ITERA

Who are we?



Susanne Semsøy

- Test Manager and Teamlead in Itera
- Summer intern (2021) and part-time (2022) and Boost graduate in Itera
- ISTQB Foundation and Agile Tester certified + Scrum Master (CSM)
- PROSA Bachelor + Master (UiO 2022)



Mia Bjørndal

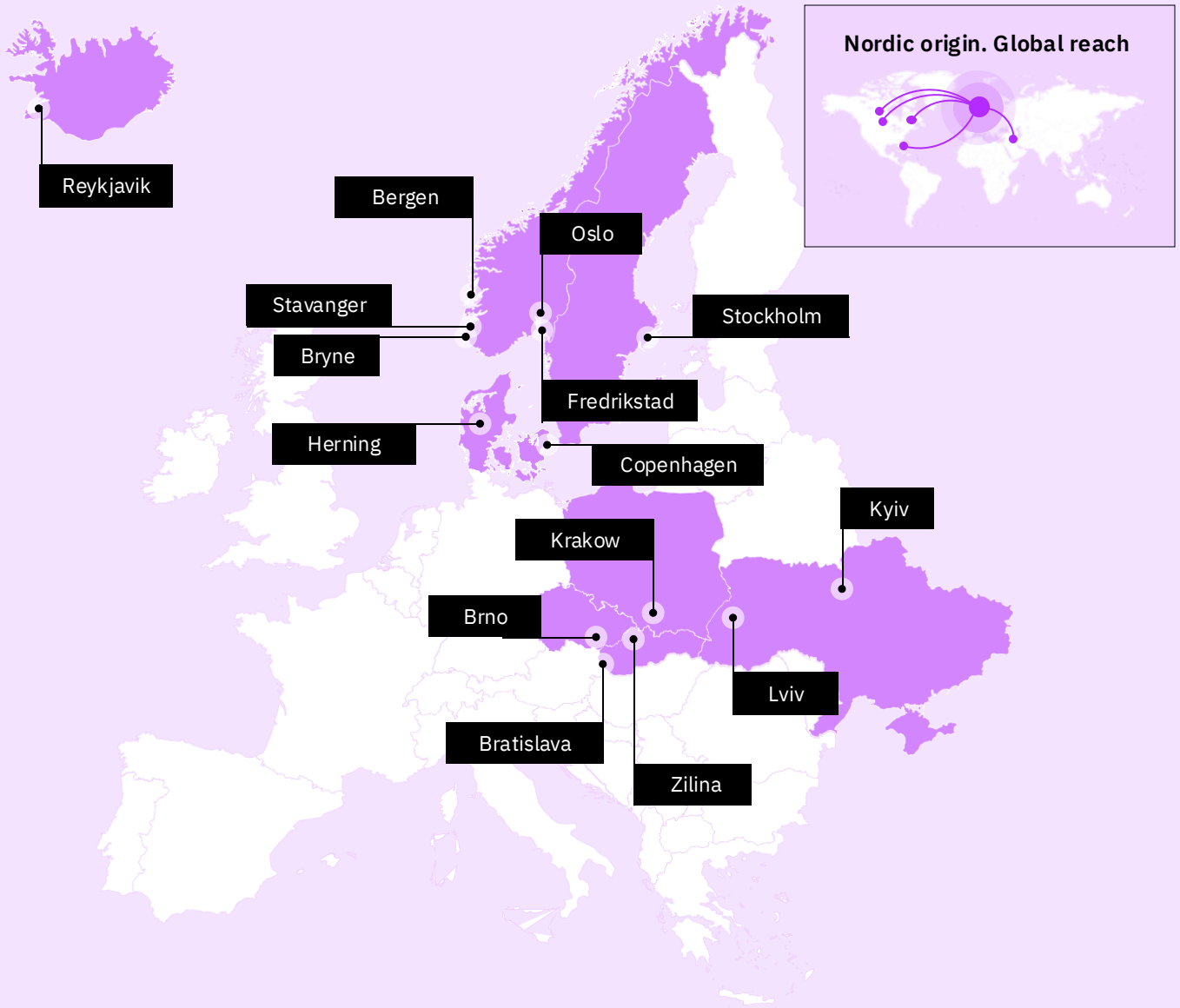
- Boost graduate in Itera (2025)
- ISTQB Foundation certified
- Informatics Bachelor + Master from NTNU (2025)

Itera at a glance

Care. Challenge. Create.
That's how change happens.

650+
people

14
offices



Our vision
Make a difference


AON
6
THUNE

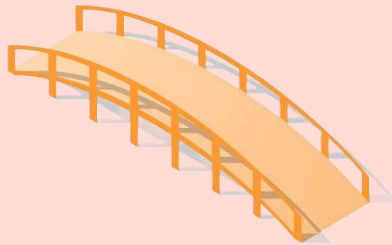
Agenda

1. What is it like to work as a tester/QA?
2. Introduction to test automation
3. Approaches to test automation
4. Test automation in the agile SDLC
5. Frameworks
6. Practical demo

What is it like to work as a tester/QA?

