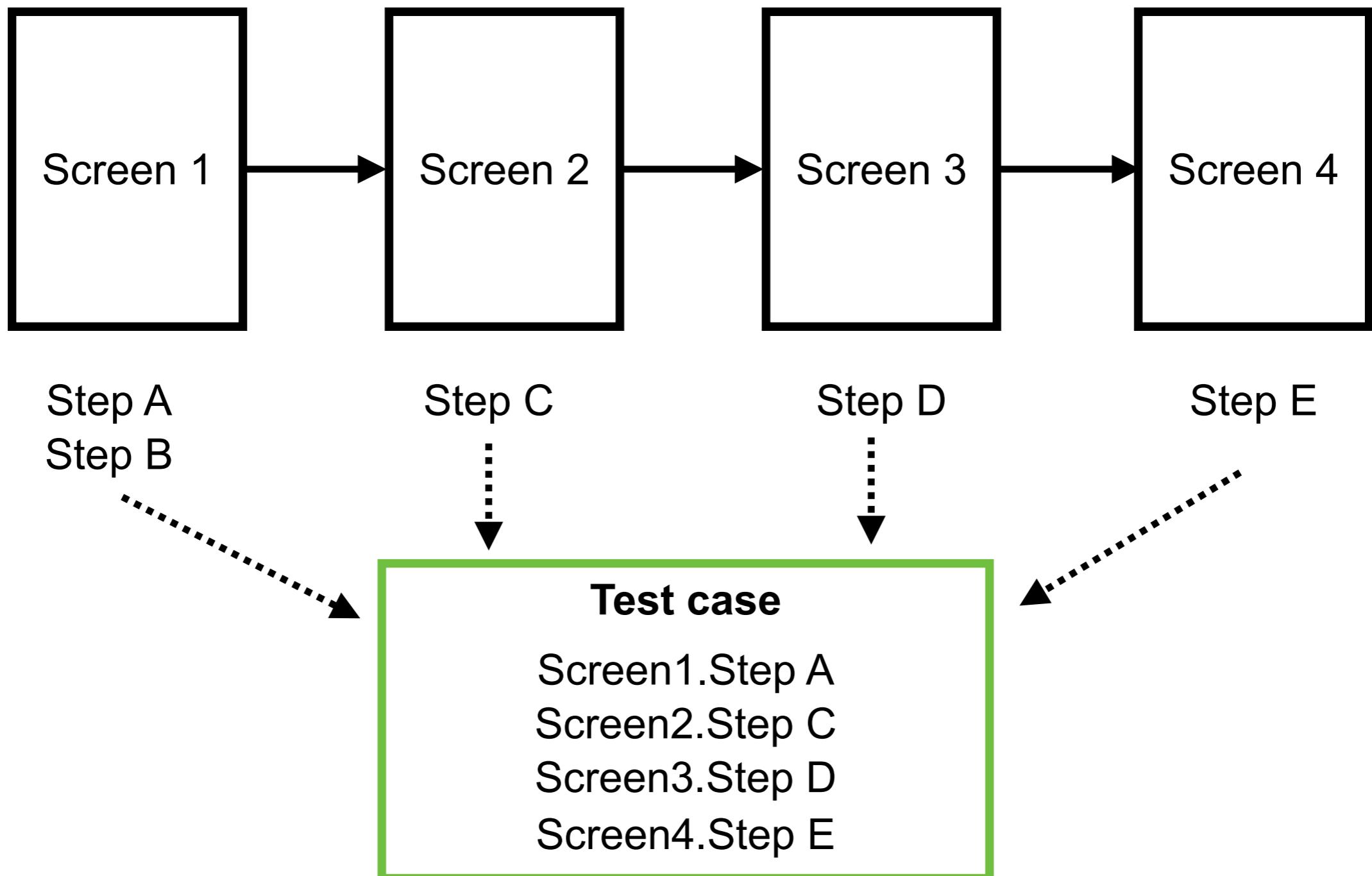
Maintainable test cases
Reduce code duplication
Working as a team



Use Page Object Pattern



Use Page Object Pattern

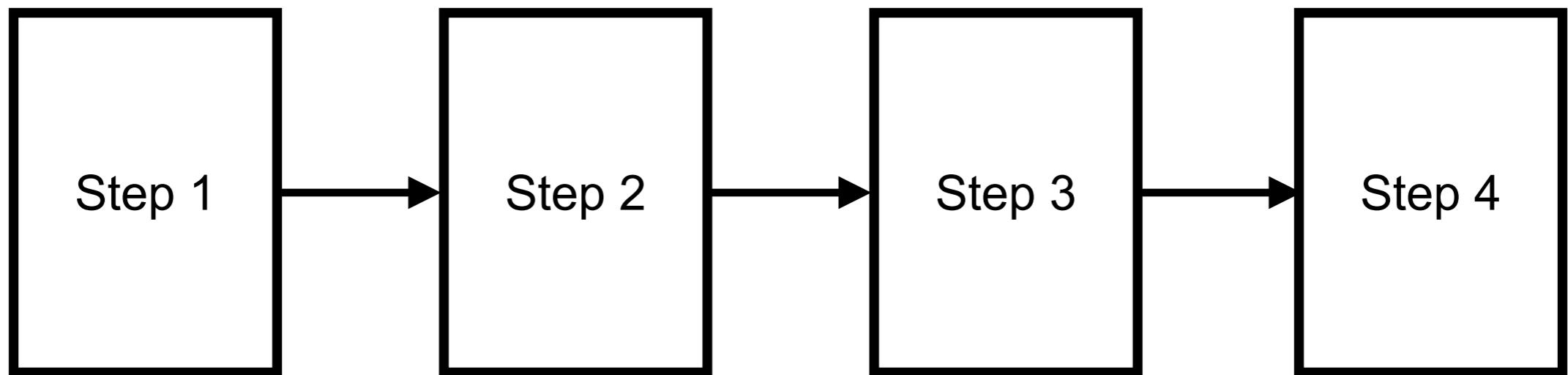


Workshop with Page Object

https://github.com/up1/workshop-playwright/tree/main/page_object



Workshop



<https://playwright.dev>

Get started

Node.js

Java



Write test case

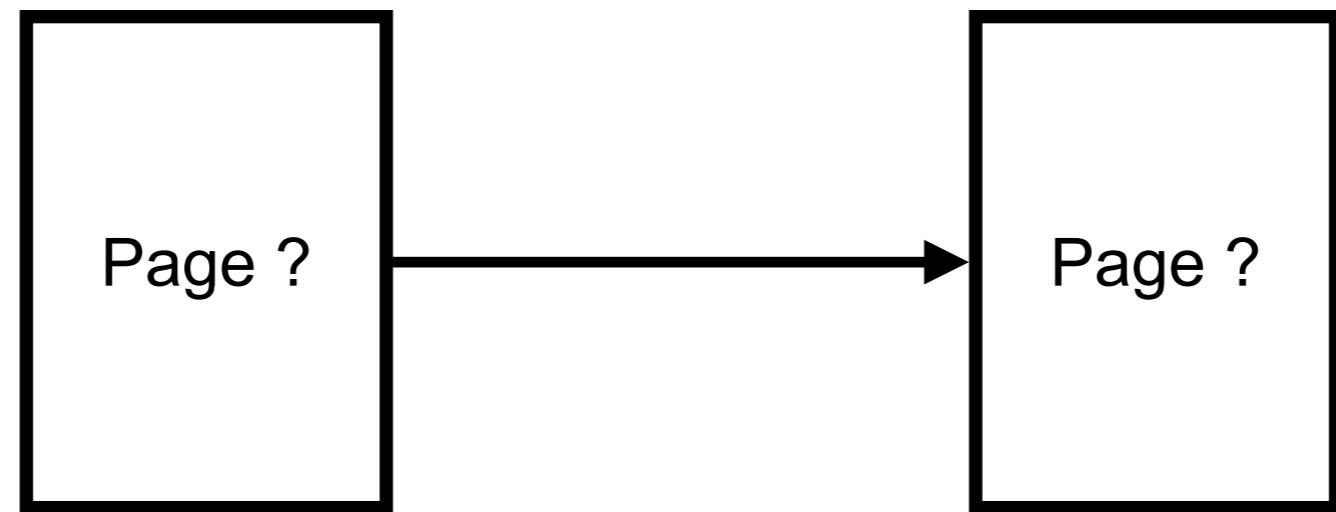
Make it work

```
test('Check title in home page', async ({ page }) => {
  await page.goto('https://playwright.dev/');
  await expect(page).toHaveTitle('/Playwright/');
});

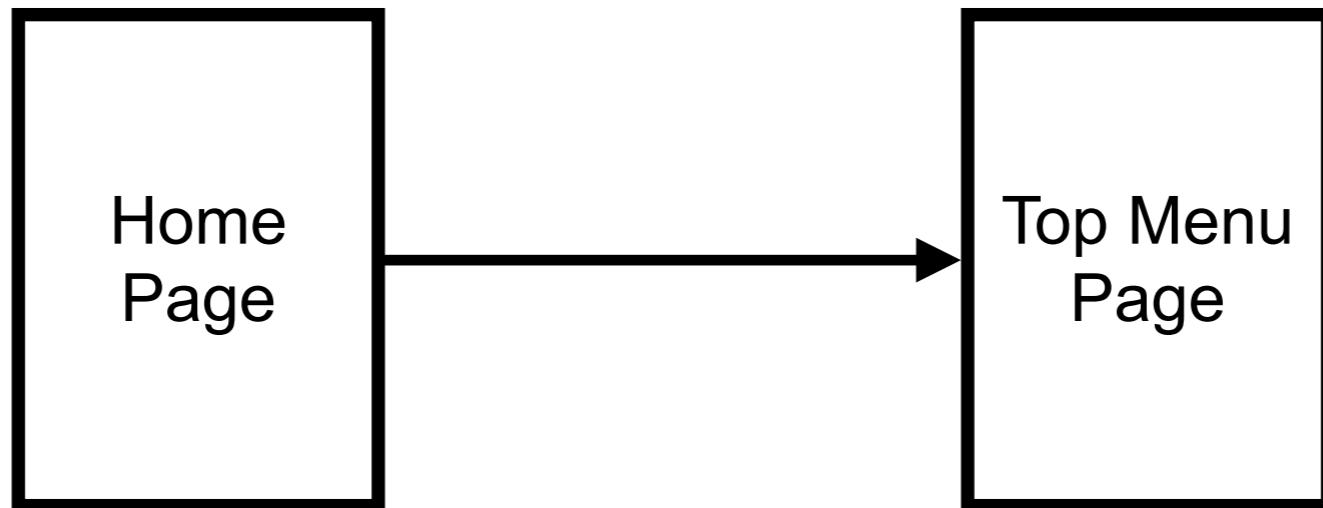
test('Check started link', async ({ page }) => {
  await page.goto('https://playwright.dev/');
  await page.getByRole('link', { name: 'Get started' }).click();
  await expect(page).toHaveURL(/.*intro/);
});
```



