Åbo Akademi University
Faculty of Science and Engineering

Maintainability of an RPA System Managing Termination of Insurance Policies

Author:
Lucie JORDAN
Student number: 42040

Supervisors:
Dr Sebastien LAFOND
M.A. Annamari SOINI

A thesis submitted in partial fulfilment of
the requirements for the degree of

Master of Science Technology
in
Computer Engineering

April 2019

ACKNOWLEDGMENTS

First, I would like to thank my supervisors, Sebastien LAFOND and Annamari SOINI for the advice they gave me, as well as the time they spent reading me.

Also, I would like to thank the colleagues I worked with at Atos, during my internship, for the knowledge they brought to me and for not losing patience for all the times I asked for help. Thank you also for offering me a good working environment and atmosphere that helped me to enjoy my return to France after a great year in Finland.

I am grateful as well for having friends who gave me their support during the whole writing of my thesis, pushed me and believed in me. Finally, a special thanks to my mother who has been my best support during this period of coding and writing.

ABSTRACT

Defined as the ease with which a code can be read, understood, reused, modified, and tested, maintainability is a major quality attribute. Indeed, an easily maintainable project is more likely to have a long lifetime than a project with low maintainability. In industry, systems with the highest lifetime probability are preferred, since the project may be subject to modifications and corrections of anomalies. That is why it is essential for the system to be easily maintainable.

Besides, use of Robotic Process Automation (RPA) within companies is becoming more and more frequent. RPA, regarded as a cousin of AI, consists in reproducing through computer processing the tasks usually performed by a human, and gradually begins to replace white-collar workers to perform repetitive tasks. This allows the company to both save money and reduce their employees' workload.

With the entry into force of a new law, the "loi Hamon", French insurers are spending more and more time terminating contracts, a task with a low added value. To address this situation, the implementation of a software robot, based on RPA and performing the termination of insurance contracts, has been studied, while focusing on the maintainability of the system. To perform these terminations, a batch process was implemented in order to reproduce the usual employee's process on the internal system of an insurance company. The limited budget being a major constraint, Selenium, a free and open source tool, was used as an alternative to the existing RPA tools.

Metrics such as LOC's, cyclomatic complexity of the code, and code smells were used to perform the analysis of the implemented system. The analysis of the system implementation has shown that the system is sufficiently maintainable, although it has been suggested that it could have been improved by modularising it differently. Future studies on a similar system could focus mainly on the limitations of Selenium WebDriver as an alternative to the RPA tools available on the market.

Keywords: RPA, process automation, maintainability, software, robot, insurance, Selenium

CONTENTS

ABBREVIATIONS AND TERMS

AI	Artificial Intelligence
BPMN	Business Process Model and Notation
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CC	Cyclomatic Complexity
CFG	Control Flow Graph
CLOC	Comment Lines of Code
CRM	Customer Relationship Management
DMS	Document Management System
DOM	Document Object Model
ERL	Electronic Registered Letter
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
LLOC	Logical Lines of Code
LOC	Lines of Code
MI	Maintainability Index
MRH	Multi-Risk Home (insurance)
MV	Motor Vehicle (insurance)
NCLOC	Non-Comment Lines of Code
NLP	Natural Language Processing
OCR	Optical Character Recognition
POC	Proof of Concept
POM	Project Object Model
REST	Representational State Transfer
ROI	Return on Investment
RPA	Robotic Process Automation
T&Cs	Terms and Conditions

GLOSSARY

Artifact (Maven)	An artifact is a file. It is usually a JAR file, and is generally deployed to a Maven repository, such as a Nexus.
DOM	In HTML, the DOM is an object model for HTML from which a web page is created. The DOM is built as a tree of HTML objects, in which HTML elements are defined as objects with their properties, methods, and events.
Motor vehicle insurance	In France, there are several packages for car insurance. Each package covers at least the driver's civil liability and the damage that may be caused to third parties in the event of an accident.
Multi-risk home insurance	In France, home insurance only covers damage caused by fire or water damage. Multi-risk home insurance covers rental risks: in addition to covering the risks to which a home is exposed, it also covers people living under the same roof as well as their belongings.
Nexus	Nexus is a repository manager which can be used by Maven. Artifacts, such as JAR files, are deployed on it so that dependencies can be easily managed and made available to other developers.
NLP	NLP is a subfield of AI that aims at analysing natural language by applying computational techniques.
OCR	OCR is the recognition of text characters by a computer. The text can be printed, written, or handwritten.
POM	Maven's operation is based on an XML file called POM. The POM contains the configuration information used by Maven to build a project, as well as information about the project.
RESTful web service	A RESTful web service is a web service that follows the REST software architectural style.

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

For some decades, technology has been fully integrated to human lives. More progress in technology has been done in a century than ever before, and the current generations witness a significant development of innovative technologies. At the end of the eighteenth century, many technological innovations emerged, marking the beginning of the industrial revolution. The agrarian society was turned into an industrial society where machines were invented and built to assist workers and to produce more and faster. After this period, a new era began. As the cars have replaced the horses, some machines substituted blue-collar workers. The way people were living drastically changed: workers had to acquire new skills and drop others in order to adapt to this new society. The rule was as simple as "adapt or die". Therefore, humans had to learn to coexist and work together with these machines.

Afterwards, automating tasks which were originally executed by humans became increasingly common. Repetitive low-skilled tasks were logically the easiest to automate. Then, along with the technology progress, came another form of automation: artificial intelligence (AI). Traces of AI appeared in science-fiction novels and films, depicting artificial characters who were given the faculty to think, thereby reflecting what some people thought about the future. Over time, ways to develop AI systems started to be studied. These systems became extremely popular even though they remain complicated to implement. Recently, a new AI-related form of automation has emerged. Called Robotic Process Automation (RPA), this new automation technology is based on the principle of reproducing tasks originally performed by users on graphical user interfaces. Even though one might think a robot is necessarily a physical device, in RPA, the robot is totally software-based. It means that the robot is actually a piece of software whose actions are executed on a computer system [1]. Faster to be developed than AI systems, RPA systems are increasingly in demand. As a result, IT companies have to adapt so that they can develop such systems for their customers.

The European IT consulting and services company Atos aims at offering solutions based on innovative technology for different industries, including the insurance industry. The majority of the software asked to be developed by insurance companies is internal use software. Thus, RPA systems are beneficial for these companies if cost and time are saved in comparison to what they already have spent at present. That is what a French insurance company has been suggested when Atos made a proposal for the development of a piece of software for internal use to help to terminate insurance policies.

