A Lightweight MBT approach for Visual Acceptance-Test Driven Development - Experience Report

Elodie BERNARD*†, Fabrice AMBERT*, Bruno LEGEARD*‡
**FEMTO-ST Institute, Univ. Bourgogne Franche-Comté, CNRS* Besançon, France
† *Sogeti*, Lyon, France
‡ *Smartesting*, Besançon, France
[elodie.bernard|fabrice.ambert|bruno.legeard]@femto-st.fr

***Abstract*—In this presentation, we give the results of an experiment using a lightweight MBT approach to implement Acceptance Test driven Development in a large Agile IT project.**

***Index Terms*—Model-based-testing, Lightweight MBT, Agile Software Development Lifecycle, ATDD**

I. Introduction

The practice of Acceptance Test Driven Development (ATDD) was born as an extension of Test Driven Development [1] [2] [3]. In ATDD, acceptance tests are produced collaboratively during requirements analysis by business analysts, product owners, testers and developers. To write readable test cases that support test automation, ATDD often use a domain-specific scripting language, called the Gherkin language [4], in a Given-When-Then format. Gherkin's language test cases written in Gherkin language are often quite atomic, covering one aspect of a User Story for example. [4]. To define more global acceptance tests, for example end-to-end test cases, an approach using a graphical representation of workflows and a lightweight MBT approach [5] can be appropriate. This is what we present in this experience report of using Lightweight MBT to implement a visual ATDD approach in the context of a large-scale IT project.

II. Experimentation and lesson learned

The research questions we aimed to verify were as follows:

- RQ1: To what extent is the Lightweight MBT approach adapted to short agile iterations?
- RQ2: To what extent is the Lightweight MBT approach efficient at creating and maintaining automated scripts?

We will present our experiments on two projects with two distinct contexts with our Lightweight MBT tool: Yest. Now we will present the two contexts of the projects on which we conducted our experiments

a) Context 1: The first project, which we will call Project A, dealt with the testing of a web application on a large quality management application in a French railway company. It was developed in an Agile context with 4-week sprints and a test approach to verify the features implemented at the end of each sprint. Test automation started 1.5 years after the beginning of the application development and used the Lightweight MBT approach (with the Yest tool) and the Selenium framework. The test cases to be automated were built by the test automation specialist via Yest then he completed the adaptation layer and built the automation code. Only test cases for automation were built with Yest, existing test cases were created in a standard test management tool (here Squash) for 1,5 years, manual test cases had been developed without any information about a future test automation plan.

b) Context 2: The 2nd project, which we will call Project B, dealt with the testing of a web application to manage employees skillset of a French railway company. It was developed in an Agile context with 4-week sprints and an ATDD approach. As early as sprint 1, the test teams were made aware of automation and the tests were built according to a set of methodologies and best practices. Automation was initiated after 6 months of the project (as soon as stable processes were identified) using an MBT approach (with the Yest tool) and the Selenium framework. The tests to be automated were built by the test team via Yest and then an automation engineer completed the adaptation layer in Yest and built the automation code. The objective of these two approaches was to evaluate how the use of Lightweight MBT could facilitate test design in an agile context and the implementation of test automation on a large IT project. The metrics collected to evaluate the efficiency of the approach were the time dedicated to test design and implementation [6], the number of test cases produced and the number of steps per test, as well as the number of US processed.

The following section will discuss the study of manual test design and implementation compared to an approach with Lightweight MBT.

A. Manual test design and implementation vs. Lightweight MBT approach

In order to compare the two approaches we confronted the test design and implementation between projects A and B over a 3-month period. For both projects, the basis for designing the tests was the US. These US had exactly the same model for both projects: i.e. a presentation of the context, followed by a description of a set of management rules. These rules must be processed in test cases.

In Project B we ensured that the Lightweight MBT approach was applied throughout the project. As part of the experiment we analyzed the US and defined the scope that would be covered during the next sprint. We therefore had visibility into the US that were new and the features that would need to be updated. So we started each sprint by quickly updating the highest priority test cases to be run.

We will now present the results obtained following these test design and implementation phases on each of the projects. For a 4-week sprint, the time dedicated to design was about 1740 minutes for Project A and 1770 for Project B (an average of four days). We will use hours and minutes as the unit of measurement, as we will report design times by test case and US. Team A processed an average of 13 US per sprint compared to 32 US for team B. Now let's take a more detailed look, i.e. at the number of test cases produced and taking into consideration the number of steps per test case. Team A produces an average of 102 test cases per sprint with an average of 10 steps per test case, and team B produces an average of 131 test cases with an average of 15 steps per test case. If we take into account the number of steps per test case, i. e. an average of 1020 steps for project A and 1965 steps for project B, we arrive at a conception time of 1 min 42 s per step for team A against 54 s for team B, i. e. a decrease by almost 50%. (47%)

For rather similar project perimeters, with a similar test case design and implementation source, the results of our experiments indicate that test design with Lightweight MBT approach is 47% more efficient than a traditional manual design approach. (shown in the figure 1)

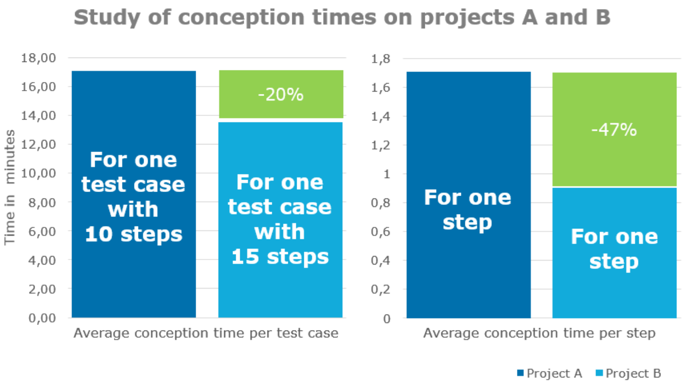


Fig. 1. Summary of results on the study of design times between manual design and implementation vs. Lightweight MBT

In order to explain these results, here is a summary of the points that made it possible to be more efficient with a Lightweight MBT approach. To begin the manual design of tests is traditionally done in a test management and execution tool such as hp ALM, Squash TM or Excel. (Here Squash TM is used) In these tools, in case of a change affecting a set of test cases it is necessary to take each of these test cases individually. With a Lightweight MBT tool it is easy to identify the model impacted by a change, update the model as required and then generate all the updated test cases, without having to modify them individually. This generation of test cases saves a lot of time, and this is possible through the easy use of the tool, hence the Lightweight aspect which allows easy and fast updates in a workflow and then regenerating test cases. This tendency is confirmed by the study of update times. In the context of project B, time spent updating test cases was distinguished from time spent creating new test cases. On the design study of 268 test cases with 110 new test cases and 156 in update, we observed an average gain of 60% on the update time of test cases, so it is twice faster than the design phase. Compared to a manual approach, design with Lightweight MBT gives good results and is more efficient for test case design and implementation and very efficient for updating test cases.