Design Page Object ?



Page Object ?



Check detail of page
Click link of Getting start

Choose Node.js
Click menu Java
Check content of Java



Home Page

```
import { type Locator, type Page, expect } from '@playwright/test';

export class HomePage {
    readonly page: Page;
    readonly getStartedButton: Locator;
    readonly pageTitle: RegExp;

    constructor(page: Page) {
        this.page = page;
        this.getStartedButton = page.getByRole('link', { name: 'Get started' });
        this.pageTitle = /Playwright/;
    }

    async clickGetStarted() {
        await this.getStartedButton.click();
    }

    async assertPageTitle() {
        await expect(this.page).toHaveTitle(this.pageTitle);
    }
}

export default HomePage;
```



First case

```
import { test } from '@playwright/test';
import { HomePage } from '../pages/home-page';

const URL = 'https://playwright.dev/';
let HomePage: HomePage;

test.describe('Playwright website', () => {
  test('Check title in home page', async ({ page }) => {
    await page.goto(URL);
    HomePage = new HomePage(page);
    await HomePage.assertPageTitle();
  });
});
```

<https://playwright.dev/docs/api/class-test#test-describe-1>



Second case

```
import { HomePage } from '../pages/home-page';
import { TopMenuPage } from '../pages/topmenu-page';

const URL = 'https://playwright.dev/';
const pageUrl = /.*intro/;
let HomePage: HomePage;
let TopMenuPage: TopMenuPage;

test.describe('Playwright website', () => {

  test('Check started link', async ({ page }) => {
    // Act
    await page.goto(URL);
    HomePage = new HomePage(page);
    await HomePage.clickGetStarted();
    TopMenuPage = new TopMenuPage(page);
    // Assert
    await TopMenuPage.assertPageUrl(pageUrl);
  });
});
```



Duplication !!

```
test.describe('Playwright website', () => {  
  test('Check title in home page', async ({ page }) => {  
    await page.goto(URL);  
    homepage = new HomePage(page);  
    await homepage.assertPageTitle();  
  });  
  
  test('Check started link', async ({ page }) => {  
    // Act  
    await page.goto(URL);  
    homepage = new HomePage(page);  
  
    await homepage.clickGetStarted();  
    topMenuPage = new TopMenuPage(page);  
    // Assert  
    await topMenuPage.assertPageUrl(pageUrl);  
  });  
});
```



Use Test Life Cycle

```
test.beforeEach(async ({ page }) => {
  await page.goto(URL);
  homepage = new HomePage(page);
});
```



Third case

```
test('Check Java page', async ({ page }) => {
  await test.step('Act', async () => {
    await HomePage.clickGetStarted();
    topMenuPage = new TopMenuPage(page);
    await topMenuPage.hoverNode();
    await topMenuPage.clickJava();
  });
  await test.step('Assert', async () => {
    await topMenuPage.assertPageUrl(pageUrl);
    await topMenuPage.assertNodeDescriptionNotVisible();
    await topMenuPage.assertJavaDescriptionVisible();
  });
});
```