As a matter of fact, a few years ago in France, insurance company customers were required to know their insurance policy renewal date[1] if they were not willing to renew it. Indeed, a policy could be terminated by notice two months before the renewal date. If the customer had not asked sufficiently early for the policy to be terminated, it was automatically renewed for a year. In 2014, a law concerning, *inter alia*, the termination of insurance policies was voted through in France. This law, named "Loi Hamon" – "Hamon law" in English – gives customers the capability to terminate their insurance policies whenever they want after a year of subscription. Even though only the motor vehicle insurance (MV) and the home insurance (MRH[2]) are concerned, the idea behind the creation of this law was to give back power to the customer. Since the law's entry into force on the 1st of January 2015, insurance companies have received a substantial amount of notices of termination. Indeed, customers have realised that requesting the termination of their insurance policy requires lower effort than before. Having also the possibility to give authorisation to the new insurer to take care of the administrative procedures for the termination on their behalf makes it even more time-saving. As a result, the former insurers are required to process a more considerable number of requests than they were used to within an identical allocated time. The recurring problem is that as soon as the notice of cancellation has been received, the policy should be broken within a month, thus making processing those notices a priority. Therefore, it must be done at the expense of other tasks, thereby being costly and time-consuming for the company.

To overcome this problem, the insurance company wants a system that would automatically handle the termination of the insurance policies. In their case, to terminate an insurance policy is repetitive and involves a considerable amount of interactions on a dedicated graphical user interface (GUI), including a lot of copy and paste operations. The amount of time spent to execute those actions is becoming too valuable compared to the added value of the task. Being considered a low value-added task, the idea would be to automate what is currently executed by the employees so that they could be more productive on higher value-added tasks, thus saving time and money.

To meet this need, Atos offers a solution based on RPA. The purpose of this thesis is to study the implementation of this solution while focusing on its maintainability. Indeed, the developed RPA system should be sufficiently

[1] In that case, the renewal date is the date at which the policy ends and is automatically renewed.

[2] In France, the multi-risk home insurance ("assurance multi-risques habitation") mainly covers all damage that can happen to buildings. Tenants are legally required to subscribe; however, it is optional for owners.

maintainable so that the next team working on the project will do so with ease. Different criteria have to be taken in consideration, such as having a fitting architectural design and being sufficiently documented. The role of the implemented robot would be to execute on the GUI the interactions usually done by the employees to terminate a policy. Those actions involve reading an email, downloading email attachments or files from an extranet website, and copying and pasting information. As the budget is limited, it is desired to use a cheap tool, allowing tasks to be automated on a browser, and one that could offer an alternative to the RPA tools available on the market. The system under study is a Proof of Concept (PoC) which aims at demonstrating the feasibility of such a system.

Similar systems using RPA already exist. There are diverse types of business processes for which RPA can be applied, in all types of industries. Mostly, RPA concerns back office processes and particularly repetitive tasks with low complexity. A Capgemini study focusing on those processes showed that they are indeed the most suitable for automation. This is why tasks that do not directly involve a customer are considered better candidates. [2, 3]

In 2018, an automated insurance claim registration system has been developed and it is relatively similar to the one studied in this thesis. Due to all the verifications and copy and paste that had to be done by the employees when receiving a claim, the insurance company had realised that registering those claims was time-consuming and costly. In an attempt to solve this problem, a solution was requested. The proposal that was obtained consisted in automating the interactions performed on the computer system during the registration of these complaints. Fundamentally, the new system complements the existing one: the existing system used by the employees to do their tasks is kept, and the automation system operates on it as if it were a human performing the interactions [1]. Even though this thesis is not about registering claims but terminating insurance policies, a similar approach is taken for the development of the system of the case study.

In Chapter 2, a definition of maintainability and explanations about its role and importance in a software development project are given. Some maintainability metrics are described. A maintainability index based on metrics such as the Lines of Code, McCabe cyclomatic complexity and Halstead complexity metrics is introduced.

Then in Chapter 3, the operating principle of RPA is explained, and details are provided to clarify what it involves. The benefits and drawbacks of the use of RPA are then analysed, comparing different impact areas. Some popular RPA tools and their main characteristics are presented.

Chapter 4 is dedicated to the case study of this thesis, being the development of a robot allowing the termination of insurance policies, and the technologies used. In this part, a clear objective of the project is defined, followed by the description of the current process for terminating policies.

In Chapter 5, the implementation of the system under study is described. The software specifications are presented, then the architectural and coding choices that were made are given.

Finally, the results are exposed and analysed in order to evaluate the system and determine how maintainable it is. Next, the results are discussed and a conclusion on the study that was conducted for this thesis is given.

2 CONCEPT OF MAINTAINABILITY

When developing software through a project, different steps to achieve the main goal should be considered. In most cases, some of them are taken more seriously than others by the different teams in charge of the realisation of this project. For instance, understanding the need for such a product, establishing the implementation strategy, and achieving the implementation are important steps on which the project team should focus. Yet designers, architects, and developers should think beyond the development of the functionalities while designing the piece of software. Indeed, elaborating a product fulfilling the requirements and meeting the expectations of the customer is important. Nevertheless, software and its implementation should be adapted to the conditions under which it will be used, as well as to the environment of use. For example, a product that will be used on a long-term basis has different requirements than a product that will be used on a short-term basis only. Not only the functionalities are concerned, but also the way the software solution and its architecture have been thought out. Two situations can be considered: the product is a customer's request (S1) or, on the contrary, there is no customer yet, but the product will be developed for future customers (S2). In case of S1, an agreement on the duration of the project is made between the customer and the project team. Consequently, the product has to be worked on for a limited amount of time, thereby constraining the project team to prioritise certain tasks at the expense of others. On the contrary, in case of S2, the absence of a customer facilitates time management that can thus be optimised and adapted to the project. Ultimately, the quality of a system and its lifespan are essentially defined by the decisions made at the design stage.

2.1 DEFINITION OF MAINTAINABILITY

During the software development process, the study of the architecture of the future software system remains a crucial step. This architectural study is performed in advance, at an early stage of the process, based on defined functional and non-functional specifications. Defining the functional specifications is the first step of the process. Once they are settled, the software architects know what the system is supposed to do and can start working on the non-functional specifications, also called quality attributes. The role of these specifications is to define the quality of the piece of software. While functional specifications determine what the system is supposed to do, the quality attributes aim at defining how it should be done by the system.

A software system has two main types of "users": the users of the system using it as a product, and the developers working on the development of the

system. As a result, two categories of quality attributes can be noticed. The first category includes those that aim at improving the user experience, such as usability, reliability, and performance. The second category gathers the attributes that have an impact on the developer experience, such as compatibility, security, and maintainability. Many quality attributes from both categories can be selected to improve the quality of the software to be developed. However, the choice of a quality attribute and the decisions resulting from it can have an impact on the other attributes previously chosen. In that case, compromises are necessary if several are to be chosen.

Maintainability is one of the most important quality attributes, as it enhances the quality of the software system on a long-term basis. Fundamentally, maintainability is the ability of being maintainable, the ease of maintaining. A literature review provides different definitions of maintainability.