A broad, flexible role

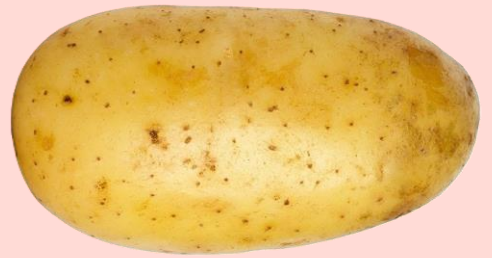
- Works across many parts of a project
- Understands the whole system
- Focuses on quality, not just finding errors
- Interdisiplinary. Collaborates with many roles.





A detective mindset

- Investigates problems
- Connects information across components
- Spots patterns and risks early
- Find creative ways to be destructive >:D

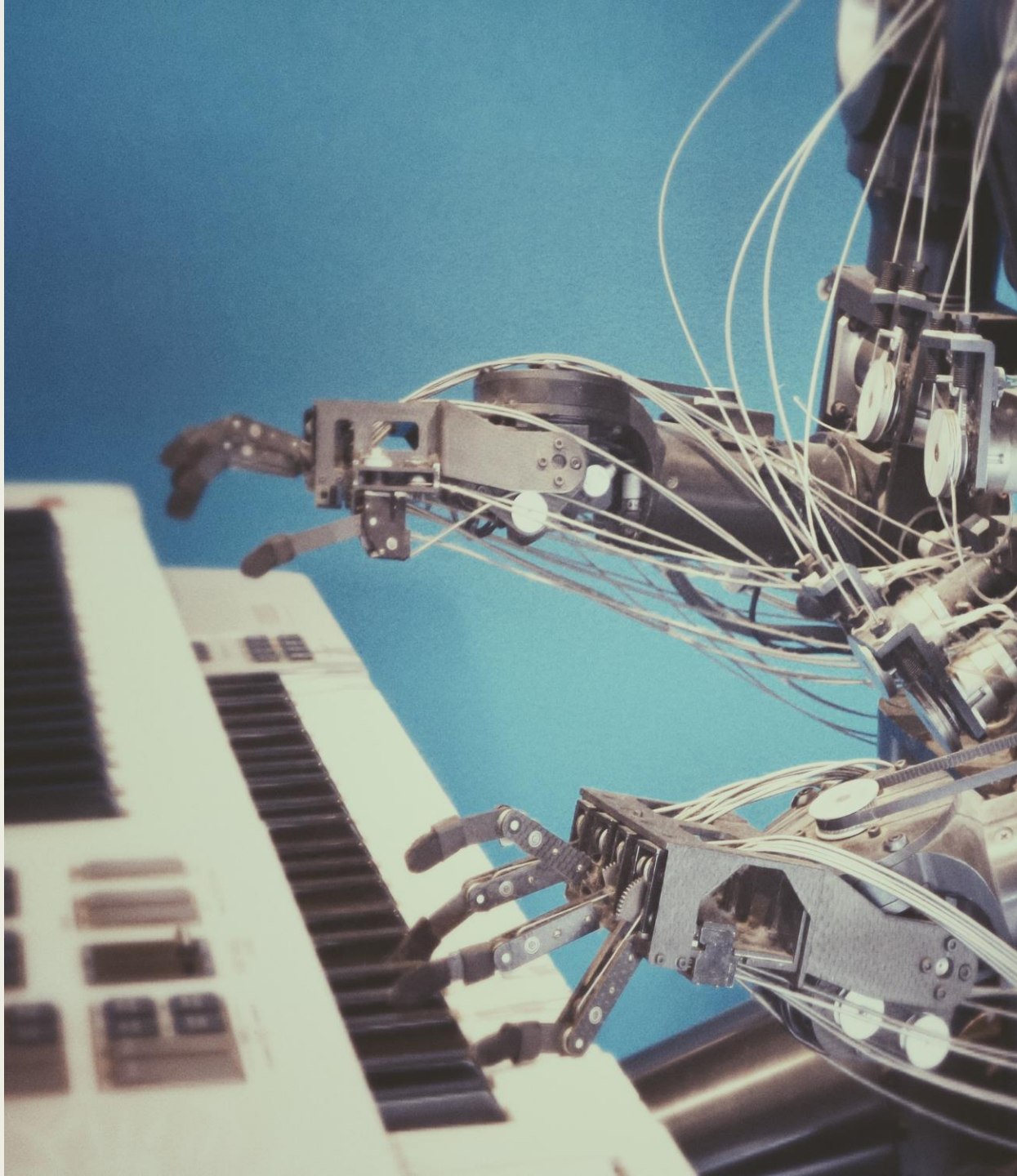


Many career paths / specializations

- Test manager
- Automation engineer
- Non-functional tester
- Etc...