<https://playwright.dev/docs/api/class-test#test-step>



API Testing

<https://playwright.dev/docs/api-testing>



Make it Fast



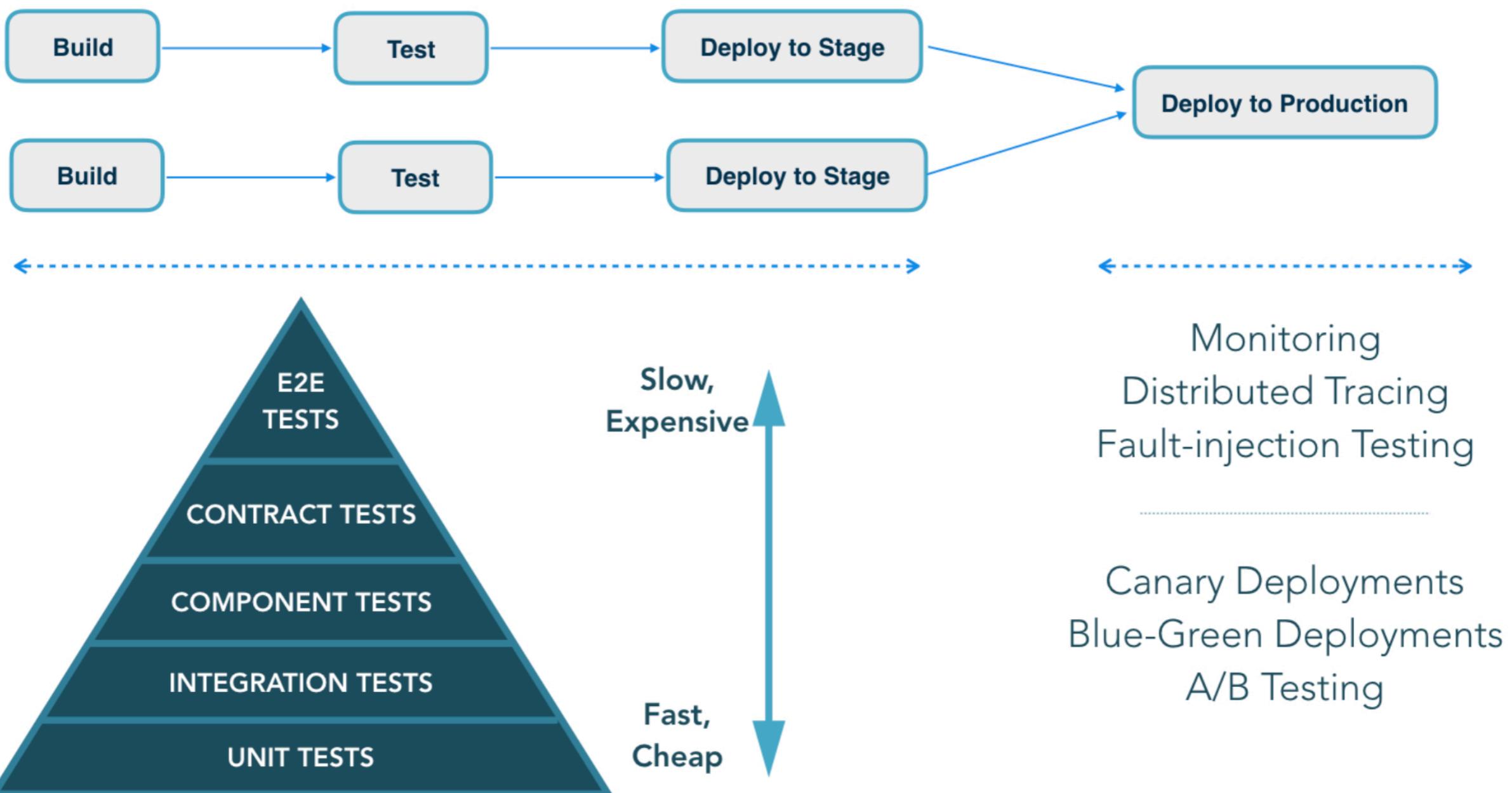
Continuous Integration



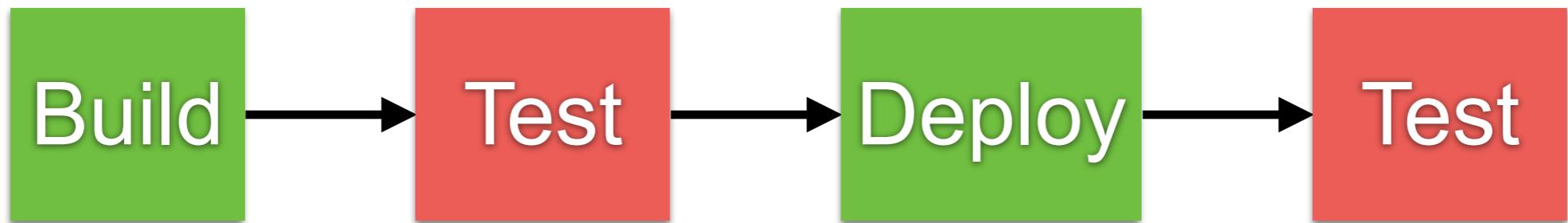
Continuous integration



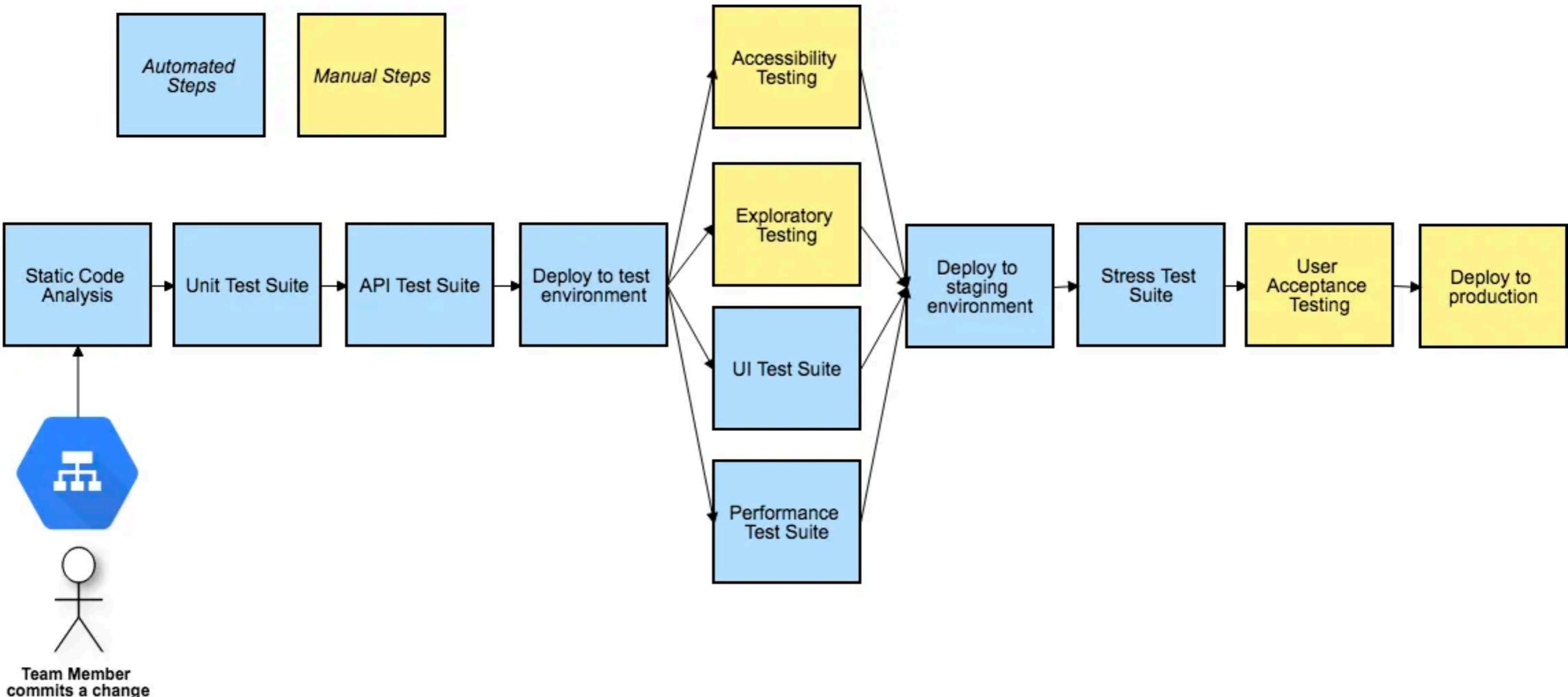
Test strategy



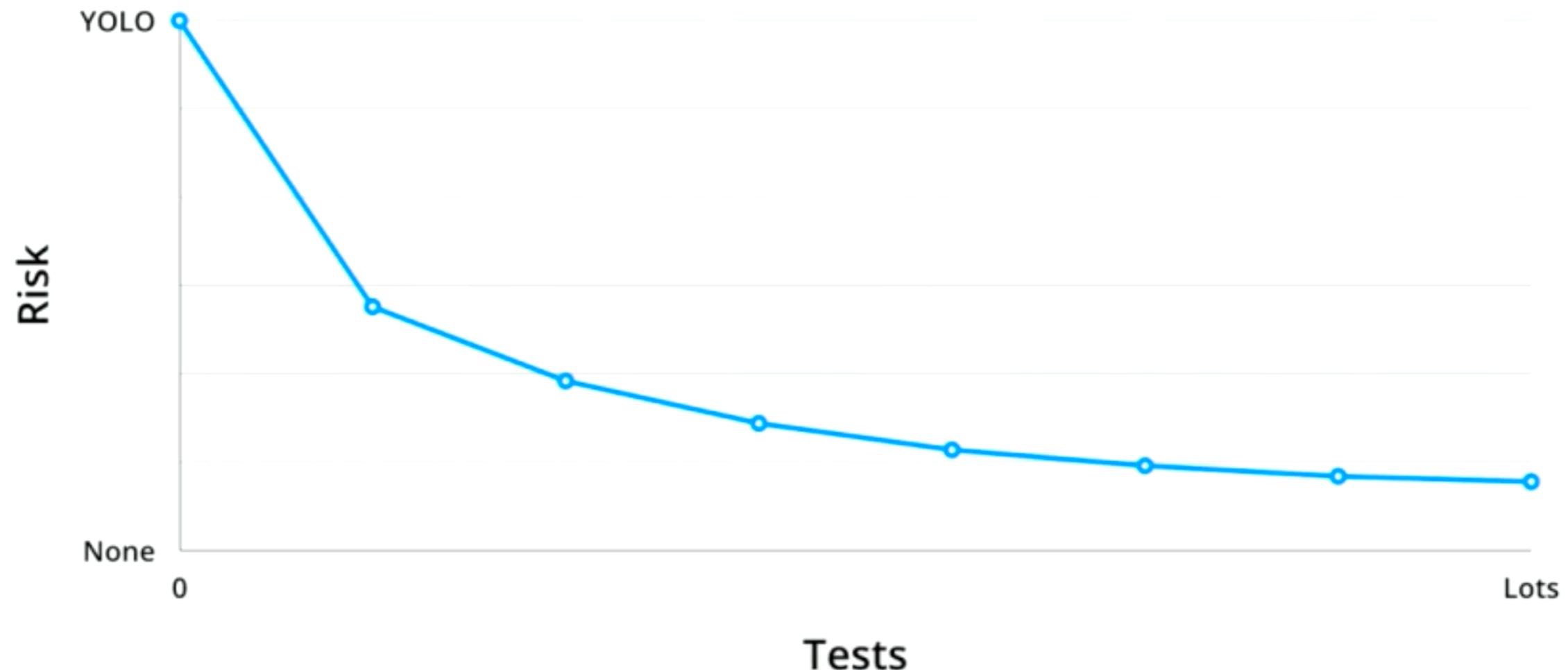
Design your pipeline