According to International Standards Organization (ISO) 24765, maintainability is "the ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment". [4] ISO 25010 defines maintainability as "the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers" [5]. Considering that effectiveness is the ability to be effective, to achieve a goal, and efficiency is about how performant the maintainer is, maintainability can be described as the ability of the person in charge of the maintenance to accomplish a task successfully in a positive state of mind. In addition, maintainability is a characteristic described as a set of the following subcharacteristics: modularity, reusability, analysability, modifiability, and testability. [6]

These characteristics are defined by ISO 24765 and 25010. A system has high modularity if it is composed of discrete components so that the impact of a change on one component on the others is minimal [4]. Analysability is defined as the ability to analyse the impact of a change in a system. [5] As a result, modularity and analysability can both influence modifiability, which is the ability of the software system to be modified without having a negative impact on its quality. [5] Then, reusability is the degree to which the various software modules can be reused. Therefore, high reusability means that software modules can be used to build other software or that other modules can be built based on them. [5] Finally, testability is defined as the ease with which the criteria to be tested, and the tests to be implemented to know if these criteria have been met, can be determined [4, 5]. For instance, the shorter the

methods and classes are, the easier they are to test, because then the determination of the criteria to test is more straightforward.

To summarize, in order to have high maintainability, the software should be partitioned into several modules to prevent errors due to probable future changes. Additionally, its code implementation should be easily understandable so that it can be modified with ease, but also for the purpose of making simple the detection of the impact that a change would have on the system. To make it comprehensible, the complexity of the code should be low. The code has to be commented and have documentation where code samples would ideally be given. The given names in the code should not be too long, and they should be explicit: the name should disclose the content and the functions of the object. Besides, as many modules composing the software system as possible should be generic, in order to be reused later for other systems. Lastly, the classes, if any, and methods should be as short as possible and implement a small number of functionalities with the purpose of being easily tested.

Therefore, thinking about the maintainability of the software system requires a serious time commitment from the beginning of the project, including the architecture study of the future software and its partitioning. For that reason, the benefits that high maintainability of the system under development can bring should be assessed as soon as possible.

2.2 IMPORTANCE OF MAINTAINABILITY

When thinking short-term, ensuring that the new software has high maintainability is expensive in terms of time and money. However, as it has been reported by the Network for Business Sustainability, in business, long-term thinking should be chosen over short-term thinking [7]. In a society where change is omnipresent, the capacity to adapt fast is a quality that is important to acquire. Indeed, as, generally, the software systems to be developed are desired to be used on a long-term basis, it is preferable to make them easily maintainable. The reason for this is that a software system used on a long-term basis will be subject to modifications because of demands of functionality additions, improvements, or removals, or simply bug corrections. Change being inevitable, software with low maintainability can be extremely costly.

In 2009, a survey showed that more than 50% of the costs of the software systems of a company is for maintenance [8]. In fact, what happens is that a software system is often to be implemented in a certain amount of time. Generally, this allocated time span is too short to implement a system that is of high quality. For instance, the implementation of all the required functionalities and the conception of software that works and has few bugs

would take precedence over the design of a good architecture. The consequences would be such that the product obtained would meet the functional requirements of the customer, but the product could be non-viable in the long term if it becomes costly to maintain.

Even though, at the beginning of the project, putting effort into the study of the maintainability of the system is time-consuming and costly, the cost will be amortised over time. Software whose architecture has been designed to ensure easy maintenance should be less costly to maintain than software whose architecture has been poorly designed. In fact, regarding software with low maintainability, time and human resources required for maintenance are reduced, resulting in a cost attenuation in the long run.

Saving a significant percentage of the total cost of the software solution, reusing some software components, and choosing to maintain or redevelop them [8] are all reasons that cause maintainability to be one of the most crucial quality attributes in terms of costs. From this stems the importance of measuring software maintainability.

2.3 MAINTAINABILITY METRICS

To know how maintainable a system is, it is first necessary to evaluate how complex the code is. Indeed, complex code is difficult to test and to maintain, thereby causing a significant increase in errors and in the software costs. Thus, for assessing the quality of the product under development, traditional metrics are used. The Lines of Code (LOC) metrics and Halstead complexity are popular metrics whose role is to measure the size and the complexity of the code. The McCabe complexity metrics is just as well-known and aims at measuring the structure of the software system. By means of those three metrics, the Maintainability Index (MI) can be calculated, and the software quality can be improved. [9]

2.3.1 Lines of Code

The Lines of Code is one of the easiest to understand of the metrics that are used to measure maintainability, for the reason that it simply consists of counting the lines of a source code. The code is not evaluated on its content or on whether it does what it is supposed to do; only its size is measured.

Several types of lines are considered when evaluating the size of the software solution. First, the number of physical lines of code (LOC) of the source code is the total amount of lines in the source. Then, the lines of the program, which are the logical lines of code (LLOC), are counted separately from the commented lines of code (CLOC). Finally, the non-commented lines (NCLOC) correspond to all the blank lines found in the code. Note that when

counting the number of each type of lines, a physical line containing logical code and comments represents one LLOC and one CLOC. [9]

The number of lines will depend on the programming language used as well. It can vary a lot from one language to another; a code written in five lines in a programming language can be written in twenty lines in another one. According to Klaus Lambertz, the size of a method should be less than forty lines, a class or a file should be shorter than four hundred lines, and comments should represent 30% to 75% of the file. That way, the file would be easier to understand. Indeed, a class with a big size easily discourages people from reading and understanding the code, whereas a short one could be more motivating. If a method is too long, it is probably because it implements too many functionalities. If so, some code of the method should be extracted, and new methods could be created. Regarding a class, if it does too many things it can be difficult to test. [9]

```
// Calculate a modulo b
public int modulo1(int a, int b) {
// Declare variables
int c;
int x;

x = a/b;
c = a - b * x;

return c;
}
```

Figure 1: Code snippet 1 for modulo method

```
// Calculate a modulo b
public int modulo2(int a, int b) {
int x = a / b; // Declare variables
return a - b * x;
}
```

Figure 2: Code snippet 2 for modulo method

The code snippet given on Figure 1 shows a Java method that calculates the modulo for a modulo b. There are eleven LOC's, seven LLOC's, two CLOC's and two NCLOC's which means 18% of the file is comments. On Figure 2, the method has the same function but has only five LOC's, four LLOC's, two CLOC's and no NCLOC's, so the comments represent 40% of the code file. The examples given on Figure 1 and Figure 2 are easy methods, but in practice the source code is more complicated: it would require more comments for the first method, and more lines for the second one to make it more readable.

By giving information on the size of the program, the LOC metrics give information on the understandability and readability of it. However, it does not give information on the complexity of the code or on its content, thereby encouraging the use of other metrics to complement the evaluation.

2.3.2 Halstead complexity measures

In 1977, in *Elements of Software Science*, Maurice Halstead introduced the Halstead's metrics with the aim of quantifying the complexity of the code when writing it.