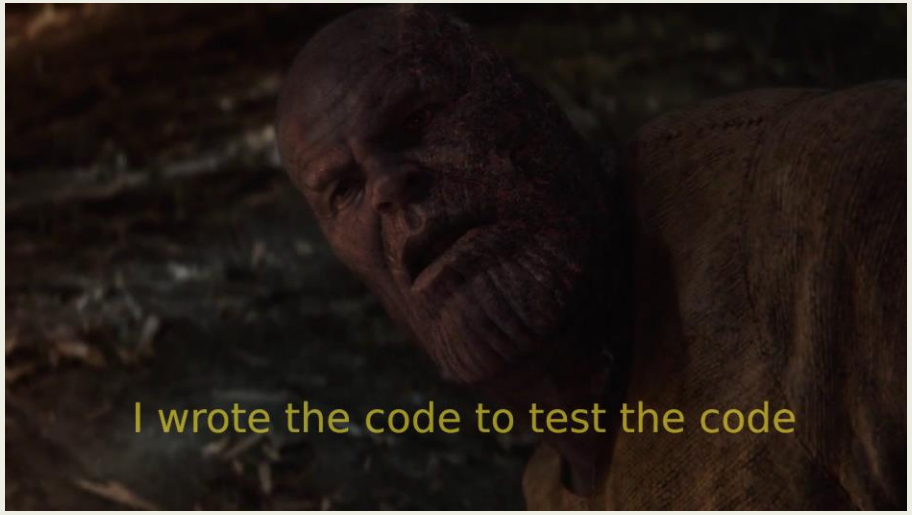
Test Automation

Guest lecture with Itera

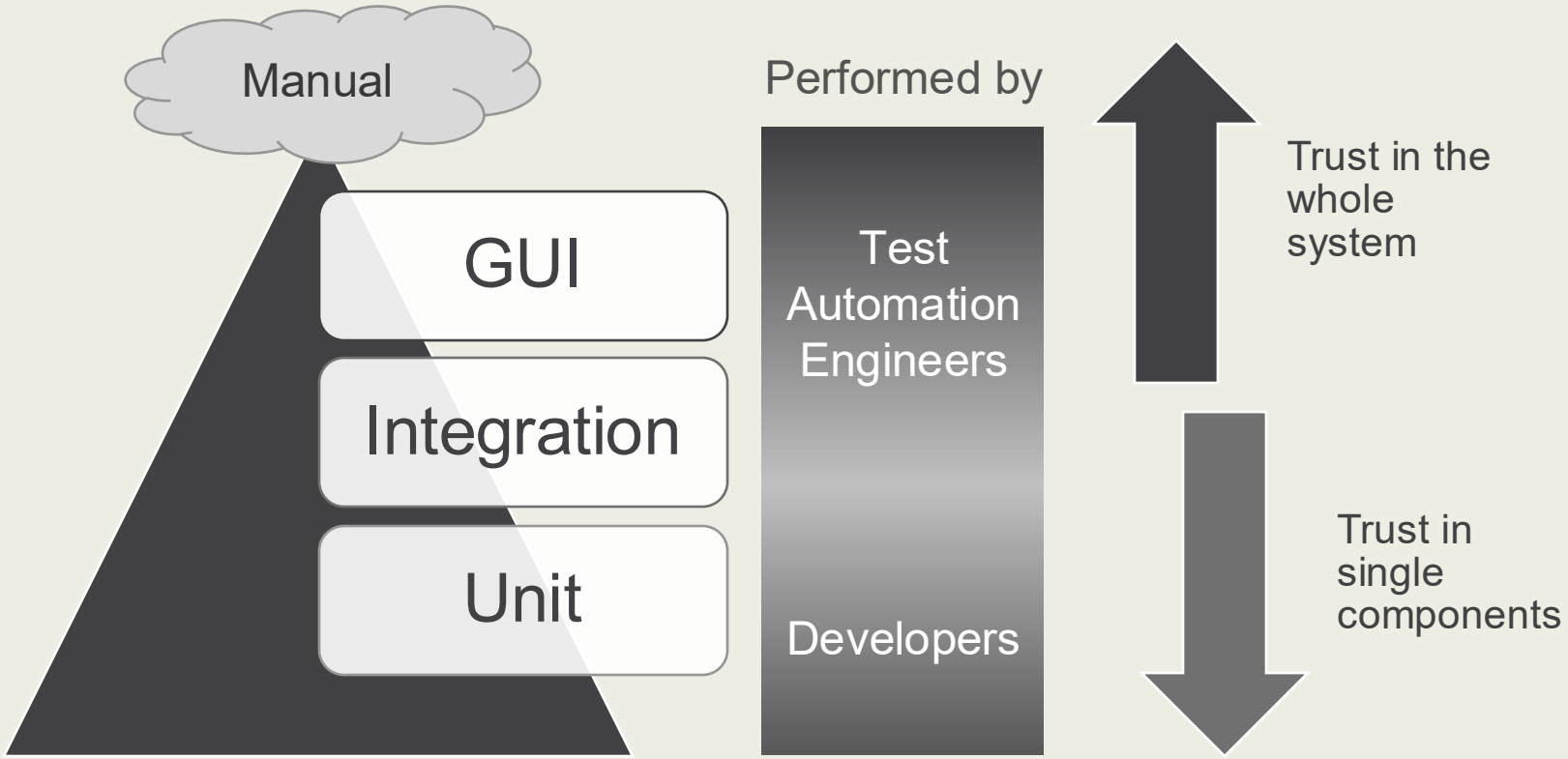


What is test automation?