Design your pipeline/process



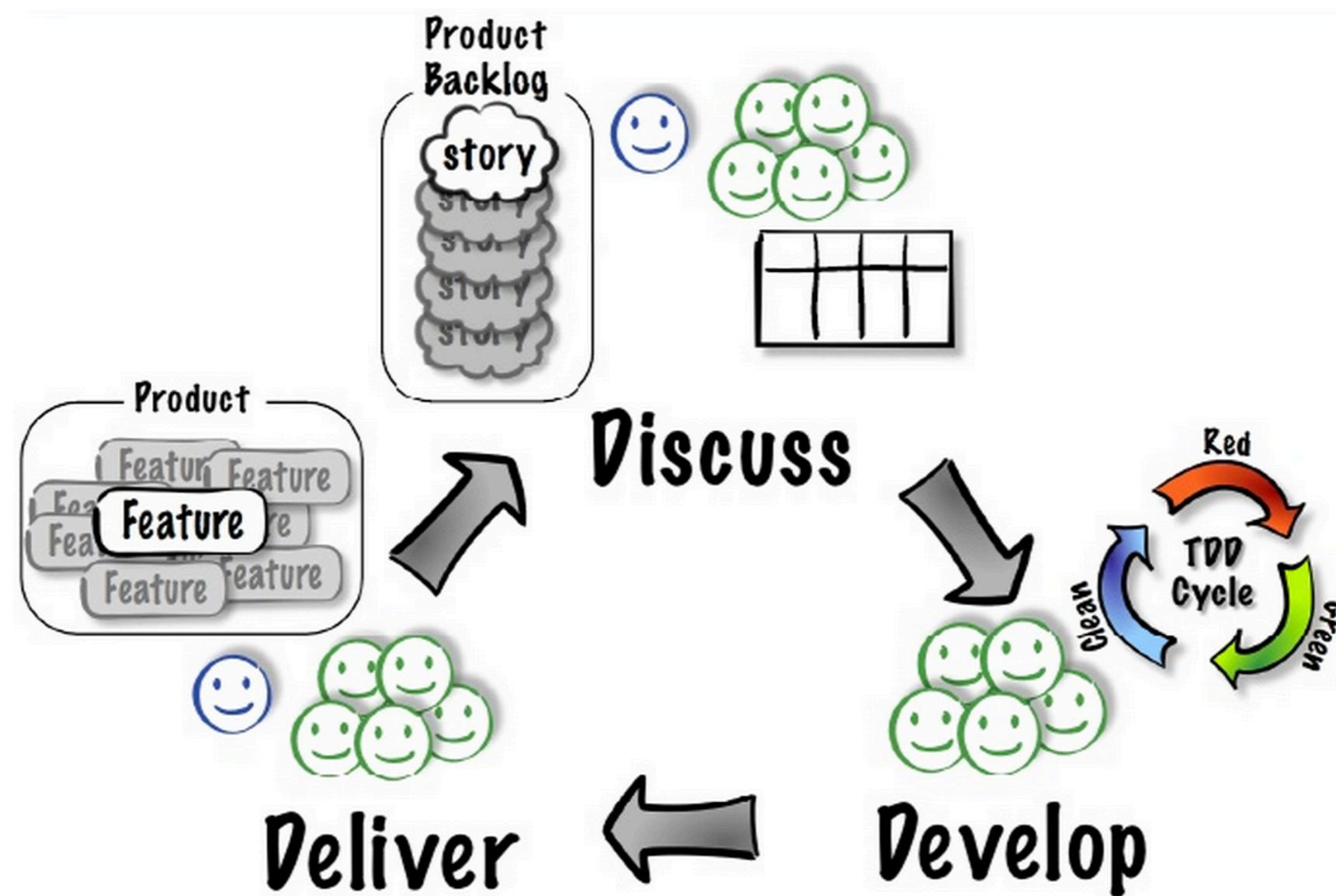
Reduce risk with tests



THINK before coding



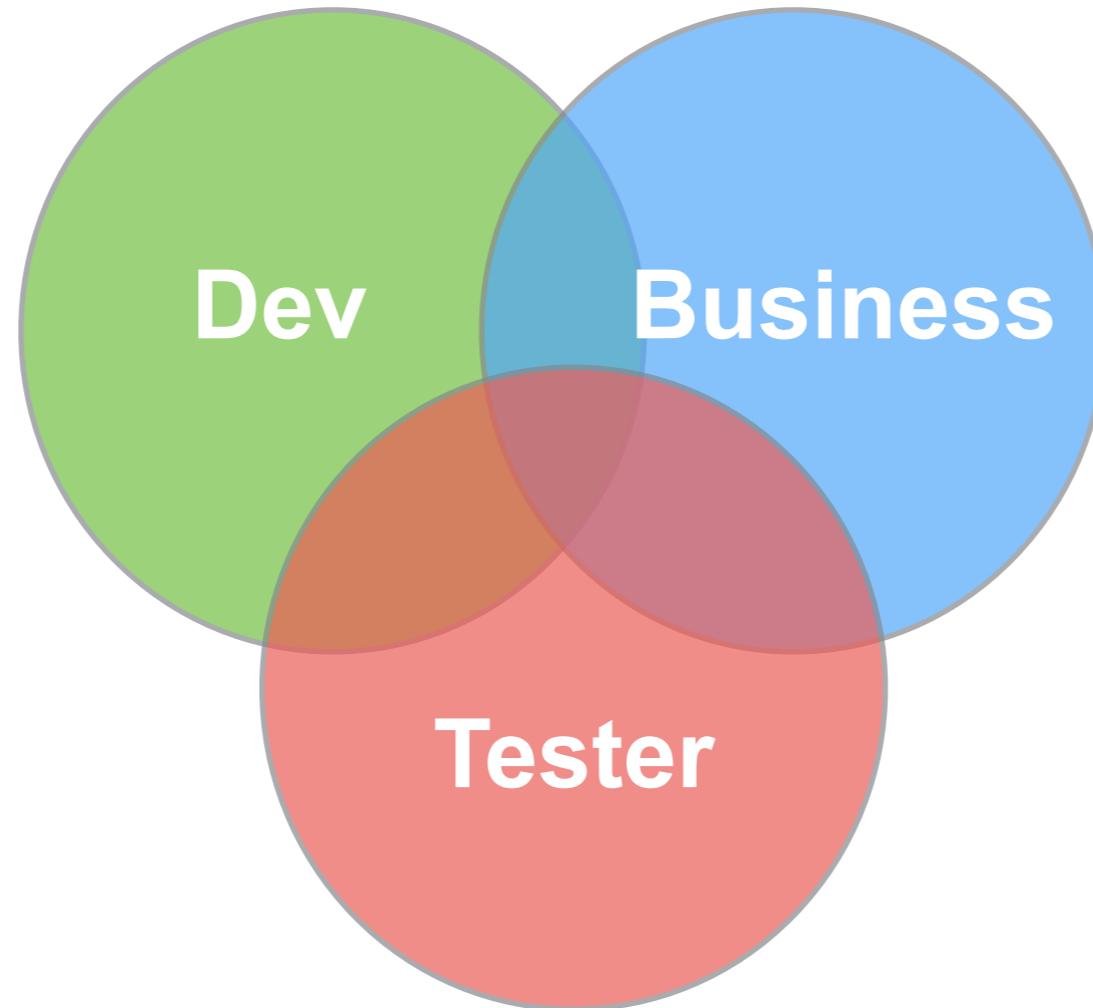
Acceptance Test-Driven Development



(Model developed with Pekka Klärck, Bas Vodde, and Craig Larman.)



Acceptance Test-Driven Development



Acceptance Tests

=

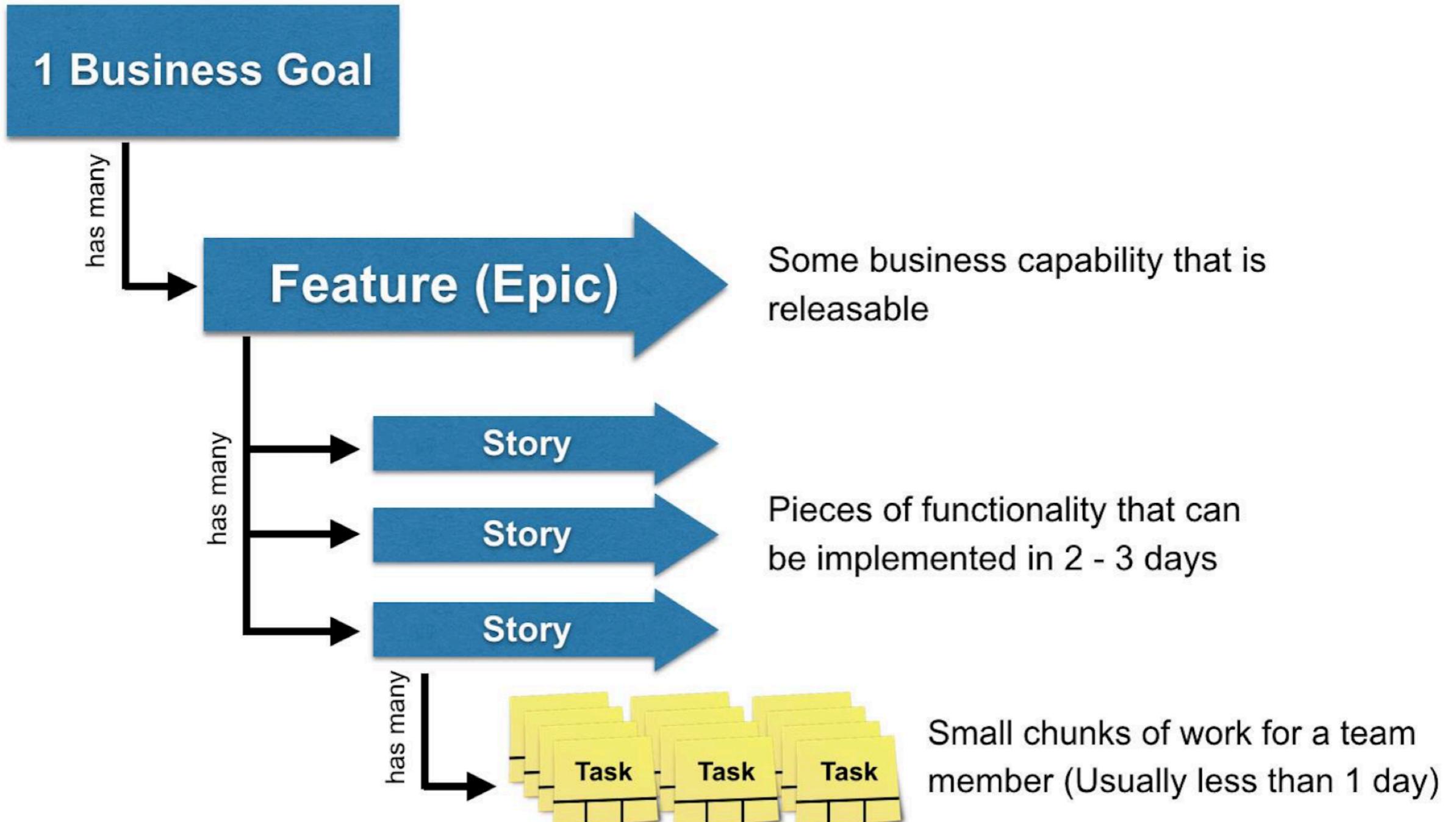
Business Criteria

+

Examples (data)