Table 1: Halstead metrics

Halstead metric	Name
N_1	Total number of operators
n_1	Total number of unique operators
N_2	Total number of operands
n_2	Total number of unique operands
N	Program length
n	Vocabulary size
V	Program volume
D	Difficulty level
L	Program level
E	Effort to implement
V	Implementation time
B	Number of delivered bugs

All the Halstead metrics, listed in Table 1, measure something different, but all of them are calculated based on four values. Those values are N_1, the total number of operators, n_1, the total number of unique operators, N_2, the total number of operands, and n_2, the total number of unique operands. The operators and operands are called Halstead operators and operands and those in Java are listed in Table 2 and Table 3. [9]

Table 2: Halstead's operators in Java [9]

IDENTIFIER	all identifiers that are not reserved words
TYPESPEC	(type specifiers) Reserved words that specify type: bool, byte, int, float, char, double, long, short, signed, unsigned, void. This class also includes some compiler specific nonstandard keywords.
CONSTANT	character, numeric, or string constants.

Table 3: Halstead's operands in Java [9]

SCSPEC	(storage class specifiers) Reserved words that specify storage class: auto, extern, register, static, typedef, virtual, mutable, inline
TYPE_QUAL	(type qualifiers) Reserved words that qualify type: const, friend, volatile, transient, final
RESERVED	Other reserved words in Java: break, case, continue, default, do, if, else, enum, for, goto, if, new, return, asm, operator, private, protected, public, sizeof, struct, switch, union, while, this, namespace, using, try, catch, throw, throws, finally, strictfp, instanceof, interface, extends, implements, abstract, concrete, const_cast, static_cast, dynamic_cast, reinterpret_cast, typeid, template, explicit, true, false, typename. This class also includes some compiler specific nonstandard keywords
OPERATOR	! != % %= & && || &= ( ) { } [ ] * *= + ++ += , - -- -=-> . ... / /= : :: < << <<= <= = == > >= >> >>> >>=>>>= ? ^ ^= | |= ~ ; =& “ “ ‘ ‘ # ## ~

The program length N is the result of the addition of the number of operators N_1 and the number of operands N_2.

$$N = N_1 + N_2$$

Formula 1: Program length

The vocabulary size n is obtained by adding the number of unique operators n_1 and the number of unique operands n_2.

$$n = n_1 + n_2$$

Formula 2: Vocabulary size

The implementation size of an algorithm is described by the Halstead volume V. Unlike LOC metrics, this metric is not based on the code layout. Its value is obtained based on N and n values as follows:

$$V = N \times \log_2(n)$$

Formula 3: Program volume

The limits of the volume for a file and a function have been defined based on the McCabe complexity (cf. part 0) limits. The volume value of a function should be between twenty and one thousand, and between one hundred and eight thousand for a file. If the volume is bigger than the recommended maximum, the function probably does too many things, or the file should be split.

The difficulty level D expresses the tendency of a program to have errors. The more the same operands are used in a program, the more the program can have errors. The following formula is used to calculate the difficulty value:

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2}$$

Formula 4: Difficulty level

The program level L is given by the inverse of D. A program of high level is of better quality than a program of low level.

$$L = \frac{1}{D}$$

Formula 5: Program level

Then, the effort to implement a program E is calculated based on its volume V and difficulty D. It represents the ease to implement or understand the program.

$$E = V \times D$$

Formula 6: Effort to implement

Proportional to the effort E, the implementation time T is given in seconds. The formula has been determined empirically by Halstead and is:

$$T = \frac{E}{18}$$

Formula 7: Implementation time

Finally, the number of bugs B can be calculated by using the effort E. It estimates the number of bugs a program could have, and could be used as an indication of the number of errors that could be found when tests are run. B is calculated using this formula:

$$B = \frac{E^{\frac{2}{3}}}{3000}$$

Formula 8: Number of bugs

Even though the metrics of Halstead have been introduced more than forty years ago, they remain relevant as indicators to analyse the quality of the program. [9]

2.3.3 McCabe cyclomatic complexity

Introduced by Thomas McCabe in 1976, the cyclomatic complexity (CC) measures the complexity of the code by indicating the number of conditional branches of a program, thereby being language-independent. In other words, this number of branches represents the number of independent linear paths of the program. When calculating the cyclomatic number for a class or a function, v_G is initialised to one. This complexity gives an indication on the number of tests to be written to test the program, and thus the efforts to do so. For instance, a program consisting of sequential lines of code has a complexity $v_G = 1$ and can be easily tested. On the contrary, a code made of several conditional branches has a higher complexity. So, the higher the complexity is, the better the maintainability is, and the more effort is needed to test the code. In the end, the number of test cases for path coverage that are to be written is at least equal to the cyclomatic number v_G. [9, 10]

To calculate this cyclomatic number v_G, several types of branches are considered, and each of them increases v_G by an increment of one. An if-statement, a for, or a while loop, a catch-block, a case-branch, use of the ternary operator (`expr1 ? expr2 : expr3`), the logical conjunction (&&), and the logical disjunction (||) are all types of language instructions that are counted to obtain the cyclomatic number. In the end, for a file the cyclomatic number is the sum of the cyclomatic number of each function in the file:

$$v_G = \sum_{i=0}^{n} V_{G_i}$$

n being the number of functions in the file. [9]

For instance, a function with two for-loops, four if-statements, two uses of the logical conjunction, and one while-loop has a cyclomatic number $v_G = 9$, so there are at least nine execution paths to test.

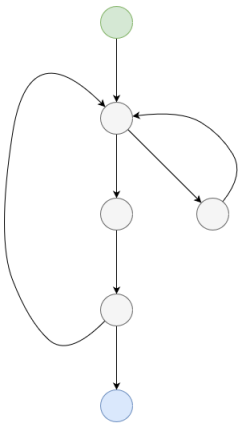


Figure 3: Example of a CFG

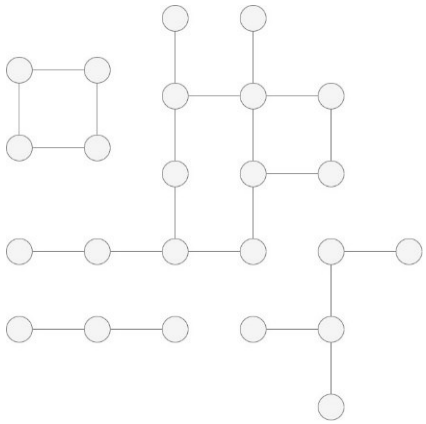


Figure 4: Graph with four components

The cyclomatic number can also be calculated using the control flow graph (CFG), which represents the paths that can be followed by a program during its execution using nodes and edges. A node is the representation of a simple conditional branch: if the conditional branch has two or more predicates, it is divided into several nodes so that there is only one predicate per node. An edge represents a jump between two nodes. A CFG can regroup many components, called "connected components", that are sub-CFGs. As examples, Figure 3 shows a CFG with a while-loop and an if-statement, and Figure 4 illustrates a graph having four components. When looking at the control graph of a piece of code, the following formula can be used:

$$v_G = E - N + 2P$$

Formula 9: Cyclomatic number

where E is the number of edges in the program, N the number of nodes, and P the number of connected components.