B. *Automation*

As part of our evaluation of the Lightweight MBT, we sought to evaluate how effective this approach was in creating and maintaining automated test cases. To this end, we conducted two experiments, one on project A and the other on project B, which we presented in the previous section. On project A no tests were produced by the test team with the MBT tool, it was the test automation engineer who created the test workflows and generated the test cases and then the scripts. The objective was to reproduce in the MBT tool the test cases already existing on the project. The choice of this approach was guided by the desire to compare two approaches: build test cases in a Lightweight MBT tool with visibility on automation (project B) and take existing test cases and adapt them in a Lightweight MBT tool, but without considering future automation (project A). Thus on project B, the test cases were already existing and the automation engineer only completed the adaptation layer to produce the scripts. Still with a view to experimenting with the Lightweight MBT approach, on project B, we completed the adaptation layer during the sprint on a 1st perimeter and then on another perimeter during a subsequent sprint. Including maintenance phases to update Keywords. These phases are perfectly integrated during the sprint. A total of 26 days were dedicated to automation on project A compared to 8 days on project B. The objective on project A was to automate 5 test cases, representing a total of 48 steps to automate and on project B 12 test cases composed of 58 steps to automate, i.e. a fairly similar scope to automate (20% step more on project B) in terms of number of steps to work on, therefore comparable.

For project A, 6 days were dedicated to the modeling of the workflow and 2 days to the completion of the adaptation layer, i.e. 30% of the total effort for automation against 1.5 days for project B, i.e. 18.75% of the total effort for automation. If we compare these two global percentages 30% and 18.75% we obtain a difference of 40%: this is explained by the fact that the workflows and test cases did not exist on the project A which required a lot of work by the automation engineer to identify the existing test cases and introduce them in Yest. The 1st results show that when the test cases are already present the effort to complete the adaptation layer in Yest is reduced. After the test design phase and the completion of the adaptation layer in Yest, we studied the design phase of Keywords. Their coding accounted for 42% of the time dedicated to automation on project A compared to 62% on project B, a difference of 20%. If we compare the coding time between the two projects (11 days on project A, 5 on project B) we observe a difference of 55%. The writing of Keywords was twice as fast on project B as on project A. This can be explained by the fact that on project B the testers who designed the tests had conceived the test workflow with a view to automation, ensuring that there were uniform and consistent test cases that could be easily adapted to a Keywords system. On the 1st project, the test cases had been developed without any real reflection on the automation phase that would follow.

An example of good practice applied is to limit the number of steps (and their content) in test cases to be automated, this facilitates the completion of the adaptation layer. This limits the writing of keywords and their sequencing. As a test case becomes longer, it can be assumed that it will be more complex and therefore involve a certain level of complexity to automate. If we review the experiments and compare the length of the test cases, i.e. the number of steps they contain between projects A and B, we can see that the distribution of the number of steps in the test cases was rather different. Indeed, on project A on 10 test cases to be automated, only one test case had less than 10 steps (10 steps exactly) that is 10% of the referential. On project B, on 13 test cases, 6 test cases had less than 10 steps, almost half of the referential (46%). A majority of short tests therefore contribute to facilitate and save time in the completion of the adaptation layer.

This was to assess how the use of good practice on the Lightweight MBT reduced the design effort of Keywords. The application of good practices facilitates automation even if the writing of Keywords remains a major part of the automation phase. Script generation on both projects is automatic in the Lightweight MBT tool: once the adaptation layer is completed and the Keywords implemented, it is possible to generate the scripts and integrate them into an automated test execution tool. We did not discuss this part in our study because it was independent of the use of Yest. Finally, we studied the maintenance phase: we measured the time dedicated to maintenance on each of these projects, 7 days on project A, 1.5 days on project B. The perimeter of Keywords was similar, but there was more complexity in maintaining the Keywords of project A than on project B. This is the direct consequence that on project A, the tests were not completely adapted to be automated so a certain gymnastics had to be set up to create adapted Keywords and therefore led quite naturally to maintenance difficulties, because the Keywords were complex. On the B project the Keywords were simpler, configurable and reusable on different test steps, thus limiting the updates to be performed to maintain the scripts. This explains why Project B has a maintenance time 78% shorter than Project A and represents 18.75% of the overall effort for automation compared to 27% for Project A.

III. Conclusion

Our approach sought to deal with two challenges : the optimization of test case design and test automation in Agile contexts. These results and our experience allow us to affirm that the tst case designe and implementation with Lightweight MBT is very well adapted to Agile cycles with short iterations, because this approach allows us to quickly generate new test cases, update it and thus allow the earliest possible test execution. On the B project, applying the Lightweight MBT approach, nearly 700 test cases were generated via Lightweight MBT in Agile and ATDD context over 8 months of project without the test design time delaying the different phases of the cycle, on the contrary.

With regard to test automation, an early account that test cases will be automated as part of the project is a key success factor and a major facilitator of automated test development and maintenance in the context of large IT projects. On the project A automation was initiated after more than a year of development without automation being planned and the subject was treated without implementation of good practices both in test design and keywords. As a result, the automation framework was too complex and its maintenance was very difficult. On project B, from the beginning of the project, the tests were designed in a uniform way, trying to use existing elements as often as possible instead of creating new ones in order to have the most ”coherent” test cases as we can.

References

[1] D. North, “Behavior modification: The evolution of behavior-driven development,” *Better Software*, vol. 2006-03, march 2006.

[2] S. Hammond and D. Umphress, “Test driven development: the state of the practice,” in *Proceedings of the 50th Annual Southeast Regional Conference*. ACM, 2012, pp. 158–163.

[3] K. Pugh, *Lean-agile acceptance test-driven development: better software through collaboration*. Pearson Education, 2010.

[4] E. C. dos Santos and P. Vilain, “Automated acceptance tests as software requirements: An experiment to compare the applicability of fit tables and gherkin language,” in *Agile Processes in Software Engineering and Extreme Programming*, J. Garbajosa, X. Wang, and A. Aguiar, Eds. Cham: Springer International Publishing, 2018, pp. 104–119.

[5] E. Bernard, F. Ambert, B. Legeard, and A. Bouzy, “Lightweight Model-Based Testing for Enterprise IT,” in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2018, pp. 224–230.