Work break down



Iterative and incremental process

Feature 1

Time



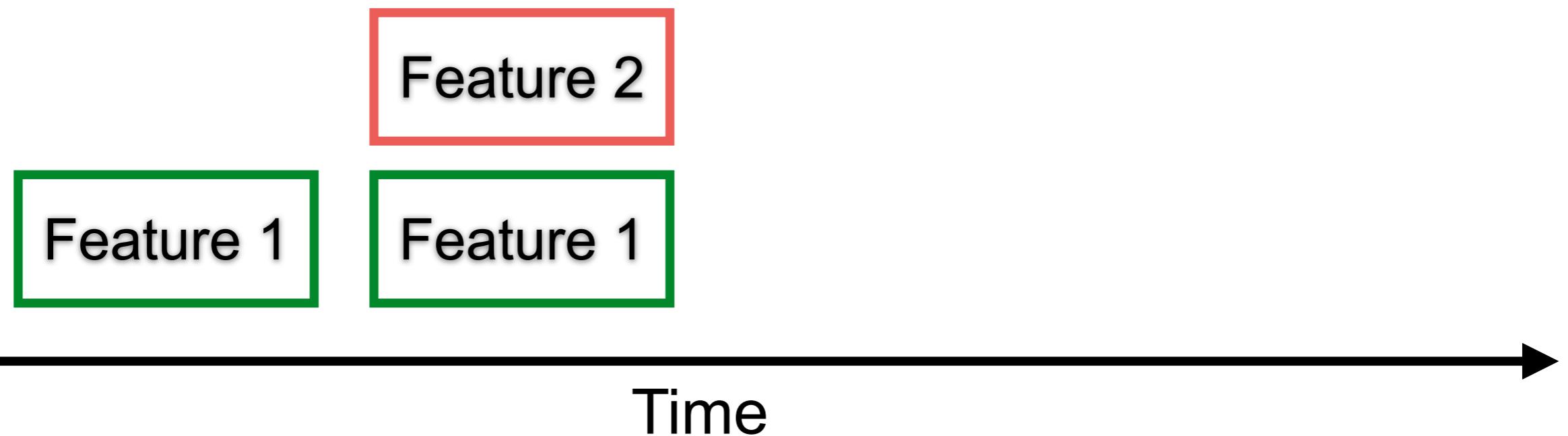
Iterative and incremental process

Done = coded and tested



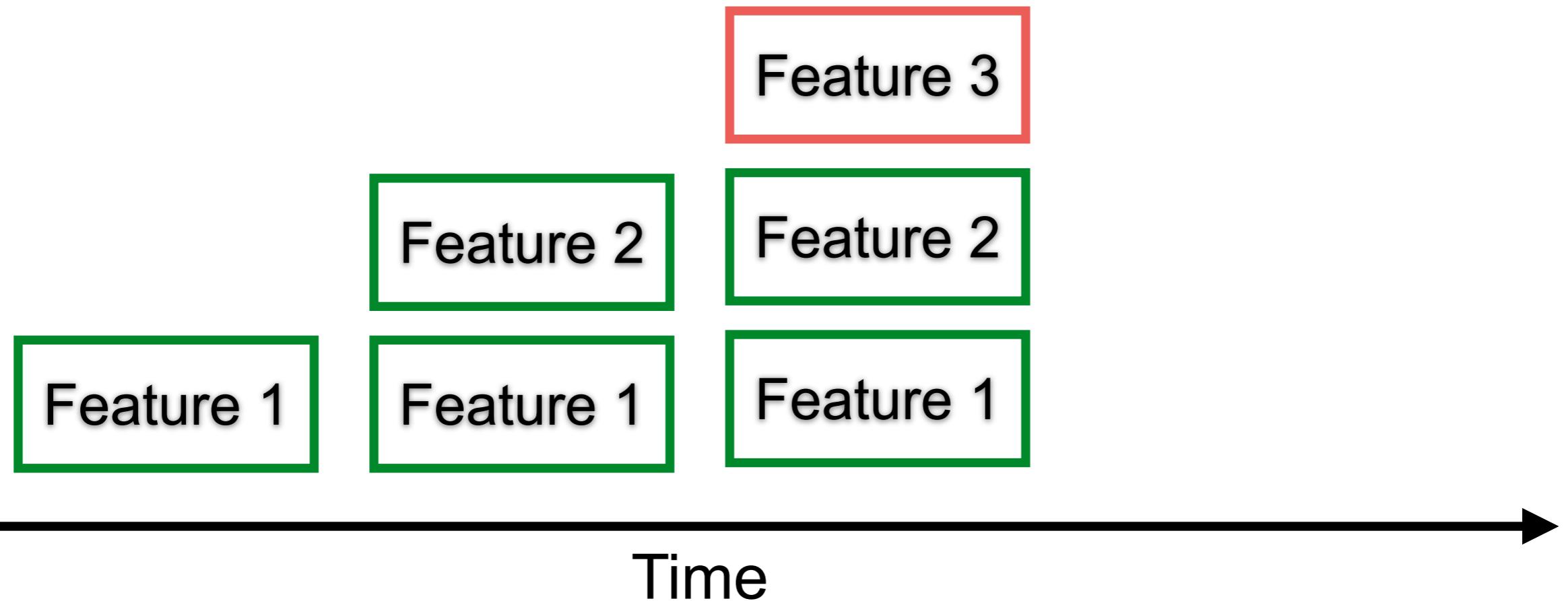
Iterative and incremental process

Done = coded and tested



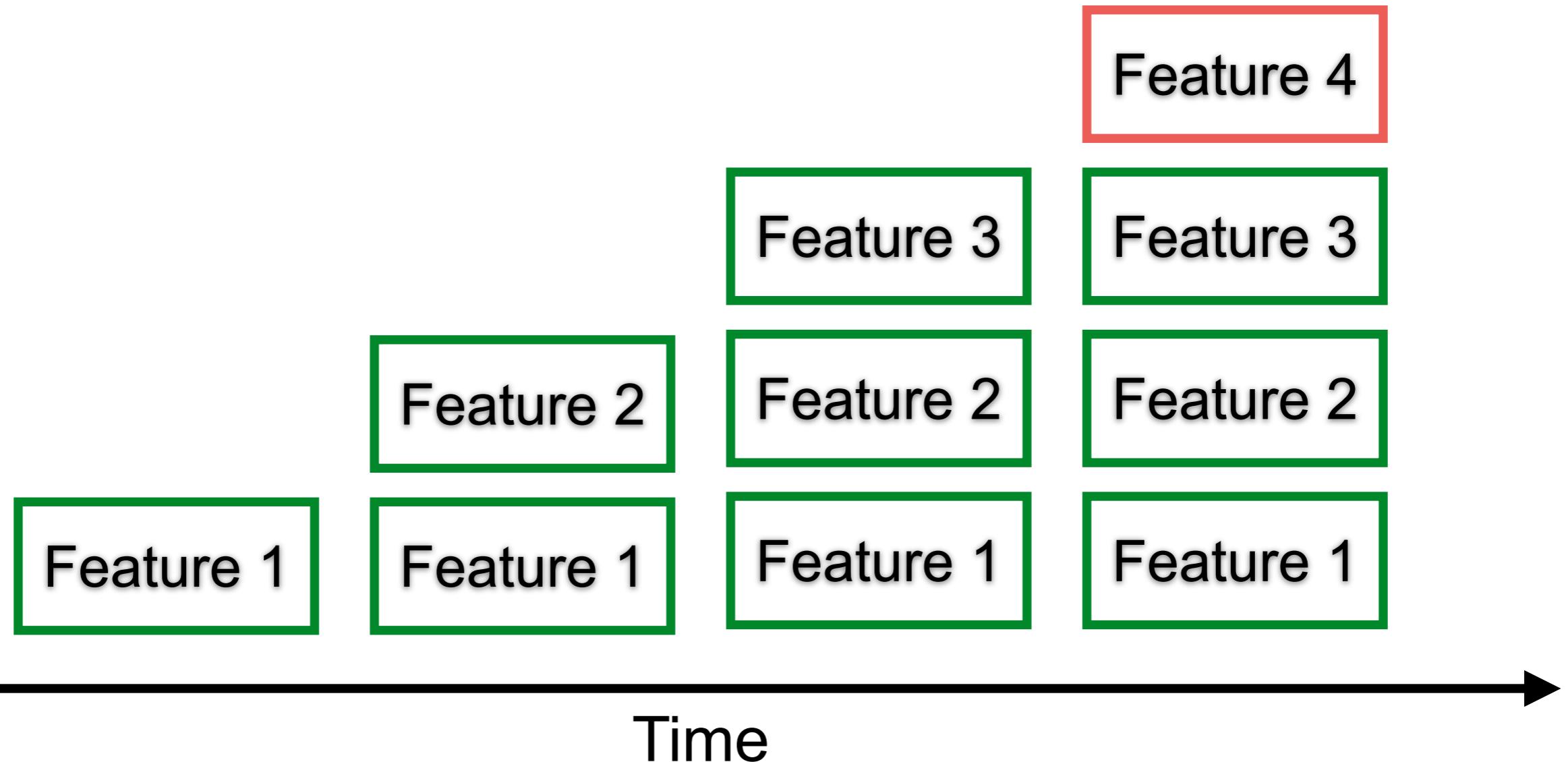
Iterative and incremental process

Done = coded and tested