To demonstrate good complexity, the cyclomatic number should be evaluated to less than $v_G = 15$ for a function, and to less than $v_G = 100$ for a file. [9] [10]

The cyclomatic complexity is strongly controversial, mainly because it was introduced a long time ago (1976) compared to the evolution of computer coding. Now, it is sometimes considered no more useful than LOC. However, it is still widely used in industry. Indeed, CC is easy to calculate, apprehend, and understand. That is why CC should be used simply as an indicator that helps at improving the quality of the code, and whose value v_G should only be used to give an idea of the complexity of the code, and in conjunction with other metrics for evaluating maintainability. [10]

2.3.4 Maintainability index

The idea of a maintainability index is to be able to know when it is worth the effort to rewrite the code instead of modifying it. This index is calculated based on the LOC, the Halstead metrics, and the McCabe complexity. First, the MI without comments is calculated using the following formula:

$$MI_{woc} = 171 - 5.2 \times \ln(\bar{V}) - 0.23 \times \overline{v_G} - 16.2 \times \ln(\overline{LOC})$$

Formula 10: Maintainability Index without Comments

where $\bar{V}$ is the average value of the Halstead volume V of each module, $\overline{v_G}$ the average value of the cyclomatic number v_G of each module, and $\overline{LOC}$ the average value of the number of physical lines of each module. Then, the MI of the comment weight is calculated as follows:

$$MI_{cw} = 50 \times \sin\left(\sqrt{2.4 \times \frac{CLOC}{LOC} \times 100}\right)$$

Formula 11: Maintainability Index of Comment Weight

where $\frac{CLOC}{LOC} \times 100$ is the percentage of comments of the evaluated element. As a result, the maintainability index is obtained by adding MI_{woc} to MI_{cw} :

$$MI = MI_{woc} + MI_{cw}$$

Formula 12: Maintainability Index

Therefore, the higher the MI is, the higher the maintainability of the code. For a non-object-oriented programming language, if the MI value is more than 85, the code is considered good. A moderate maintainability is given by an MI between 65 and 85, and a code with an MI below 65 is considered poorly maintainable. [9]

However, the MI only suggests how easily the person in charge of the maintenance should be able to understand and make modifications to the code. Indeed, considering that the calculation is based on the style of the code and not its content, and that every programming language has its own writing standards, the limit values can vary from one language to another. The MI has become highly controversial, most of the time considered to be no more useful than the LOC metrics, also because its variation cannot be easily predicted. Indeed, the formula given is complex and depends on multiple factors which may influence each other when a change is made. Consequently, a developer cannot predict easily how the MI will vary [11]. Besides, in 2012, a case study questioning the maintenance metrics was published [12], explaining that no information seems to exist about the MI value that an object-oriented programming language should have. Since for some years the MI was highly

debated, it is very likely that no research or study had been done to determine these values. However, in 2015, a case study showed that no absolute limit values could be determined for object-oriented languages, and that in these cases, the MI should be used with caution [13].

2.3.5 Comments and documentation

During the life of a project, the developer in charge of the code can change. Avoidable waste of time can be prevented by increasing the readability of the code and making it a crucial quality attribute of the project. Indeed, any newcomer must be able to understand fast enough the code written before: how the code works, what elements are used, where and how they are used. That is why comments are essential to the understanding and readability of the code. Comments are used to describe the code and answer the following questions: what is this element? What is its purpose? How can it be used? Does it need parameters? If so, what are they? To recap, comments must make the code understandable. [14]

```
/**
 *  This is the Javadoc of the following method
 *  The following method deducts b from a
 *
 *  @param a  the value from which b is deducted
 *  @param b  the value to deduct from a
 */
public int subtraction(final int a, final int b) {
    return a-b;
}
```

Figure 5: Code snippet with Javadoc

For instance, Javadoc is a Java documentation that can be automatically generated as easy-to-read HTML files, so that the next developer can understand what the previously implemented elements are. The Javadoc of a project is written in the code by using the characters "/** */"and its role is to describe the functionality of an element in the code (e.g. a class, a method, an interface, etc.), such as on Figure 5 where the functionality of the method is described and the purpose of each parameter is given.

To determine the comment ratio in the source code, the percentage of comment lines in the source code has to be calculated (Formula 2).

$$perCM = \frac{CLOC}{LOC}$$

Formula 13: Percentage of comments

A file should have more than 30% of comments, and less than 75%. A comment ratio of less than 30% indicates that either the file is trivial to understand, or explanations are missing. Despite giving information on the

quantity, this method does not estimate how good the comments are. As a result, the developer must show some common sense and update them when the code changes so that the comments will not be misleading. Giving self-explanatory names to methods and variables helps to reduce the comment quantity. [14]

2.3.6 Code smells

In his book *Refactoring: Improving the Design of Existing Code* [15], Martin Fowler introduced for the first time the concept of "code smells". He defines "code smells" as poor software design practices that may have an impact on future software development or maintenance. Fowler explains that different types of code smells can be defined and that no exhaustive set can be given, considering that well-informed developers have their own intuition. He also specifies that what may be a code smell does not necessarily have an impact on future modifications, but remains an indication of what could be modified in the code to avoid major development slowdowns in the future. Guidelines and examples are provided. For instance, objects, methods, or classes have to be named with an unambiguous name so that no confusion is possible, and so that a future developer reading the code does not search long for any explanations. He adds that the shorter the methods are, the better, because too long methods can be difficult to read. He also explains that code duplication should be avoided to facilitate the modifications of the code and to avoid forgetting any necessary changes. Fowler explains plenty of code smells and what solutions should be taken into consideration. [15]

To summarise, there are different categories of code smells. For each code smell, they may be a solution that is considered universal but that is not exclusive.

2.3.7 Technical debt

Introduced in 1992 by Cunningham, technical debt is an analogy of financial debt [16]. This means that rather than constantly reimbursing interests, it is better to repay the totality of the debt one day. For technical debt, this is directly related to software quality. It depends on the software design but also on the source code, the documentation, and the tests. This debt has no immediate effect, and that makes it invisible at first sight [17]. In the article "Technical Debt: From Metaphor to Theory and Practice", from IEEE Software (2012), technical debt is defined as "the invisible result of past decisions about software that negatively affect its future" [18]. However, technical debt may be intentionally increased to achieve some short-term objectives, but it should be moderated so that it does not become too large and does not have a significant impact on the maintainability of the software

system [16]. The cost of the technical debt can be estimated with a duration: each task to be performed to reduce this debt has an estimated correction time. This cost can be estimated thanks to some predictive models that are used by tools like SONAR (cf. section 4.4.6.3) [19]. For example, an attribute should not have the same name as the class to which it belongs. This increases the technical debt because it can make future developers or maintainers confused and therefore prevent a quick understanding of the code. SONAR estimates that ten minutes are needed to correct this problem.