[6] “ISTQB Certified Tester Foundation Level Syllabus, 2018 Version.” [Online]. Available: https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html

A Lightweight MBT approach for Visual Acceptance-Test Driven Development

Experience Report

Elodie Bernard, Fabrice Ambert, Bruno Legeard

QRS 2019

The 19th IEEE International Conference on
Software Quality, Reliability, and Security
July 22-26, 2019 • Sofia, Bulgaria

Summary

- Introduction
- Context presentation
- Manual test design and implementation vs. Lightweight MBT approach
- Automation
- Conclusion

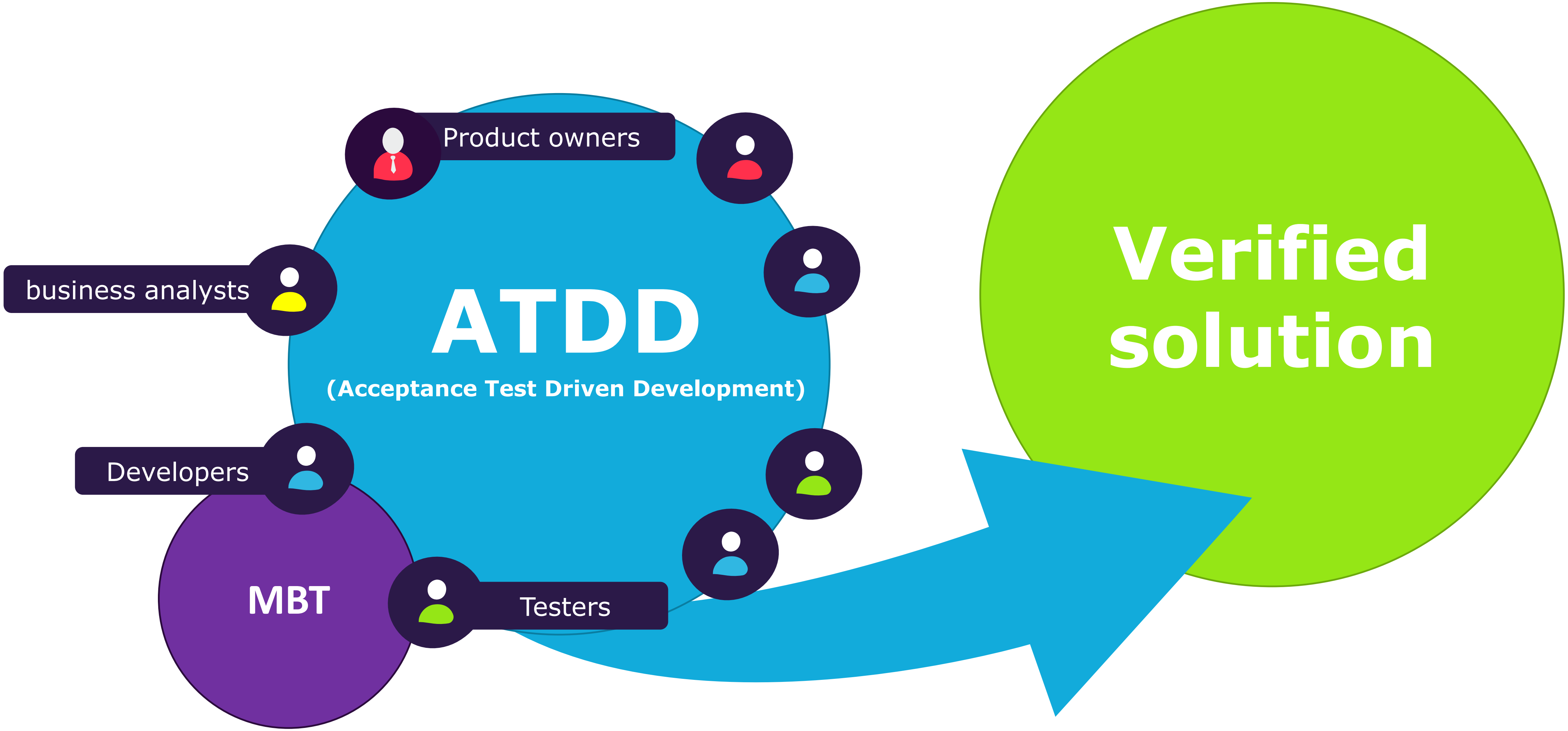
ATDD

Experience Report

Lightweight

MBT

Introduction

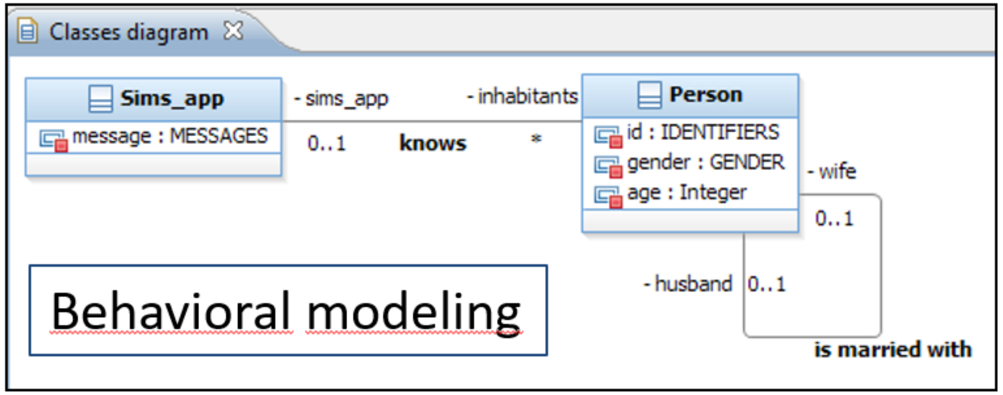


Context presentation

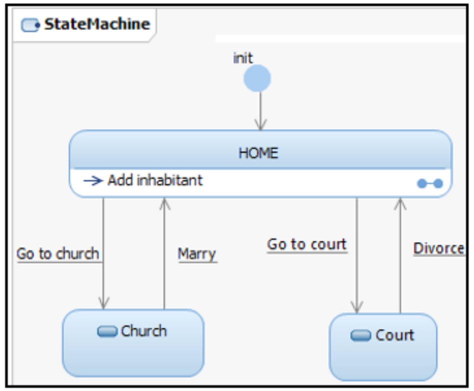
Project A	**Project B**
• **Testing of a web application** • **Agile context with 4-week sprints**	• **Using the Lightweight MBT approach (with Yest)** • **Using Selenium and Junit**
Only test cases for automation have been created with Yest	**All test cases** have been created with Yest
Do not use ATDD approach	**Use ATDD approach**
Test analyst : built test (only for manual execution)	**Test analyst :** built test (including those that will be automated)
Test automation specialist : • built test to automate (Based on the current one) • completed the adaptation layer • built the automation code.	**Test automation specialist :** • completed the adaptation layer • built the automation code.
Initialization an implementation of automation : after one and a half years	• **Initialization of automation :** from the beginning of the project • **Implementation of automation :** after 6 month (as soon as stable processes were identified)

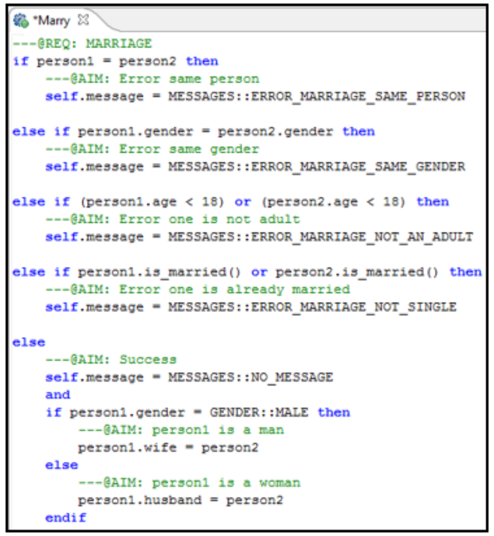
Manual test design and implementation vs. Lightweight MBT approach