Test automation *refers to the* ***use*** *and* ***development*** *of* ***specialized software*** *(separate from the software being tested) to* ***automatically execute*** *tests on software applications, as opposed to performing them manually.*



Levels of test automation



Why test automation?



Increased Efficiency: Automated tests can run much faster than manual tests and therefore run more often. This allows for quicker feedback regarding software quality.



Improved Accuracy: Automation reduces the risk of human error that can occur during manual testing.



Reusability: Once created, automated tests can be reused across different versions of the application and across several browsers



Continuous Testing: Automation facilitates continuous integration and delivery practices by enabling frequent testing with minimal effort.



Documentation: Automation allows for documentation of requirements and generation of test result reports

Why not?

- Initial investment required
- Additional costs
- Technical skill required
- Maintenance requirement
- Distract from testing objectives
- False alarms

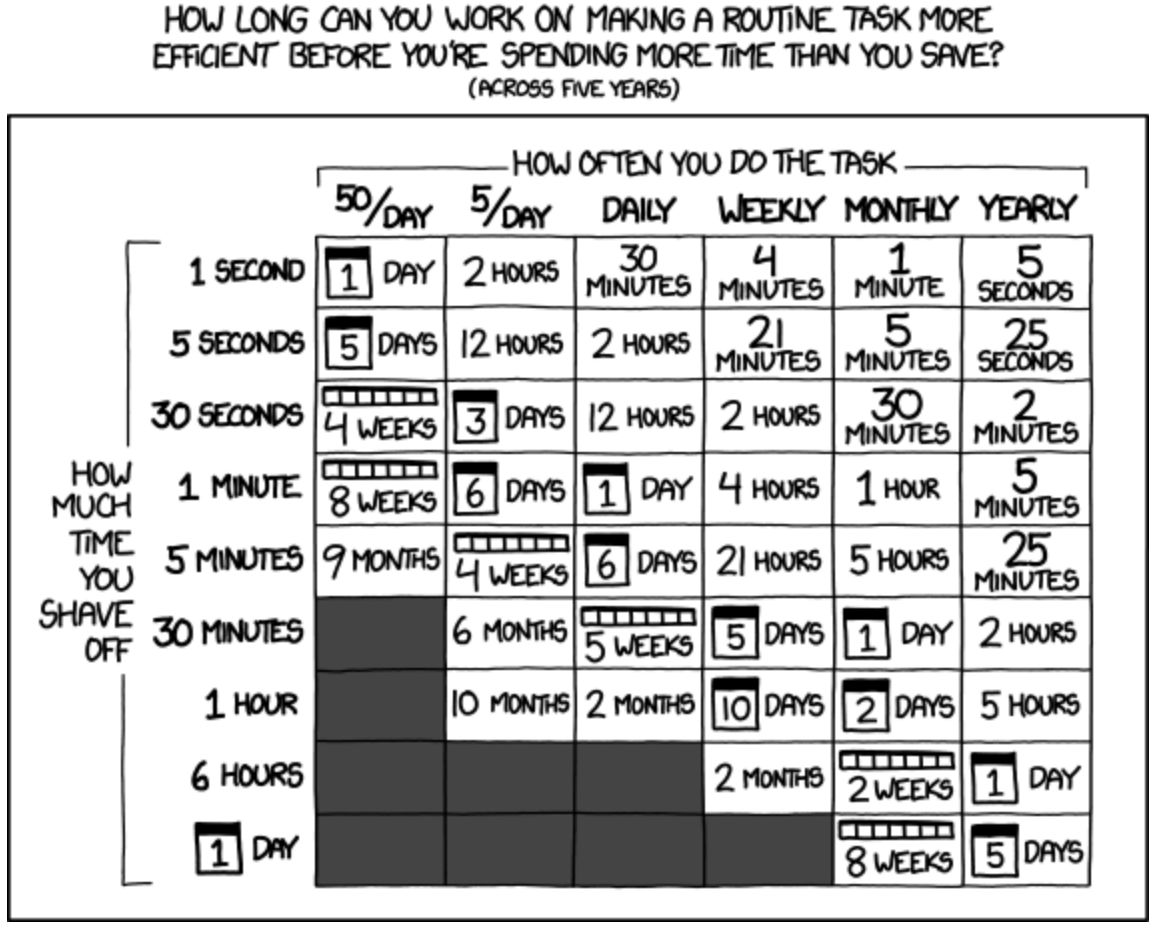
Test Automation Limitations

- Not all tests **can** be automated
- Not all tests **should** be automated
- Not a replacement for exploratory testing
- Not feasible to automate systems that:
  - Changes often and drastically
  - Soon to be retired
  - Small in size



Spend 10 minutes doing the task manually

Spend 10 hours writing code to automate it



AUTOMATION VS. MANUAL TESTING



Combine test automation with manual testing!