All in all, technical debt is a measure that shows the consequences that a lack of maintainability of a piece of software can have for the future in terms of costs for a company.

3 ROBOTIC PROCESS AUTOMATION

In modern society, use of information technology tools is progressively more recurrent if not necessary. These tools are playing a key role in companies, sometimes becoming indispensable. As an example, in a company, office employees may have to spend more time on repetitive low value-added tasks than on some higher value-added tasks. This is problematic for the company in terms of costs. Some may even find themselves forced to hire employees to do only these tasks, which can quickly become tedious for the person performing them. Not forgetting to mention that in the current competitive world, businesses are looking for tools allowing them to provide the best customer experience. Therefore, the automation of such processes could help to overcome this problem.

3.1 DEFINITION OF ROBOTIC PROCESS AUTOMATION

Robotic Process Automation (RPA) is the automation of repetitive rule-based tasks using a software robot. More precisely, it consists in automating the performance of well-defined tasks initially performed by a human on a computer. Regarded as a cousin of Artificial Intelligence (AI), which consists in simulating human intelligence by using machines, RPA differs widely from it. In both cases, there is a robot performing tasks usually performed by a human. The main difference is that RPA is rule-based, and AI is based on algorithms used to analyse and interpret data. While AI gives a set of likely results, tends towards self-learning, and is therefore probabilistic, RPA provides predetermined answers, hence making it deterministic. [2]

The tasks for which RPA can be expected to provide a solution are specific. What is to be robotised are repetitive tasks with a high volume, frequently performed, and with a low added value. In this way, tasks requiring real human intervention, with greater added value, can be focused on. The main objective is to make the job of these people more motivating and interesting by allowing them to focus on tasks that are more enjoyable to do. However, the tasks that are to be robotised must be targeted: in order to limit the risks, they must initially be perfectly understood. Indeed, the digital transformation of a company will be more easily accepted by its employees if the first experiments have borne fruit. Thus, starting by robotising the most time-consuming and repetitive tasks should be a priority, so that results can be shown as quickly as possible. Nevertheless, in some situations the robot will not be able to perform the task since it is not able to think. In these cases, human beings must be able to intervene to achieve it by themselves. Therefore, it must be possible to take control of the robot at any time. [1] [2] [3]

Overall, the aim of RPA is to robotise some tasks performed by white-collar workers as had been done for blue-collar workers during the industrial revolution. Ultimately, this raises questions about job loss and job creation, leading to wondering what benefits RPA could bring.

3.2 BENEFITS AND DRAWBACKS OF ROBOTIC PROCESS AUTOMATION

In many ways, technology has brought facility into human life. Indeed, technology is continuously evolving year after year, becoming more and more advanced, with the result that it is now possible to automate a wide range of manual labour and office tasks. The reactions to this are opposed. Technology pessimists find it frightening, while technology optimists find it exciting. Certainly, automation can have both negative and positive effects. In this section, different arguments will be given. The idea is not to prove that RPA is good or bad, but to show different advantages and drawbacks of such automation for the current society.

3.2.1 Impact on the employees

During their working hours, office workers have to use the tools of the company's information system to perform properly the tasks they have been assigned. They constantly navigate between the ERP, CRM, DMS, intranet, extranet, or any other internal company software. Some tasks consist of copying and pasting, or checking that the data is identical. This can quickly become repetitive, daunting, and time-consuming. In addition, they are asked to do more and more over the same time frame. When the managers do not want to hire an additional person to perform these tasks or reduce the workload, this can result in mental overload and decrease the employees' well-being, sometimes making them ineffective.

In fact, one of the main arguments in favour of automation is the reduction of the employee's workload. Because every employee is different, everyone experiences working conditions differently. Working conditions are defined as the set of elements that make a job more or less difficult for a worker. These elements are a combination of work-related constraints and employee-specific capabilities. First, they include the requirements of the task such as deadlines, complexity, or diversity. Additionally, the social environment in the company (e.g. relationship with colleagues or hierarchy), the physical environment (e.g. noise, heat), and the design (e.g. arrangement) of the workstation have influence on these conditions. Then, since every individual is defined by a personality and skills of his/her own, everyone perceives his/her work differently. The employee's age and state of health can have an impact on working conditions, but also the functional state such as stress and fatigue. Not to mention that each worker is influenced by extra-professional factors

that differ from one individual to another. This results in three different loads: the physical load, the mental load, and the psychological load. The workload is the set of the three of them. [20]

The physical load, being the physical effort required for carrying out the task and bearing its consequences, has very little to do with white-collar workers. Indeed, office workers are more subject to psychological and mental loads. The psychological load is related to elements that cause anxiety and can result in disorders such as mood disorders, insomnia, etc. while the mental load refers to the employee's cognitive activity. It is up to the mental capability of the employee to accumulate tasks needing cognitive capacity, such as making decisions, calculations, etc. A mental overload could lead, in some situations, to bad decision making with negative consequences. [20] For instance, calling several times wrong policyholders and asking them to pay their insurance premium when they already have done so could tarnish the company reputation.

This mental load will be impacted by RPA. By automating repetitive and rule-based tasks, employees could focus on more pleasant tasks. They would only be charged with carrying out the tasks the robot could not do because of its deterministic behaviour.

3.2.2 Impact on the companies

Companies are increasingly seeking to boost the productivity of their employees: the amount of work is constantly on the rise, while only a few new jobs are created. RPA appears as an economical solution to both increase employee productivity and improve their working comfort. Exempted from performing repetitive and boring tasks, employees can focus on higher value-added tasks. In addition, human errors are avoided since procedures and controls are strictly followed and performed. In 2017, a study on the automation of a business process [2] showed that a group of employees working with RPA could solve 20% more cases than a group of employees working without RPA over the same period. Although this may seem interesting to companies, they are often curbed by the investment cost that RPA implies. Prioritising repetitive rule-based tasks with high volumes, low added value, and few exceptions to manage [2] generally results in a ROI (Return on Investment – cf. Formula 14) that rises quickly enough to prove to the company the effectiveness of RPA. In addition, according to a Capgemini study, a full-time employee would cost three to five times more than an RPA robot licence [3].

Thus, by helping to enhance the productivity of employees, RPA is becoming increasingly more economically attractive.

$$ROI = \frac{Gain\ from\ Investment\ -\ Cost\ of\ Investment}{Cost\ of\ Investment}$$

Formula 14: Return on Investment

3.2.3 Impact on the job market

The automation of tasks usually performed by humans often raises issues about the loss or creation of jobs. Indeed, job losses are a major consequence due to the replacement of men by machines. The absence of job creation in return feeds a public fear: the risk of a significant increase in the unemployment rate. In the long term, jobs may be lost: low-skilled jobs would be taken over by robots. Employees would be asked to interfere only when needed to resolve complicated cases requiring interpretation or subjective judgment. The new jobs that could be created would involve, for example, the implementation of RPA tools or the maintenance of the robots set up, which means that more technical jobs could appear at the expense of business-oriented jobs.