Manual test design vs. Lightweight MBT approach
Overview

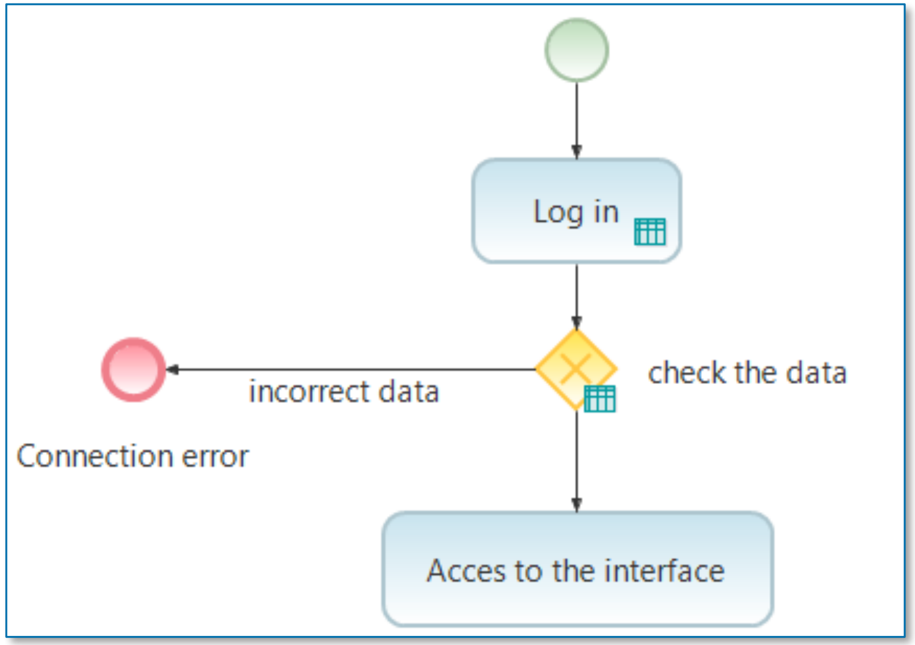


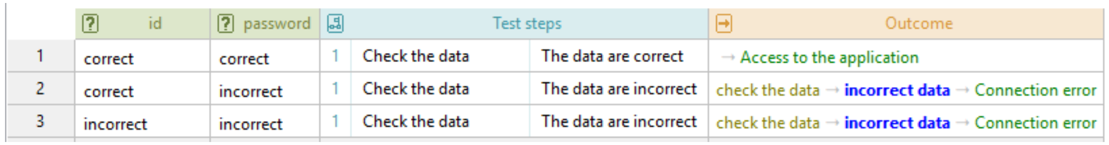
Behavioral modeling





```
---@REQ: MARRIAGE
if person1 = person2 then
    ---@AIM: Error same person
    self.message = MESSAGES::ERROR_MARRIAGE_SAME_PERSON

else if person1.gender = person2.gender then
    ---@AIM: Error same gender
    self.message = MESSAGES::ERROR_MARRIAGE_SAME_GENDER

else if (person1.age < 18) or (person2.age < 18) then
    ---@AIM: Error one is not adult
    self.message = MESSAGES::ERROR_MARRIAGE_NOT_AN_ADULT

else if person1.is_married() or person2.is_married() then
    ---@AIM: Error one is already married
    self.message = MESSAGES::ERROR_MARRIAGE_NOT_SINGLE

else
    ---@AIM: Success
    self.message = MESSAGES::NO_MESSAGE
    and
    if person1.gender = GENDER::MALE then
        ---@AIM: person1 is a man
        person1.wife = person2
    else
        ---@AIM: person1 is a woman
        person1.husband = person2
    endif
```

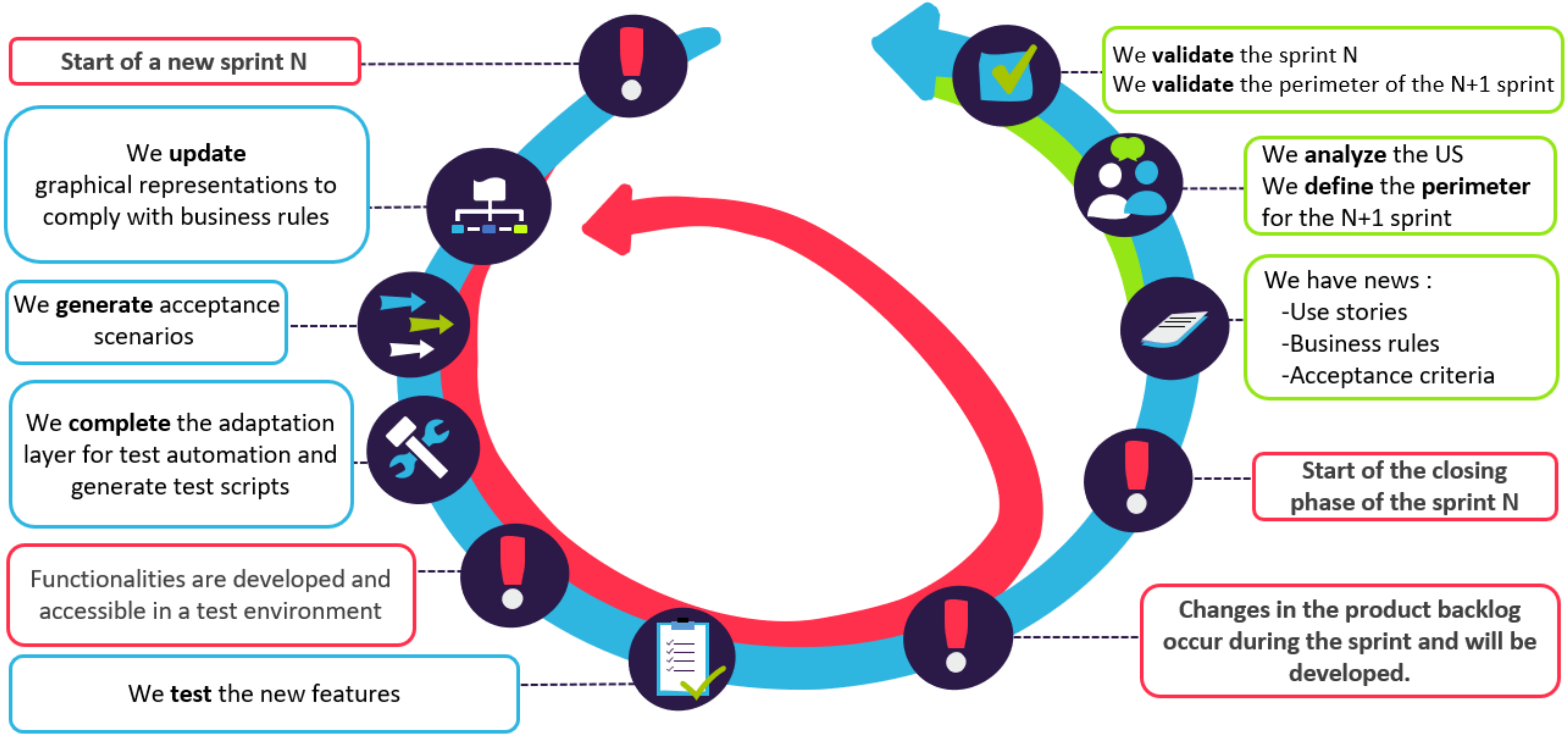




	id	password	Test steps			Outcome
1	correct	correct	1	Check the data	The data are correct	→ Access to the application
2	correct	incorrect	1	Check the data	The data are incorrect	check the data → incorrect data → Connection error
3	incorrect	incorrect	1	Check the data	The data are incorrect	check the data → incorrect data → Connection error

