ATTAFI YOUSRA

Certified Selenium Tester

Part 4
Preparing Maintainable Test Scripts

Knowledge levels of learning objectives

01

K1: remember

02

K2: understand

03

K3: apply

04

K4: analyze

Maintainability of test scripts

3

Maintainability of test scripts

- An automation tool does not have human intelligence. Scripts do not benefit from any intuition.
- This can be overcome by adding elements of **context** and of **plausibility** testing when programming automated scripts.
- Automation is a complex activity for two main reasons:
  - It must integrate intelligence into the scripts so that they best simulate a human tester.
  - It must manage an ever-increasing complexity of SUTs
- But the more complexity we add to our automation, the more predictable risks there are of automation failures.

Good practices

- **Managing web item selectors**
  - When testing a browser, be sure not to use absolute paths when locating web items. Relative paths tend to change less often.
- **Smart chaining**
  - The execution of scripts must be chained regardless of the result of executing a script whether it fails or succeeds.
  - If we can always run the next test, we can always run the whole suite, regardless of the number of failed tests during the execution.

Good practices

- **Code sharing**
  - Any code that automation needs to call repeatedly can be created as a function. You can create libraries of functions that can be included in your code to provide toolkits for the whole team.
- **Variable naming**
  - Define global names that make sense.
  - This makes the code more readable, which has the side effect of making the automation code easier to maintain.
  - It takes a few extra seconds to find suitable names, but it can save hours of work afterwards.

Good practices

- **Use of comments**
  - Comment. Lots of comments. Meaningful comments.
- **Smart logging**
  - Dwell-managed logs from the startwill make automation more scalable.
  - Accurate logs can reduce problem diagnosis time.
  - Consider the needs of your organization. If your SUT is a critical system, your logs must be complete enough to be audited.

Test Accounts and Fixtures

- **Automation needs to have certainty about the data used in automation.**

- It is essential to create user accounts and fixtures that are specifically intended for automation.

- These accounts and associated fixtures should not be tied to any particular person. Generic accounts that mimic real accounts are much better at isolating them from changes. Having enough different accounts ensures that tests do not interfere with data from other tests.

File naming

- **The files**
  - Save files in well-organized folders. Consider creating a folder with a timestamp in the name and placing all the files for a run there. If multiple machines are running the automation, or if you are testing in different environments, you can add the workstation name or environment names to the folder names.
  - Place them in directories that can be destroyed without damage.

ONE OF THE IMPORTANT PRINCIPLES OF SETTING UP A MAINTAINABLE TEST AUTOMATION ARCHITECTURE IS TO **DIVIDE IT INTO LAYERS**.

WHEREVER POSSIBLE, SCRIPTING COMPLEXITY SHOULD BE MOVED TO THE AUTOMATION ARCHITECTURE/AUTOMATION FRAMEWORK LEVEL.

WE WILL DISCUSS THIS SUBJECT A LITTLE MORE IN DETAIL IN THIS SECTION THROUGH THE FUNCTIONS **WRAPPER** AND DESIGN PATTERNS **OBJECT PAGE**.

Wrappers Objects and Functions Page

11

The test abstraction

One of the important principles of setting up a maintainable test automation architecture is to divide it into layers.

One of the layers offered by the ISTQB is the **Abstraction layer of tests**; this layer provides the interface between the **test case business logic** and the **concrete needs for steering the SUT.**

Page Objects and Wrapper functions are part of this layer by abstracting the GUI from the SUT.

Principle of Wrapper functions

- Intelligence is put into **aggregate functions** rather than in each script individually.

- By moving the intelligence of the test outside of the scripts themselves, the code becomes easier to maintain and more scalable.

- If the code fails, the correction can be achieved in a single corrective action.