Overall, the adoption of RPA by companies will result in a major change in the way the current society works.

3.2.4 Impact on the society

Adjusting the current society to a society in which robots have replaced humans at most jobs requires changes at the root that must be anticipated. Indeed, robots are only able to perform low-skilled tasks. This means that human beings must adapt in order to be able to perform complex tasks that cannot be performed by robots. In other words, to compensate for the disappearance of low-skilled jobs, it would be necessary to learn new abilities. To begin with, employees should adapt and train to acquire the skills needed for higher-skilled positions. It would also affect the education system: the knowledge to be acquired during the studies would be different, more advanced. Education should therefore be reformed and reorganized to provide students with the necessary and appropriate skills so that they could find a job in this society. [21]

The impact of RPA on the job market must be anticipated to prepare the population to adjust to this new and growing society, which emphasises intelligence. While manual labour jobs were the most common jobs during the Industrial Revolution, jobs demanding cognitive capabilities could be the jobs of the future. This raises the question of whether other types of jobs could be considered in the future.

For many centuries, work has been the centre of humans' lives, and people are spending more and more time working. At a conference, Kai-Fu Lee, an AI

expert, explained how automation could help the society to detach from this idea [22]. According to him, most people have spent too much of their lives focusing on their jobs, thereby becoming workaholics. As a result, people started to neglect their personal life, such as their love life or family life, as well as their health. Lee believes that humankind should learn to coexist and work together with AI so that humans could spend more time with their loved ones. Besides, he explains that since what differentiates us from AI is creativity, love, and compassion, the jobs requiring those skills will not be taken over. [22]

In the end, there would always be a need for creative people with strong technical skills, but also for psychologists, social workers, teachers, and people with other professions requiring compassion. RPA will affect the professional world at different scales: the choices made by companies will impact on their employees, but also on the job market, and by extension on society. RPA would provide economic benefits to companies while improving employee workload. However, measures may have to be taken to anticipate the disappearance of jobs taken over by robots, so that people acquire skills suitable for the new jobs that may emerge because of such a digital transformation.

3.3 TOOLS FOR WEB RPA

Recently, RPA has been gaining a foothold on the market. As process automation has become undeniably more popular, companies are increasingly willing to invest in it. However, most software for process automation requires paying a license fee, which makes RPA more difficult to access for low-budget companies. This section therefore introduces RPA tools that can be used for process automation on a browser, as desired for this case study, as well as a possible alternative to these tools: Selenium WebDriver.

3.3.1 Well-known automated process tools for RPA

To implement the desired robots, specific tools have been designed for RPA implementation. The most popular RPA tools include Automation Anywhere, Blue Prism and UiPath, which are currently world leaders in this field.

Table 4 presents the main differences between these three tools. For UiPath and Blue Prism, the cost of the licence depends on the number of robots used and the number of development platforms, while it depends on the number of automated processes for Automation Anywhere. In any case, the cost of these licences is in thousands of euros per year. The three of them enable browser and web automation. However, there are some major differences between those three tools. First, while UiPath and Automation Anywhere provide a tool to record the operations performed and generate a script

afterwards, Blue Prism does not, thereby making it more difficult to handle at first use. However, unlike Automation Anywhere, UiPath is easier to use by someone with little programming skills. Besides, UiPath also gives the possibility to have a first impression of the software functioning thanks to the free version that is made available – the functionalities are only limited. Online support as well as courses are made available in order for the developer to become familiar with the tools and learn the fundamentals.

Table 4: Automation Anywhere, Blue Prism and UiPath [23] [24] [25]

	Automation Anywhere	**Blue Prism**	**UiPath**
Licence cost	Depends on the number of processes	Depends on the number of robots used and the number of development platforms (thousands of euros per year)	Depends on the number of robots used and the number of development platforms (thousands of euros per year)
Free trial	Yes (30 days)	No	Yes Free Community Edition for personal use
Tool for recording the operations performed and generating the script	Yes	No	Yes
Web browser automation	Yes	Yes	Yes
User-friendly	Necessary to have some programming skills	Easy to configure	Intuitive and with good learnability
Training and assistance	Online courses	Videos Webinar	Documentation Guides Forum Videos Webinar
Training Lessons	Online courses	PDF files and a knowledge validation questionnaire	Videos, exercises, correction of the exercises, summary and multiple-choice knowledge validation questionnaire

To summarise, Automation Anywhere, Blue Prism, and UiPath are the most popular among the RPA tools currently on the market, but their use entails high investment cost for the companies. Each tool has strengths and weaknesses that would need to be studied closely to determine which one would best meet the company requirements.

3.3.2 Selenium WebDriver: an alternative to RPA tools

Selenium WebDriver was developed with the aim of automating tests for web applications. This automation testing tool can open a browser to automate and perform actions, then verify the web application does what it is supposed to do.

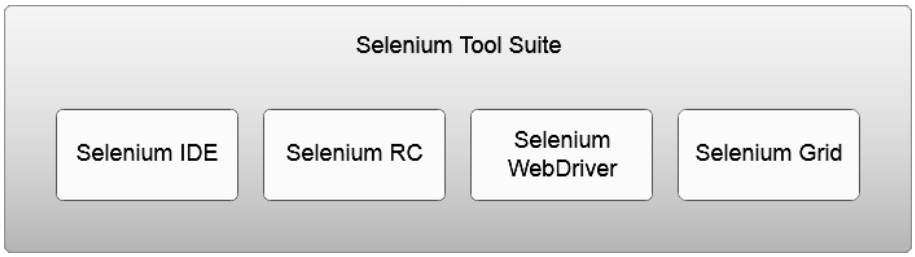


Figure 6: Selenium Tool Suite

Table 5: Selenium Tool Suite comparison

	Selenium IDE	Selenium RC	Selenium WebDriver	Selenium Grid
Browser	• Google Chrome • Mozilla Firefox	• Internet Explorer • Google Chrome • Mozilla Firefox • Opera • Safari	• Internet Explorer • Google Chrome • Mozilla Firefox • Opera • Safari	• Internet Explorer • Google Chrome • Mozilla Firefox • Android • iPhone • Opera
OS	• Windows • Mac OS • Linux	• Windows • Mac OS • Linux	• Windows • Mac OS • Linux	• Windows • Mac OS • Linux
API for programming languages	No	• Java • JavaScript • Ruby • PHP • Python • Perl • C#	• Java • JavaScript • Ruby • PHP • Python • Perl • C#	• Java • JavaScript • Ruby • PHP • Python • Perl • C#
Record and play tests	Yes	No	No	No
Programming skills	Not required	Required	Required	Required

Selenium WebDriver is part of the Selenium open source set of tools for test automation in a web browser. In other words, using Selenium makes possible web applications testing, but not desktop applications testing. All Selenium projects are under Apache 2.0 licence. Selenium is therefore a free software that can be used for commercial purposes as long as the copyright notice is included. The Selenium tool suite includes Selenium IDE, Selenium 1 (or Selenium RC), Selenium 2 (or Selenium WebDriver), and Selenium Grid.