Manual test design vs. Lightweight MBT approach

Lightweight MBT approach



Start of a new sprint N

We **update** graphical representations to comply with business rules

We **generate** acceptance scenarios

We **complete** the adaptation layer for test automation and generate test scripts

Functionalities are developed and accessible in a test environment

We **test** the new features

We **validate** the sprint N
We **validate** the perimeter of the N+1 sprint

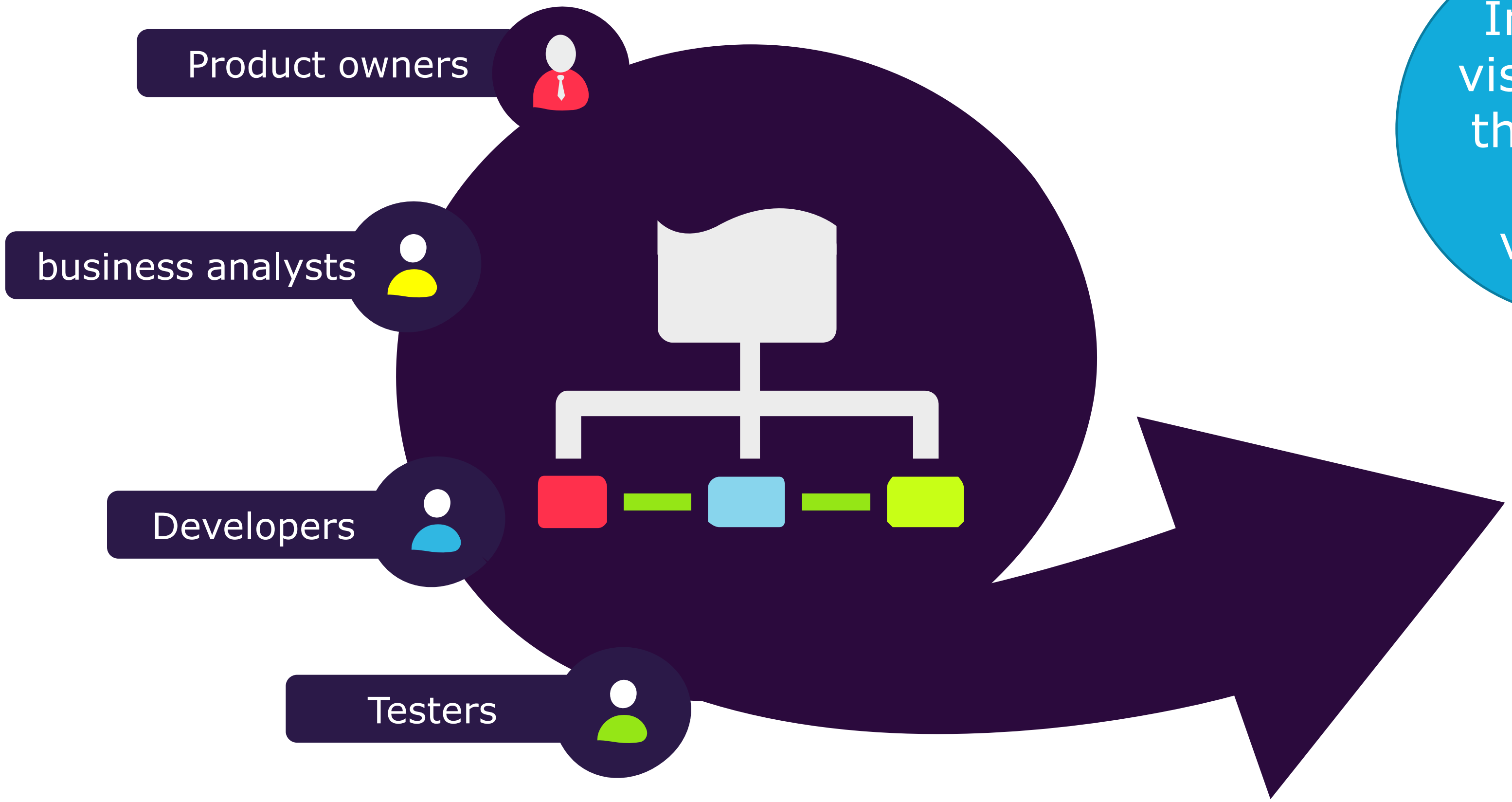
We **analyze** the US
We **define** the **perimeter** for the N+1 sprint

We have news :
- Use stories
- Business rules
- Acceptance criteria

Start of the closing phase of the sprint N

Changes in the product backlog occur during the sprint and will be developed.