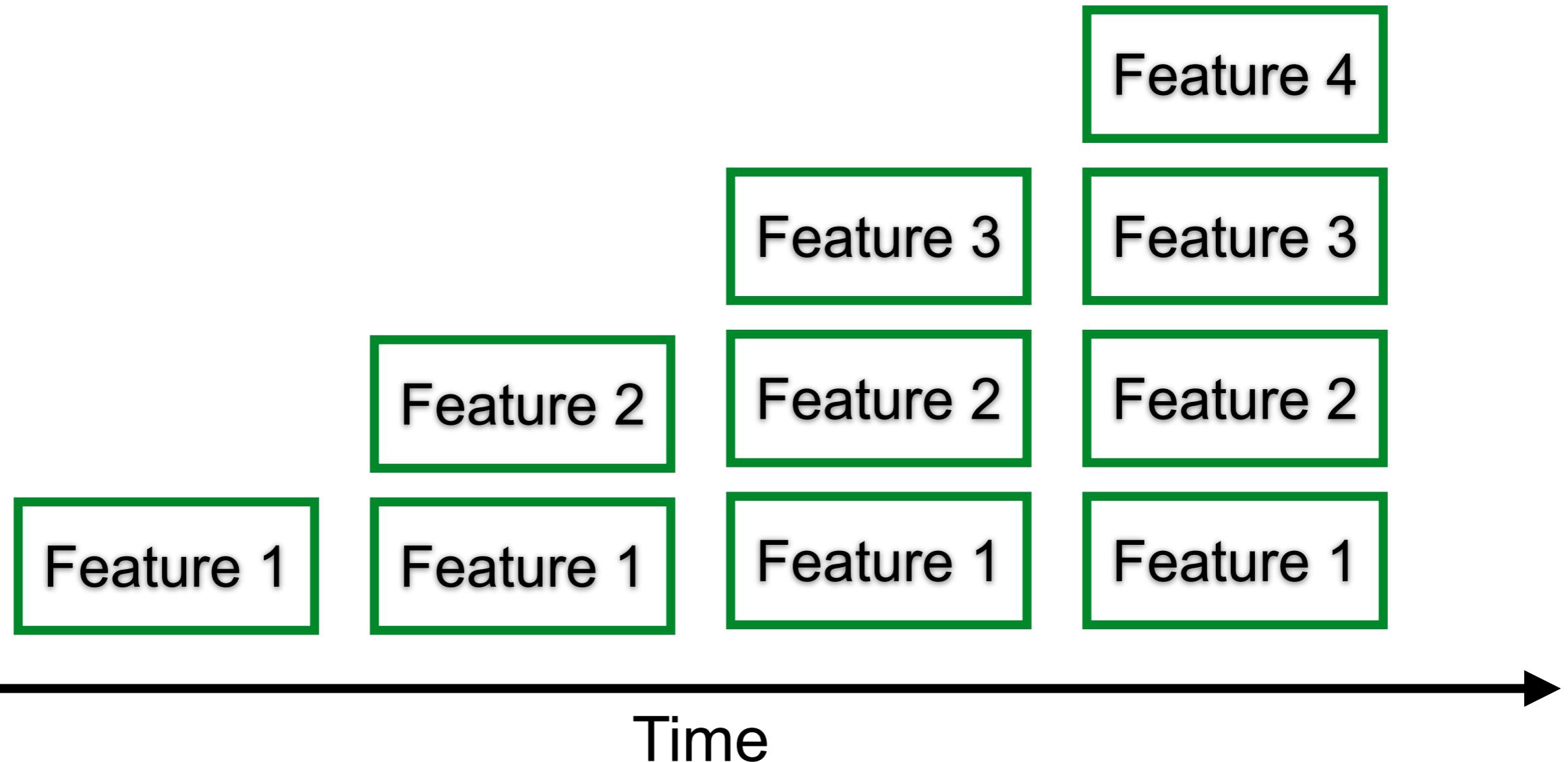
Iterative and incremental process

Done = coded and tested



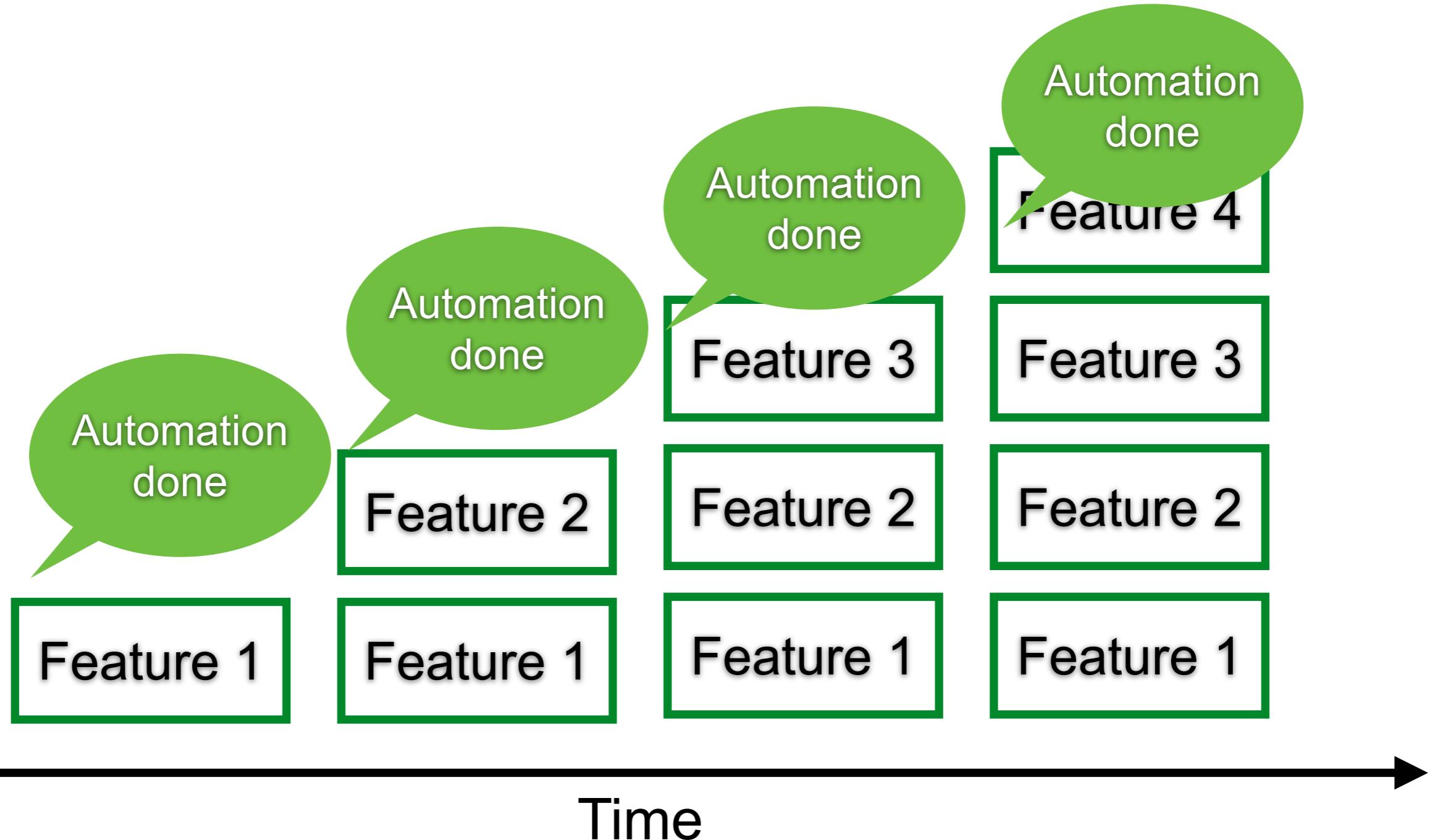
Iterative and incremental process

Done = coded and tested



Iterative and incremental process

Done = coded and tested

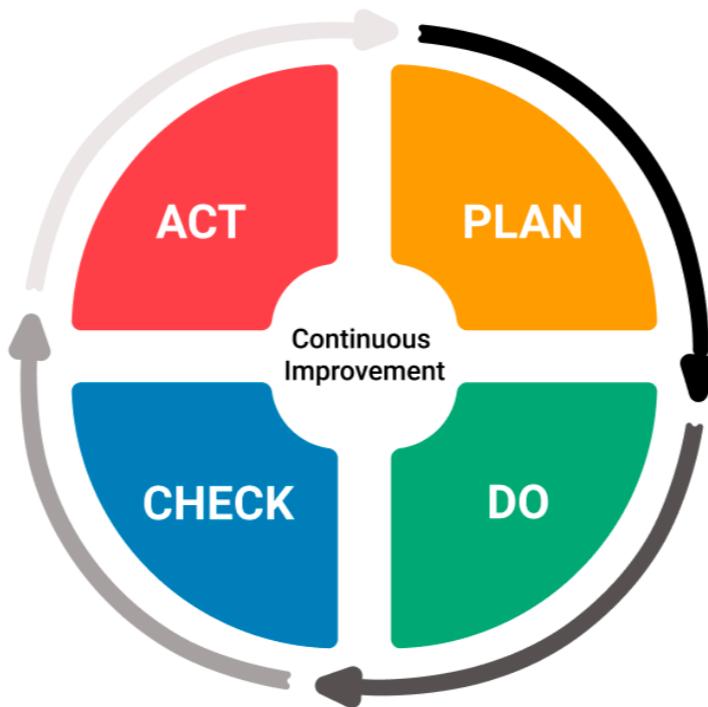


Automation feedback

Easier over time ?

Time spent on maintenance ?

Test find regression bugs ?