Risk Level	Automated Tests	Exploratory Tests	Black-box Tests
High	++	+	++
Medium	+	+	+
Low	o	++	+

Table 1: Mission and/or Safety Critical Systems

Risk Level	Automated Tests	Exploratory Tests	Black-box Tests
High	+	++	+
Medium	o	++	o
Low	--	++	--

Table 2: Non-Mission and/or Non-Safety Critical Systems.









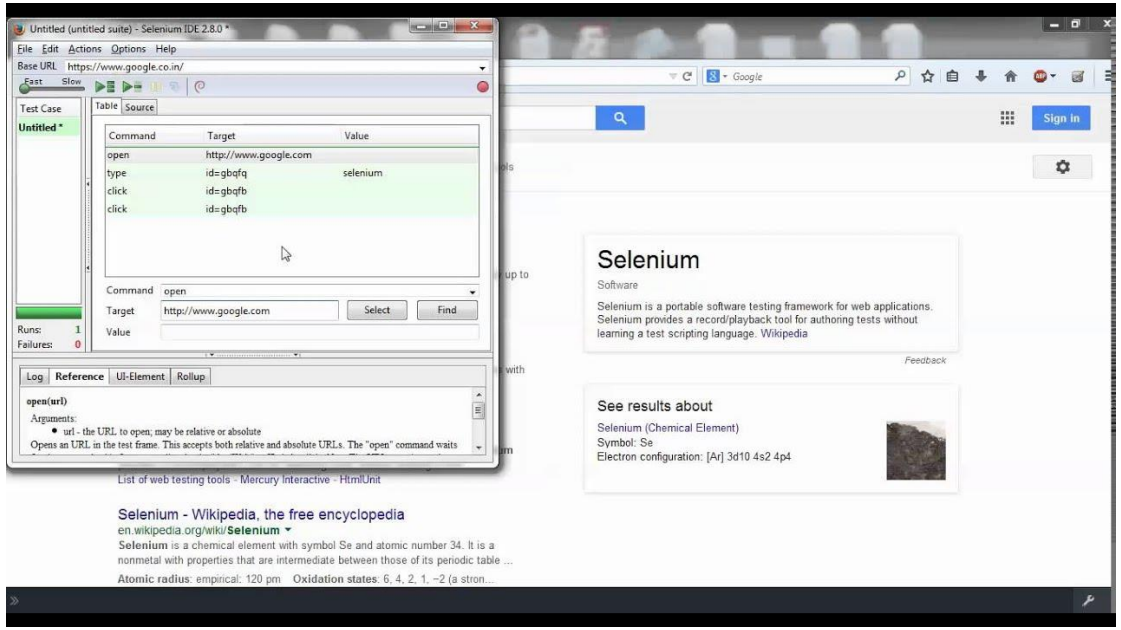
Test Automation Approaches

Three main approaches for test automation:

- **Capture & playback**
- **Structured scripting**
- **Model-based scripting**

Capture & Playback

- Tool will ‘record’ actions and interactions from the user
- Save the recording as one single test session
- Tool will playback the actions and interactions recorded
  - Pixel by pixel
  - Screen resolution dependent
- Easy to setup and use
- Lacks abstraction
- Increases maintenance



Selenium

Structured Scripting

- Starts with manual test procedures
- Transform them into automated test scripts
- Has a scripting library (functions, classes, etc)
- Has abstraction
- Improves adaptability and maintainability
- Requires initial investment to create script library
- Requires programming skills