Using Wrappers

- An interaction with a web element by Selenium can be designed in these few steps:
  - Make sure the element on the interface is ready to use
  - Temporize if necessary
  - If the element is still not usable, log the error and close the script execution
  - Manipulate element
  - Verify that the control behaves as expected after handling it.
  - In case of problem, write a useful log message

Objects page

Principle of the Page Objects design pattern

- A Page Object represents an area of the web application interface that your test will interact with. There are several reasons for using this mechanism:
  - It creates reusable code
  - It reduces the amount of duplicate code.
  - It encapsulates all operations on the SUT GUI in a single layer.
  - It clearly separates the business and technical parts of test automation design.
  - It reduces code maintenance costs and efforts
  - It gives a single entry point in the code to correct all the scripts concerned in case of change

Object page: definition

- Page Object designates a **class**, a **module** or **a set of functions** which contains **interface** to a form, page, or page fragment of the SUT that you want to control.
- The Page Object Pattern is used to manage business actions that are used as test steps in the **test execution layer ,** they are therefore part of the **test adaptation layer**.
- Page Object Pattern is thus an application of the approach called Keyword Driven Testing, which allows an automation project to reduce maintenance efforts.

Design rules for a Page Object

- Page Objects must not contain assertions about the **business logic** or checkpoints.
- All assertions and verifications **techniques** concerning the graphical interface must be carried out in the Page Objects.
- All wait mechanisms should be encapsulated in Page Objects.
- A Page Object must contain calls to Selenium functions, and only it.

Page Objects

- A Page Object does not need to span the entire page or form. He can control a section or another specific part of it.

- In a test automation architecture, Page Objects encapsulate calls to Selenium WebDriver methods, so that **the test execution layer deals only with the business level**.

- There is no import or use of the Selenium library in the test execution layer.

Keyword-driven testing (Keyword Driven Testing)

Keyword-driven testing(KDT)

- A natural extension of the Page Object Pattern
- Keywords are abstract, business-oriented terms that functionally describe a task that the SUT must perform
- Focuses on the question "**what do I te**st" rather than "**how do I test it**"
- The KDT emulates the way a hand tester works
- Test analysts define keywords based on what they need
- Automation engineers create the functionality behind the keyword and the framework to run the test

ISTQB Definition

A scripting technique that uses data files to contain both test data and expected results, but also **keywords**.

Keywords are business actions or steps in a test case. These are exactly the same as those in the first column of a manual test case.

These keywords are interpreted by **supporting scripts** specific that are called by the **control script** of the test.

Keyword-driven testing

- The keyword (KeyWord, ActionWord) is an abstract business action or a step in an abstract test case.
- Like a manual test, a KDT script is generally summarized in three columns
  - The task to be performed (the keyword)
  - The data(s) to be taken into account
  - The expected result
- A KDT scriptdoes not contain specific low-level actions to perform on the SUT.
- The exact concrete actions on the SUT are hidden in the implementation of the keywords.

Keyword-driven testing: benefits

- The design of the test case is decoupled from the implementation of the SUT.
- The test execution, test abstraction and test definition layers are clearly separated.
- An almost perfect division of labour:
  - Test analysts design test cases and write scripts using keywords, data, and expected results.
  - Technical test analysts implement the keywords and execution framework needed to run the tests.
- Keywords are reusable in different test cases.
- Test cases are more readable

Keyword-driven testing: benefits

- Less redundancy.
- Reduced maintenance costs and efforts.
- A manual tester can run the test manually directly from the automated script.
- Scripts using keywords can be written long before the SUT is available for testing
- Automation can be ready earlier and used for functional testing and not just regression testing.
- A small number of test automation engineers can work with an unlimited number of test analysts, which facilitates the extension of automation.
- Because scripts are separate from the implementation level, different tools can be used interchangeably.

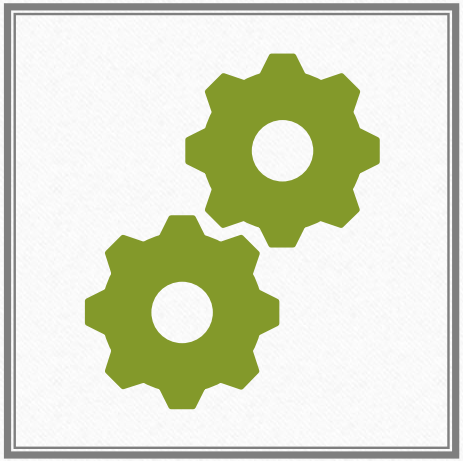
Things to consider

- More precise keywords allow for more specific scenarios, but at the expense of script maintenance complexity
- Allowing access to low-level actions increases flexibility but, when tied to the GUI, test maintenance increases
- Aggregate keywords can simplify initial development but complicate maintenance
- Initial keyword design is important, but eventually new and different keywords will be needed that involve business logic and automation functionality for execution