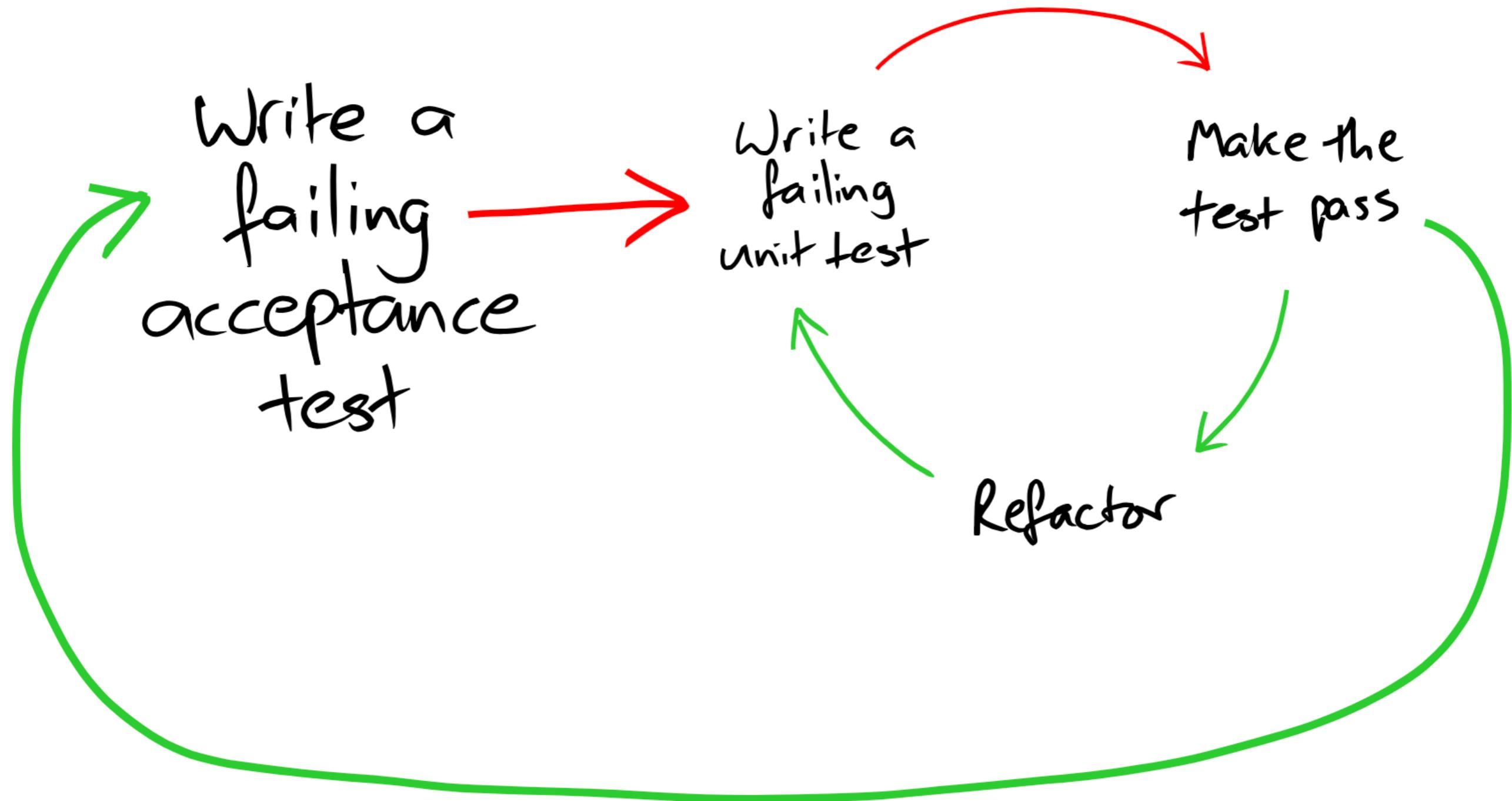
Start with simple



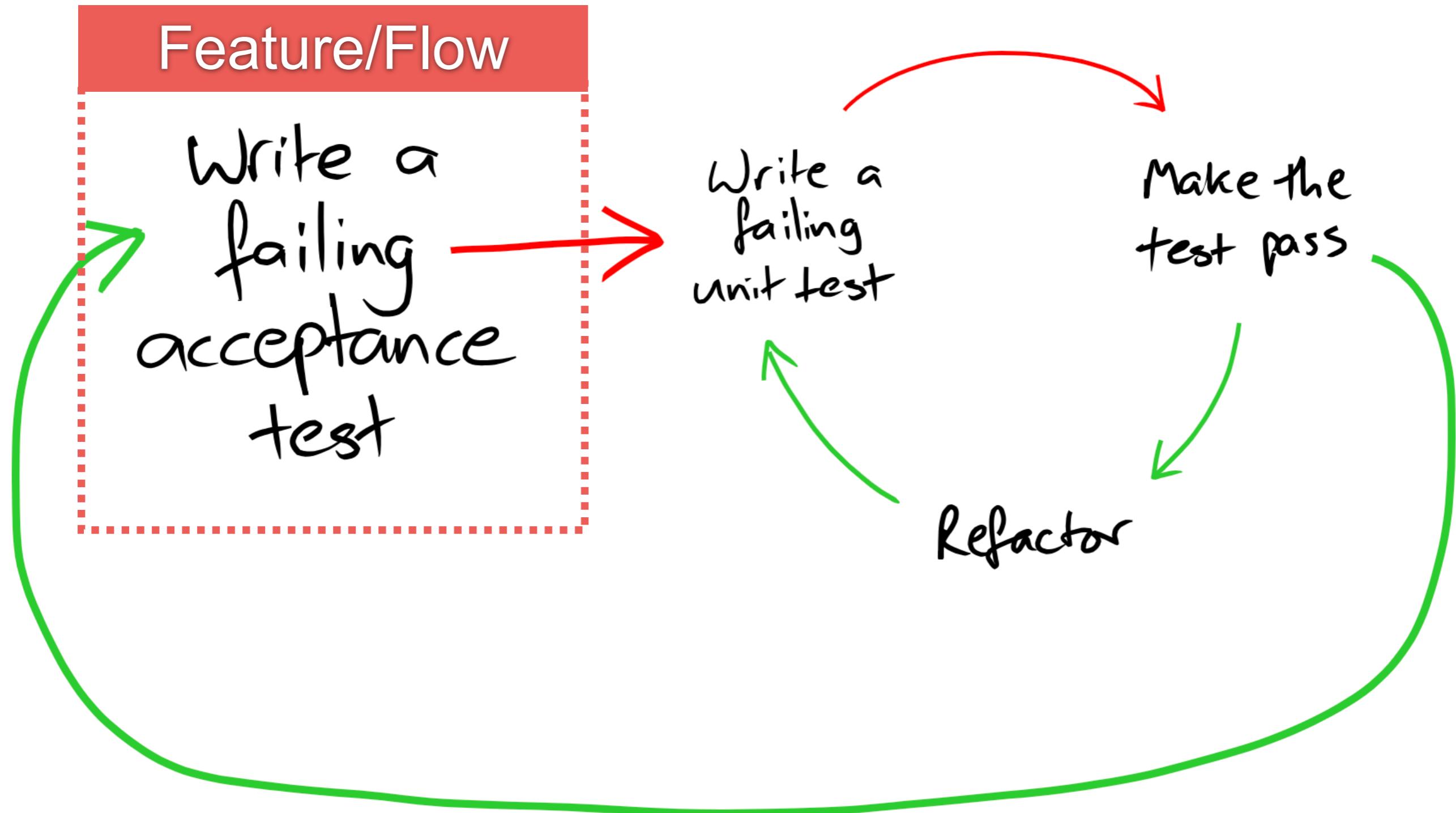
Use **feedback** to improve



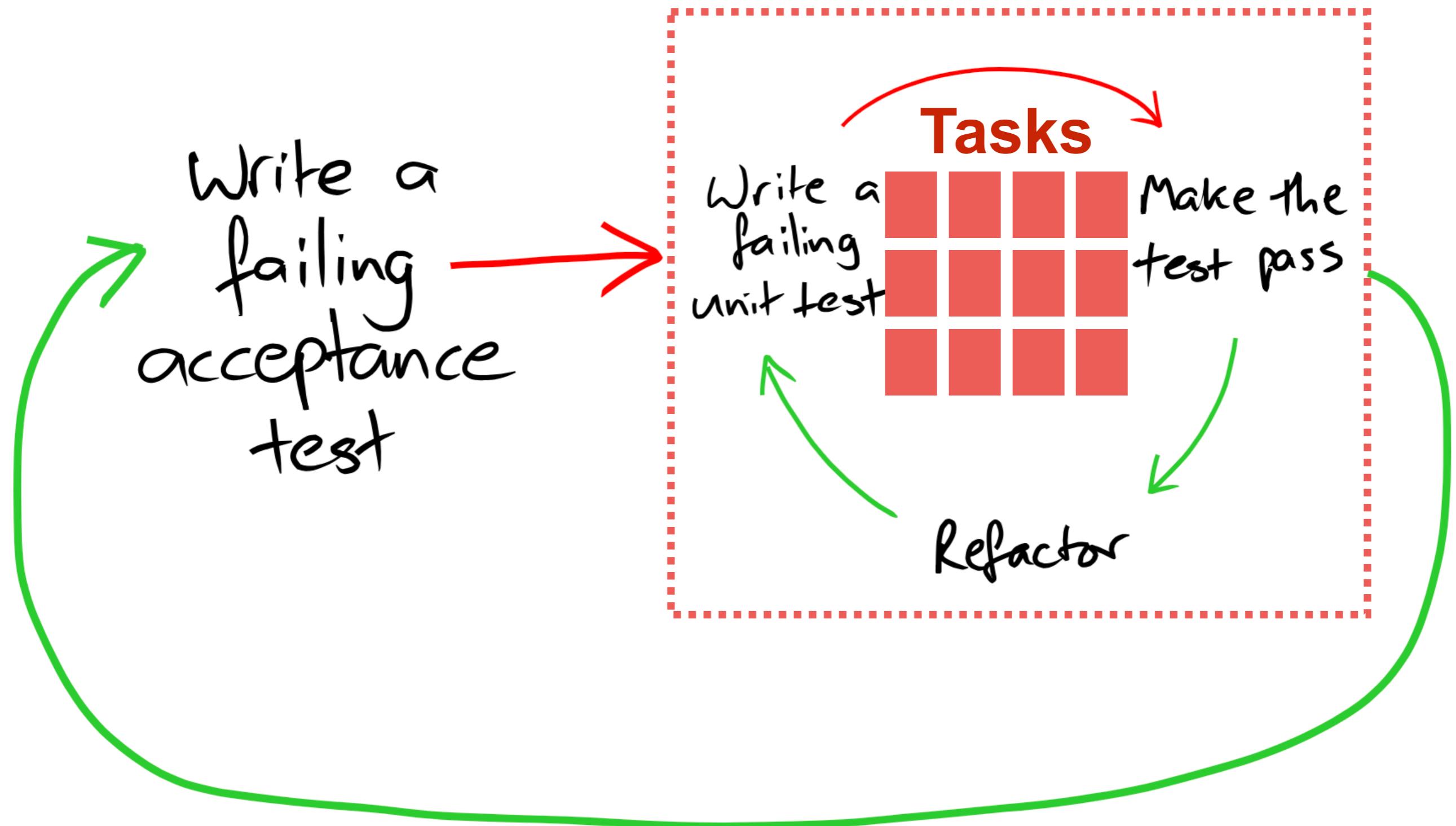
Outside-in develop/test



Outside-in develop/test



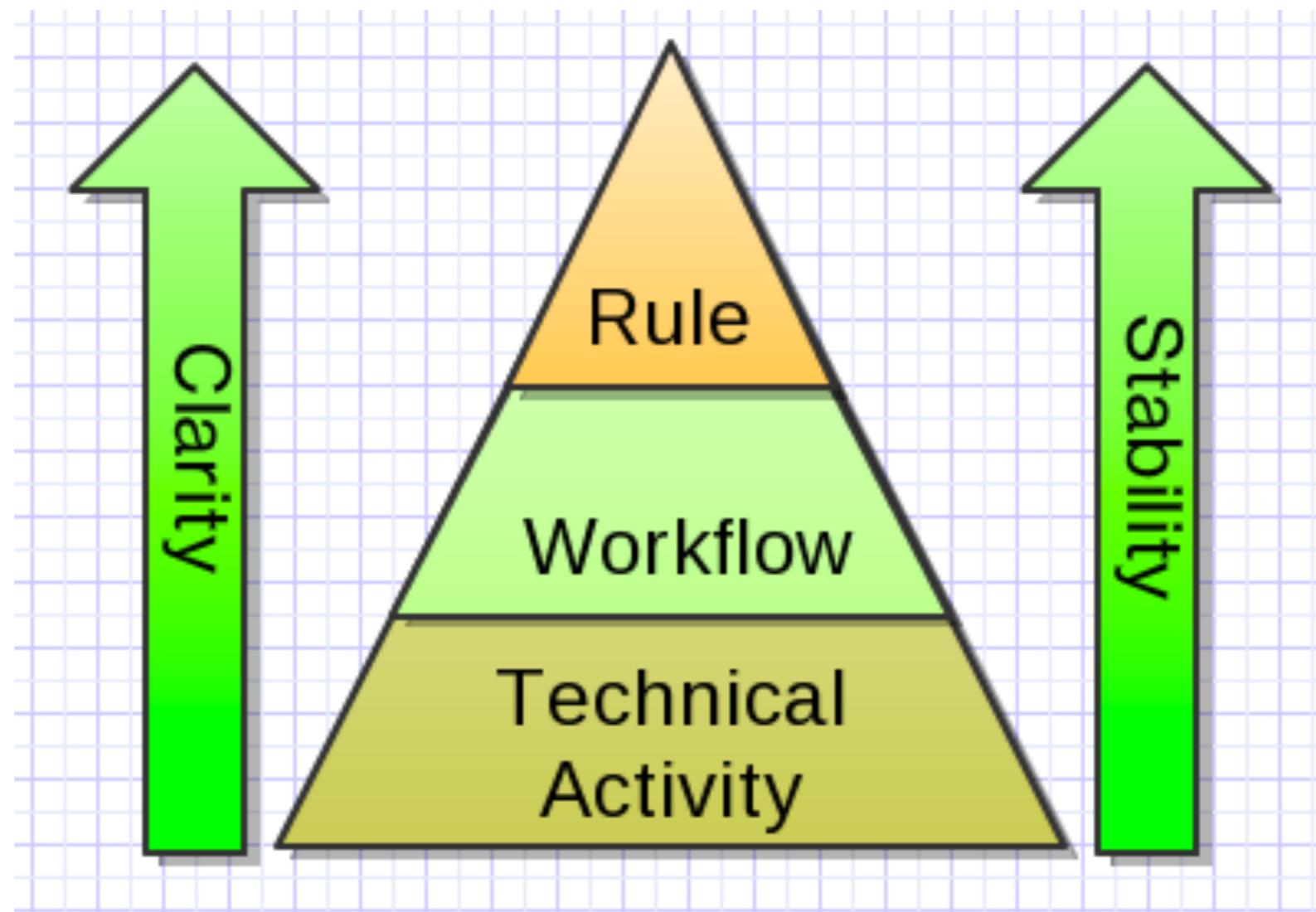
Outside-in develop/test



UI Test Automation



3 levels of UI test automation



3 levels of UI test automation

Business rule/functionality level

what is this test demonstrating or exercising

User interface workflow level

what does a user have to do to exercise the functionality through the UI

Technical activity level

what are the technical steps required to exercise the functionality