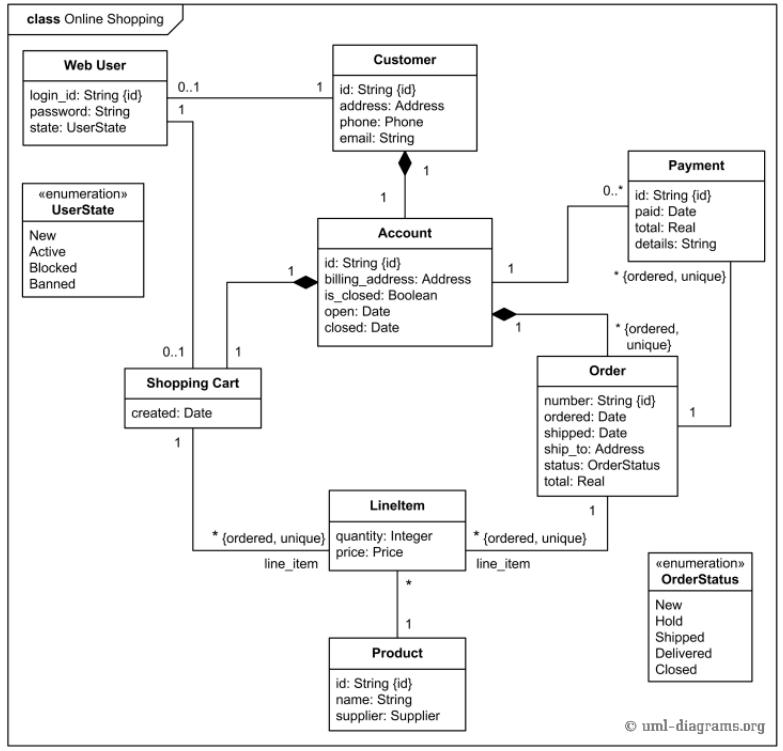
cypress

Playwright

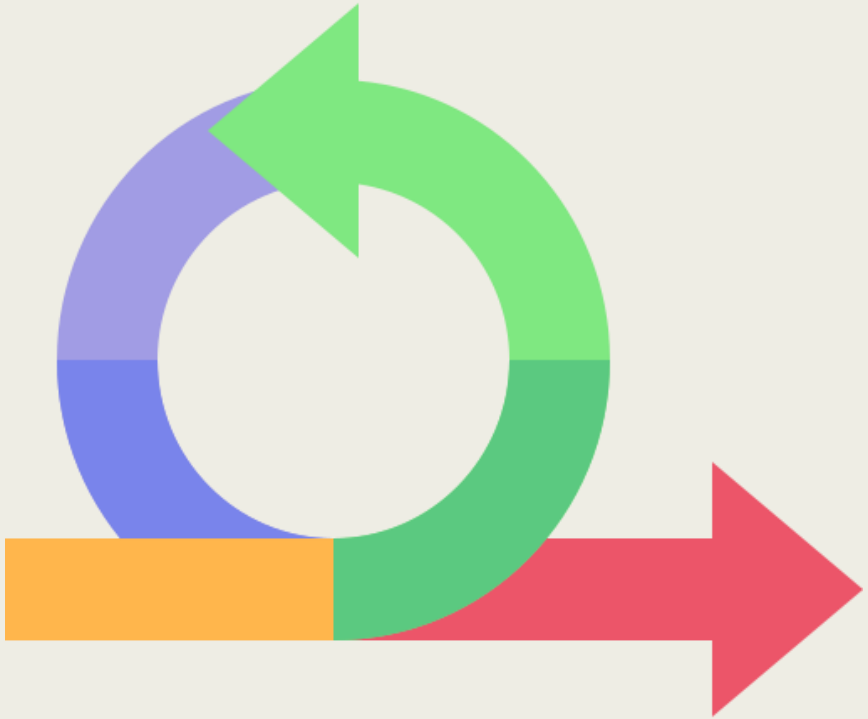
Model-based Scripting

- Automatically generate test cases & test scripts based off the system's model

- Generating tests for different systems
- Technology independence
- Requires modeling expertise
- Model-based testing tools are not yet mainstream



Test automation in the agile SDLC



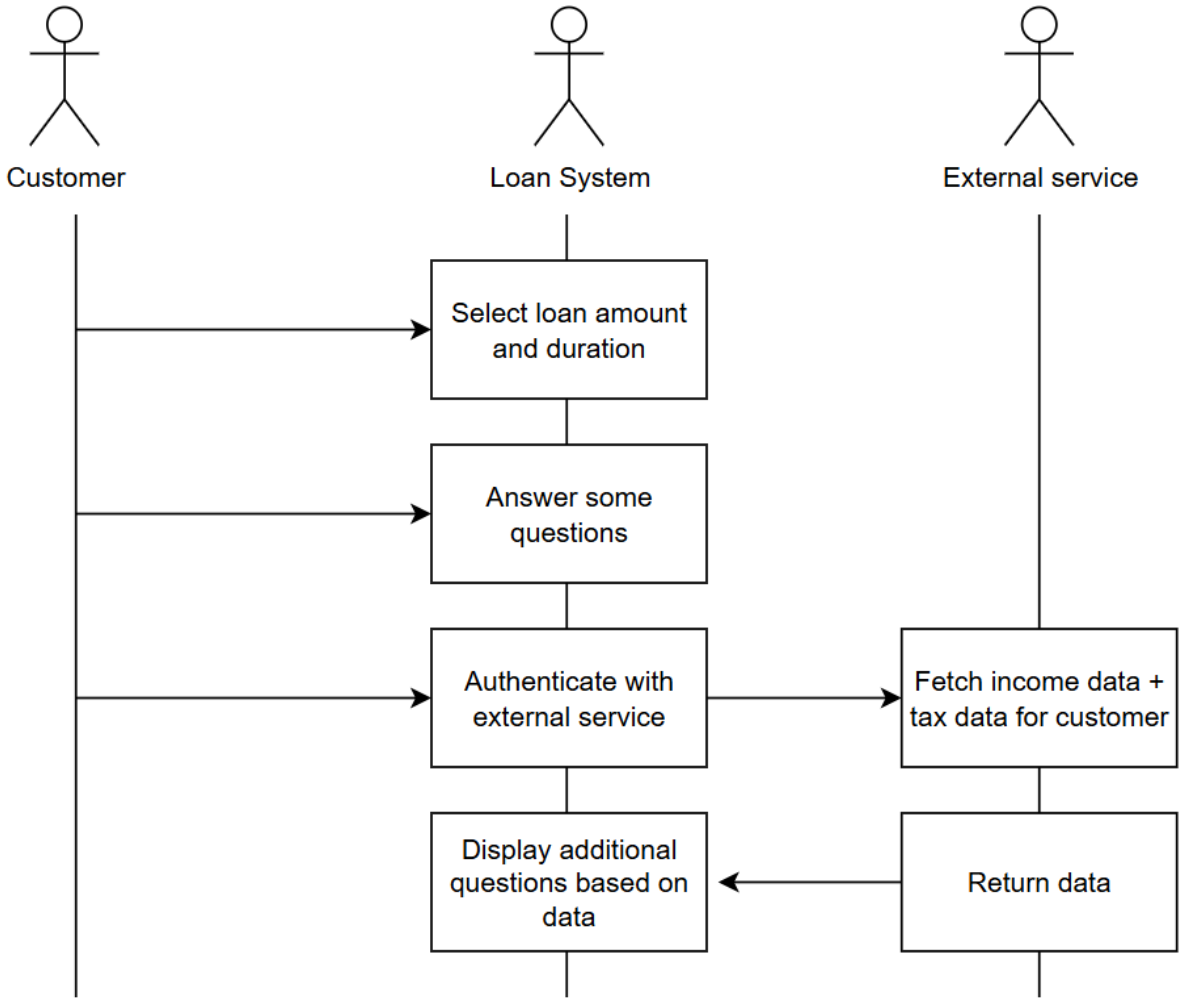
Challenges of test automation in agile SDLC