Implementation of the KDT

- **Top-down** approach
  - Write test case steps like you would write manual tests, then implement them in your tool
  - These steps can be written by test analysts
  - More profitable when many keywords have already been implemented or when test scripts are updated
- **Bottom up** approach
  - Record a script, then re-adapt it to meet the KDT architecture
  - Usually used at the start of test automation

KDT Tools



- Cucumber
- Robot Framework
- Katalon Studio
- Harmony
- HPE Unified Functional Testing

•Keyword-driven architecture can be implemented in many programming languages

Acronyms

- API: Application Programming Interface
- CERN: European Council for Nuclear Research
- CI: Continuous Integration
- CSS: Cascading Style Sheets
- DOM: Document Object Model
- GUI: Graphical User Interface
- HTTP: Hyper Text Transfer Protocol
- ISTQB: International Software Testing Qualifications Board
- KDT: Keyword Driven Testing

Acronyms

- REST: Representational State Transfer
- ROI: Return on Investment
- SDLC: Software Development Life Cycle
- SOAP: Simple Object Access Protocol
- SUT: System/Software Under Test
- TAA: Test Automation Architecture
- TAE: Test Automation Engineer
- TAS: Test Automation Solution
- TCP: Transmission Control Protocol
- UI: User Interface
- W3C: World Wide Web Consortium

Glossary

- **class attribute**: HTML attribute that points to a class in a CSS style sheet. It can also be used by a JavaScript to make changes to HTML elements with a specified class
- **comparator**: A tool to automate the comparison of expected results against actual results
- **css selector**: Selectors are templates that target the HTML elements you want to style
- **Document Object Model (DOM)**: Application programming interface that treats an HTML or XML document as a tree structure in which each node is an object representing a part of the document

Glossary

- **fixture**: a test fixture is a fictional object or environment used to constantly test an item, device or piece of software
- **framework**: Provides an environment to run automated test scripts; including tools, libraries and fixtures
- **function**: A Python function is a group of reusable statements that perform a specific task.
- **hook**: An interface that is introduced into a system that is created primarily to provide better testability to that system
- **HTML (HyperText Markup Language)**: The standard markup language for creating web pages and web applications

Glossary

- **ID**: Attribute that specifies a unique identification string for an HTML element. The value must be unique within the HTML document
- **iframe**: An HTML inline frame, used to embed another document in an HTML document
- **modal dialog**: A screen or box that forces the user to interact with it before they can access the underlying screen
- **Object Pattern page**: A test automation model that requires technical logic and business logic to be processed at different levels
- **pesticide paradox**: A phenomenon in which repeating the same test several times leads to finding fewer defects

Glossary

- **persona**: A user profile created to represent a user type that interacts with the system in a standard way
- **pytest**: A Python testing framework
- **tags**: HTML elements are delimited by tags, written using brackets <>
- **technical debt**: Involves the additional cost of redesign caused by choosing to ignore bad designs or short-lived implementations
- **WebDriver**: The interface in which Selenium tests are written. Different browsers can be controlled through different Java classes, for example, ChromeDriver, FirefoxDriver, etc.

Glossary

- **wrapper**: A function in a software library whose primary purpose is to call another function, often adding or enhancing functionality while hiding complexity
- **XML (eXtensible Markup Language)**: A markup language that defines a set of rules for encoding documents in both human and machine readable format
- **XPath (XML Path Language)**: A query language to select nodes from an XML document

Thank You

ATTAFI YOUSRA
EXPERT EN TEST LOGICIEL
@ : ATTAFI.YOSRA@HOTMAIL.FR
LINKEDIN : Yousra ATTAFI