Manual test design vs. Lightweight MBT approach
Lightweight MBT approach



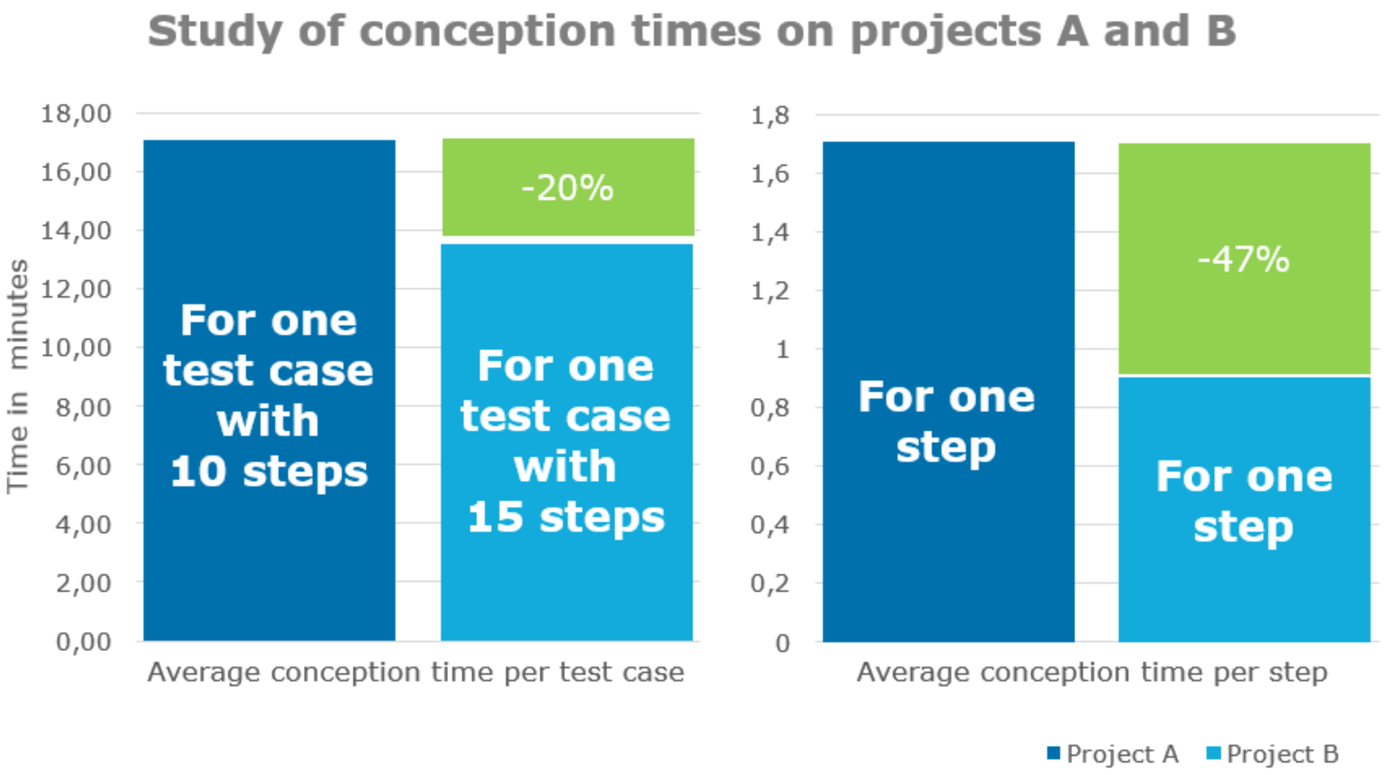
Increase visibility of the scope of the version

Clarify the business needs

Improve the validation of the US

Make testing easier

Manual test design vs. Lightweight MBT approach
Results



On project B
an average gain of 60% on the update time of test cases

Manual test design vs. Lightweight MBT approach
Conclusion

- Design and update time are almost halved
- Communication between stakeholders is facilitated
- The expression and dissemination of customer needs is improved
- Overall test management is straightforward

Automation

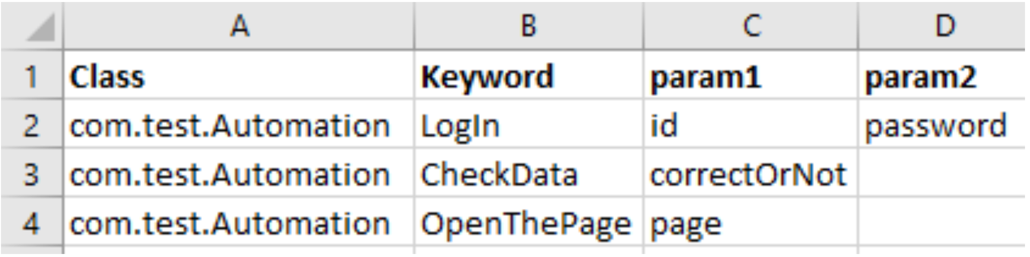
Context presentation (Reminder)

Project A	**Project B**
• **Testing of a web application** • **Agile context with 4-week sprints**	• **Using the Lightweight MBT approach (with Yest)** • **Using Selenium and Junit**
Only test cases for automation have been created with Yest	**All test cases** have been created with Yest
Do not use ATDD approach	**Use ATDD approach**
Test analyst : built test (only for manual execution)	**Test analyst :** built test (including those that will be automated)
Test automation specialist : • built test to automate (Based on the current one) • completed the adaptation layer • built the automation code.	**Test automation specialist :** • completed the adaptation layer • built the automation code.
Initialization an implementation of automation : after one and a half years	• **Initialization of automation :** from the beginning of the project • **Implementation of automation :** after 6 month (as soon as stable processes were identified)

Lightweight MBT for automation
Overview

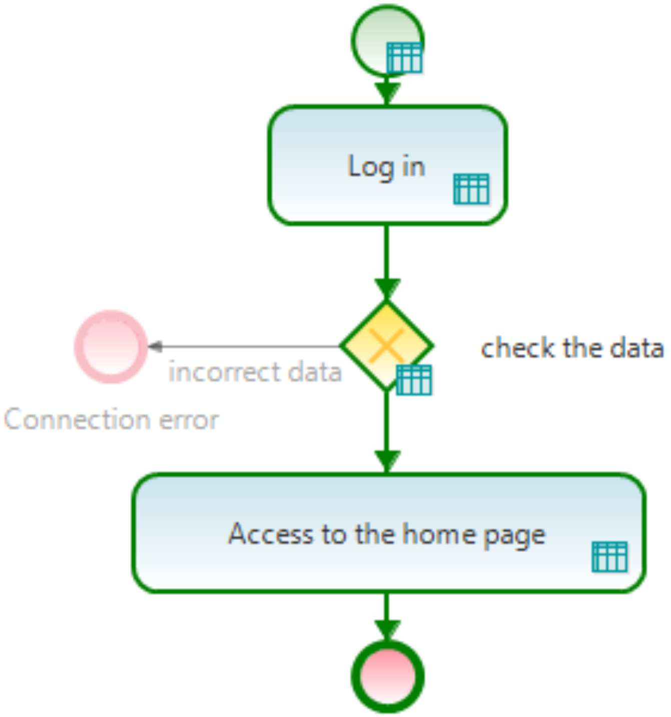
- Keyword-driven-testing
- Java Selenium add-on
- Data set management

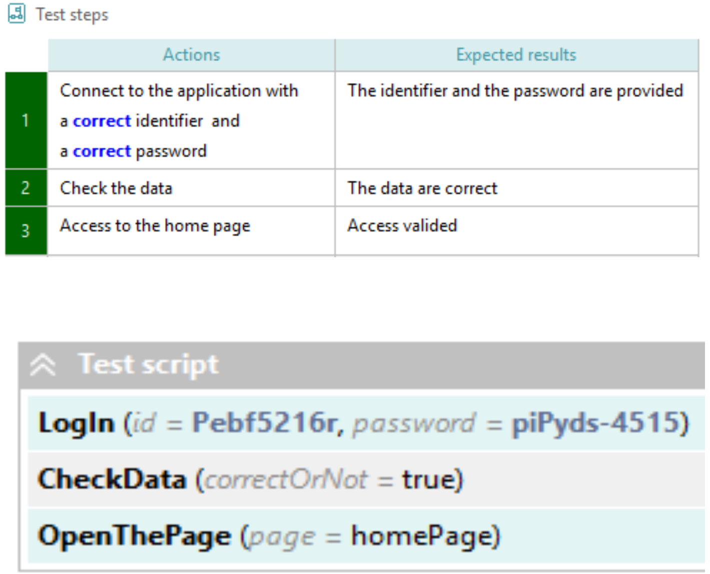
Keywords table with Yest



	A	B	C	D
1	**Class**	**Keyword**	**param1**	**param2**
2	com.test.Automation	LogIn	id	password
3	com.test.Automation	CheckData	correctOrNot	
4	com.test.Automation	OpenThePage	page	