- Rapid changes in requirements (“responding to change over following a plan”)
- Maintain speed
  - Resource allocation
  - Initial investment
- Balancing costumer value and adequate test coverage
- Test become more complex as the feature grows
- Collaboration between teams

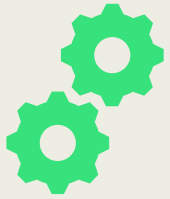


Challenges with test data

- What is synthetic test data?
  - Data that is artificially created and mimics real-world data
  - Synthetic test data are created for the purpose of testing in non-production environments
- Where can we fetch test data from?
  - Create them ourselves
  - Fetch them from other teams
  - Fetch them from external systems (Tenor testdata)
- Example: Applying for a loan



But, test automation can enable teams to work more agile



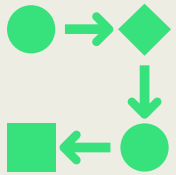
Continuous process



Maintain quality and speed with frequent releases



Be mindful about what is automated



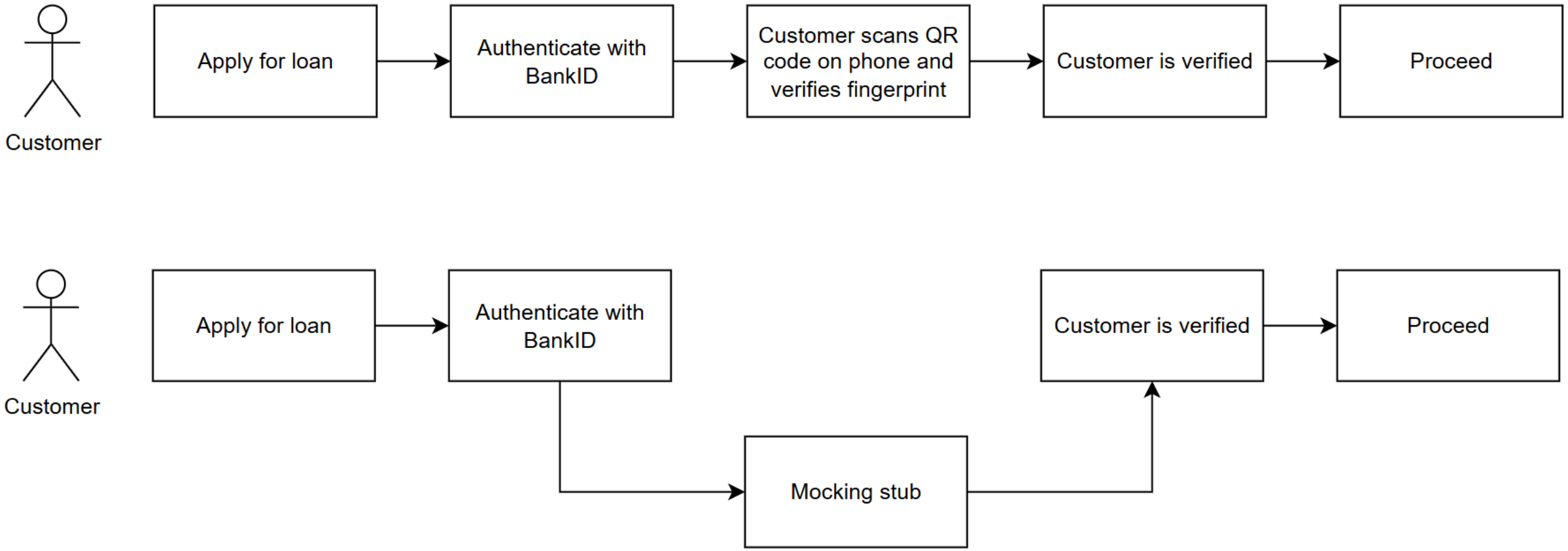
Minimize the initial investment and build in iterations

Who should work with test automation?