Lightweight MBT for automation
Test automation process





Test steps

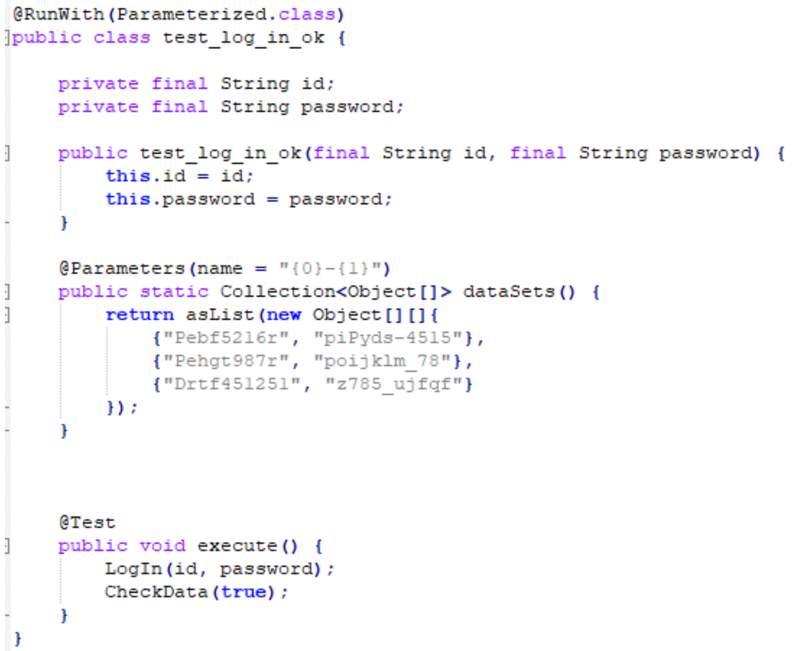
	Actions	Expected results
1	Connect to the application with a **correct** identifier and a **correct** password	The identifier and the password are provided
2	Check the data	The data are correct
3	Access to the home page	Access valided

Test script

LogIn (*id* = **Pebf5216r**, *password* = **piPyds-4515**)

CheckData (*correctOrNot* = true)

OpenThePage (*page* = homePage)



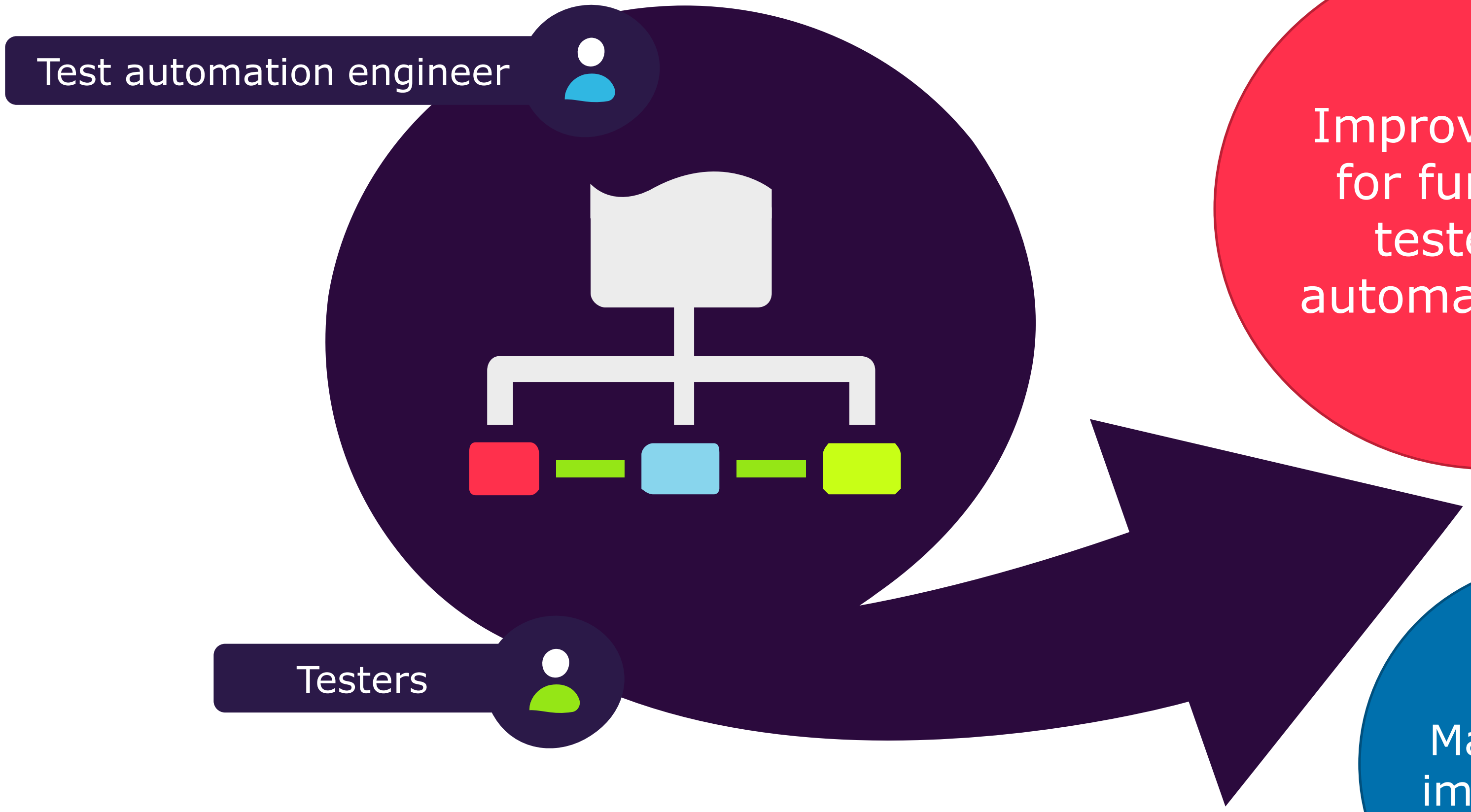
```java
@RunWith(Parameterized.class)
public class test_log_in_ok {

    private final String id;
    private final String password;

    public test_log_in_ok(final String id, final String password) {
        this.id = id;
        this.password = password;
    }

    @Parameters(name = "{0}-{1}")
    public static Collection<Object[]> dataSets() {
        return asList(new Object[][]{
            {"Pebf5216r", "piPyds-4515"},
            {"Pehgt987r", "poijklm_78"},
            {"Drtf451251", "z785_ujfqf"}
        });
    }



    @Test
    public void execute() {
        LogIn(id, password);
        CheckData(true);
    }
}
```