- Team setup will vary from project to project
- Design, implementation and maintainence is technical in nature
- A need for domain expertise and testing skills
- Everyone should participate! Cooperation between developers and test engineers



Cooperation between QA and Developer – example



Frameworks

How to choose your framework?

- No silver bullet
- Paid or open source?
- Does your framework have continued support?
- Is the community active?
- Consider using a test automation framework that was made for your chosen development framework
- Which framework do your team have experience in?



What language can I use?















Different frameworks





Protractor



Serenity BDD



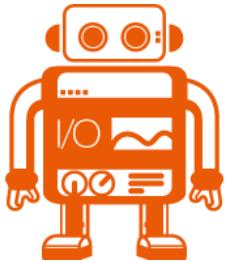
Cucumber



NightWatch



RobotFramework



WebDriveIO



RedWood



Playwright

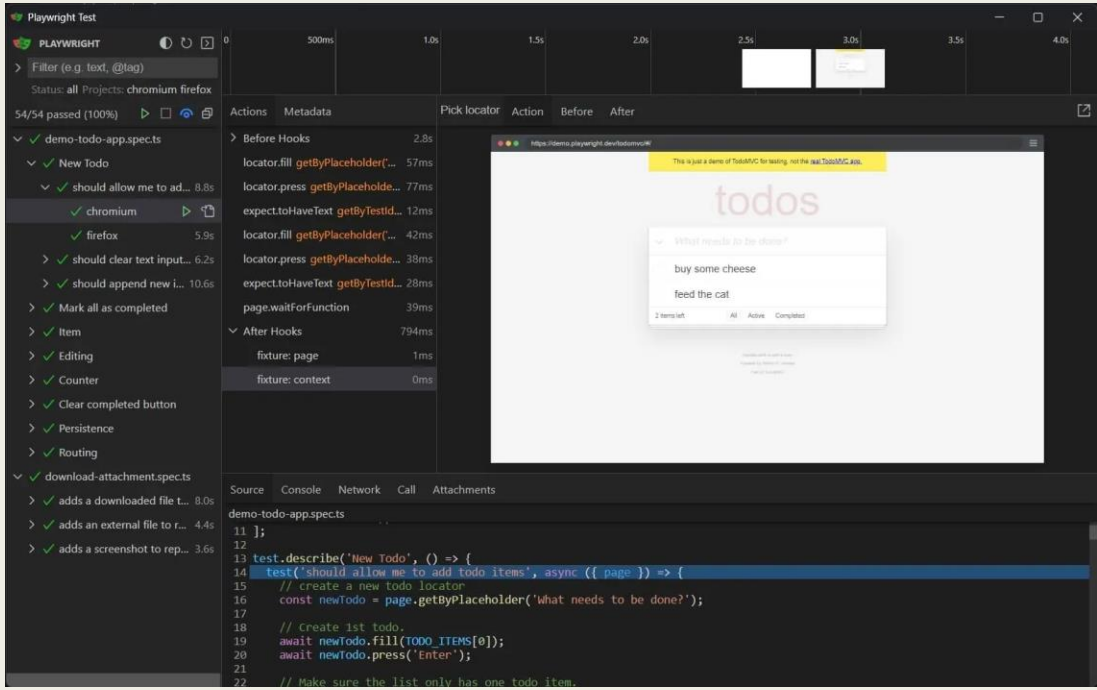
What is Playwright?

- Created by Microsoft in 2020
- JavaScript/TypeScript-based framework for end-to-end testing
- Front-end testing tool built for Web and API testing
- Free & Open-Source



Features

- Cross browser support
- Screenshots and Videos
- Multiple languages support
- Parallel execution
- Stubs and network manipulation
- Continuous Integration



Playwright vs Cypress

Playwright

- Focus on local and CI testing
- All-in-one package
- Iframes support
- Supports several superdomains within one test
- Open source

Cypress

- Focus on local testing
- All-in-one package
- No Iframes support
- Complex workaround for superdomains
- Time travel
- Open source / paid license


Let's get practical

Creating our very first test

In the ‘tests’ folder, create your own file that will include your first playwright test. The file **must end in .spec.ts** for Playwright to recognize it as a test-file. Copy the following line and paste it on the very top on your file:

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

In Playwright, we use `test.describe` to organize our test-suites which groups several test cases that test the same area / functionality. **It serves no purpose other than visual grouping**. For example:

```
test.describe('testing the kitchen', async() => {
    test('makes sure that the oven works', async({ page }) =>{

    })

    test('makes sure that stove top works', async({ page })  => {

    })

    test('makes sure the faucet works', async({ page })  => {
```

Material provided

- Playwright introduction
- Installation and setup guide
- Playwright cheat sheet
- Weekly exercises
- Mandatory exercise
- This presentation

- Playwright official documentation: https://playwright.dev/docs/intro

ITERA

Questions or curious? Feel free to contact us here:

Susanne.semsoy@itera.com

Mia.bjorndal@itera.com