The visual representation of the test

The abstract scenario and the corresponding automated test script

The test script in java/Selenium with the use of dataset

Lightweight MBT for automation
Test automation process

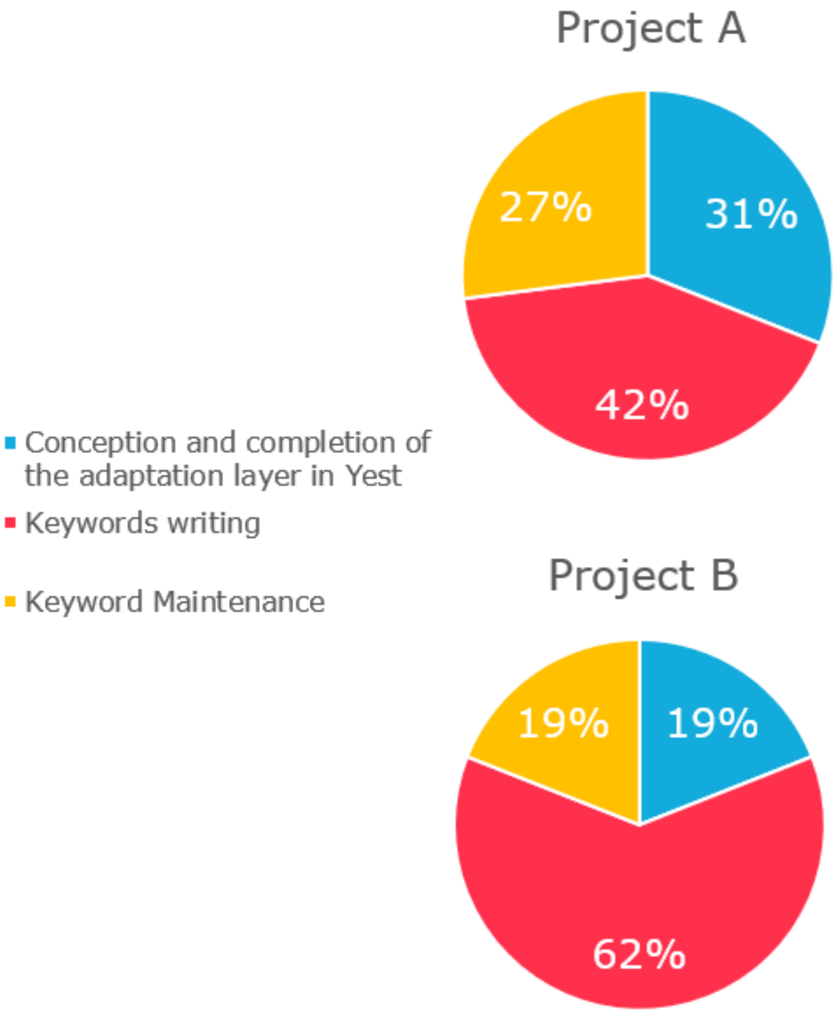


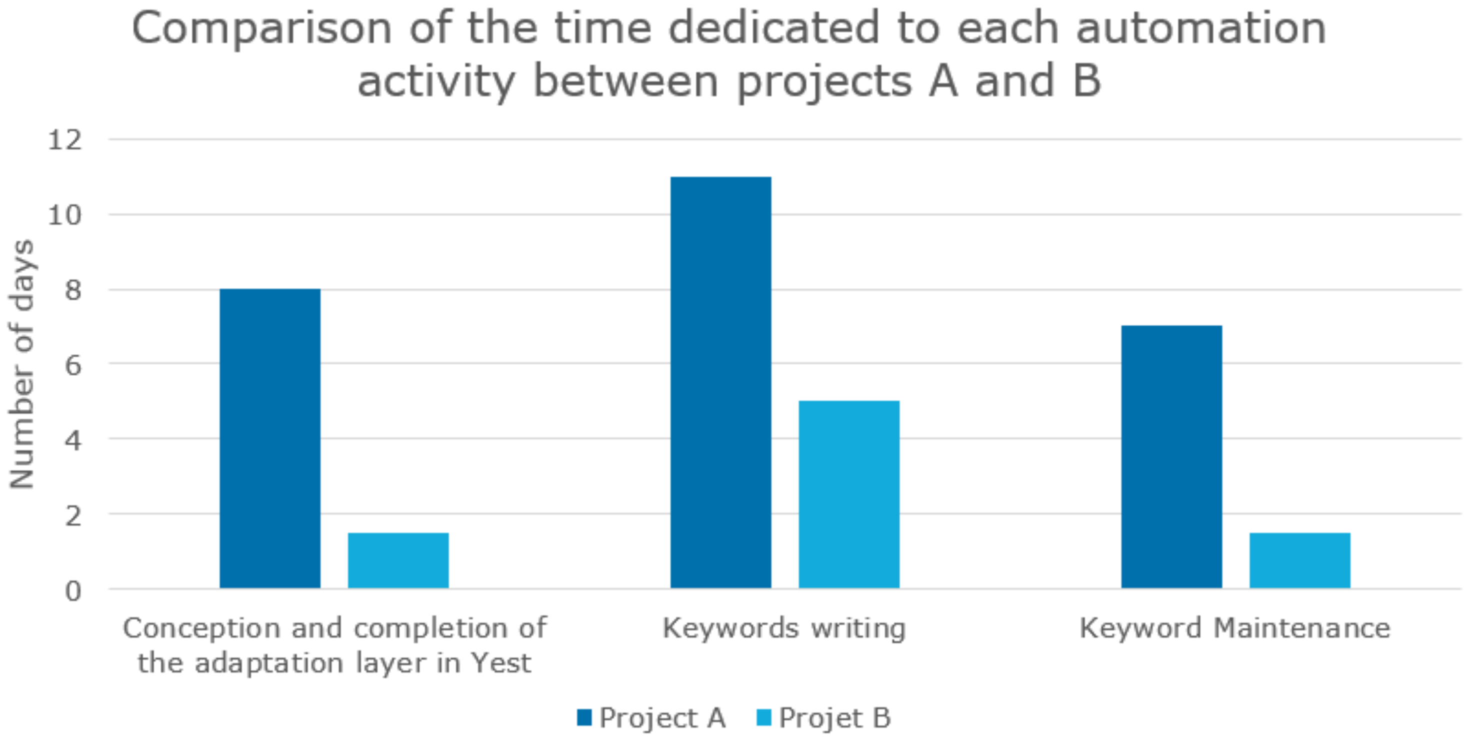
Improve clarity for functional testers on automated tests

Functional testers actively participate in automation

Maintenance improvement

Automation
Results





Conclusion

Increase visibility of the scope of the version

Improve the validation of the US

Clarify the business needs

Test Design and maintenance improvement

Functional testers actively participate in automation