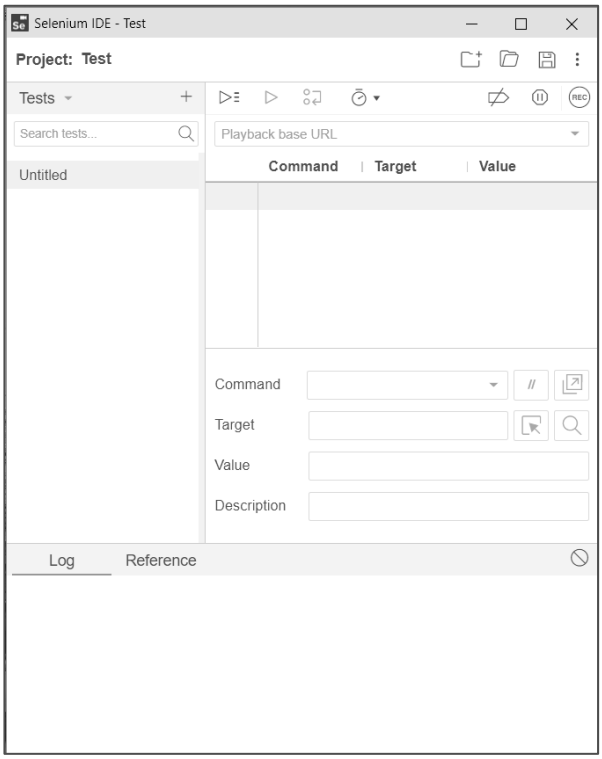


Figure 7: Graphical User Interface of Selenium IDE

As shown in Table 5, each tool can be used with, at least, Google Chrome and Mozilla Firefox, and are all compatible with Windows, Linux, and Mac OS. Selenium WebDriver is a new version of Selenium RC, with a different system architecture but similar or improved functionalities. In addition, Selenium WebDriver is more adapted to object-oriented programming. Then, Selenium Grid can launch simultaneous tests under different platforms and different browsers of different versions, by using either Selenium RC or Selenium WebDriver: in this way, time can be saved when running tests. Besides, APIs for different programming languages, including Java, are offered by Selenium RC, Selenium WebDriver, and Selenium Grid. Finally, the Selenium IDE tool (cf. Figure 7) aims at recording operations performed on the web pages of a browser in order to use them as test scenarios. It is a Firefox and Chrome add-on whose principal features are to record, edit, and debug tests. Test scenarios can therefore be created, as well as test sequences.

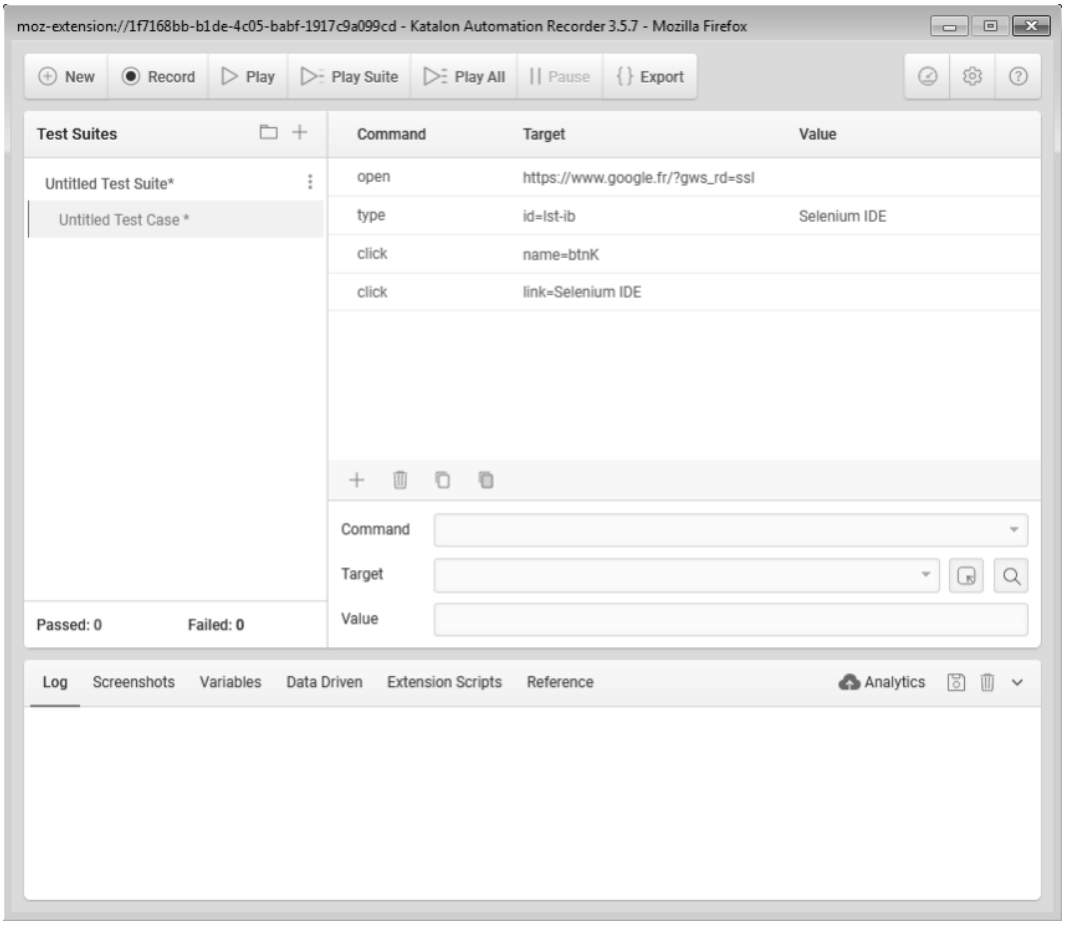


Figure 8: Graphical User Interface of Katalon Automation Recorder

In view of the fact that Selenium IDE does not allow the export of test scripts, Katalon Automation Recorder IDE can be used as an alternative. This free IDE, whose interface is displayed on Figure 8, is an add-on supported by Firefox and Chrome which enables one to perform similar actions to those supported by Selenium IDE. It has the additional functionality of exporting code in the desired programming language (e.g. Java) from the test sequences performed on the browser. Therefore, it can provide a first draft of the code of the test sequence to be written, which can be completed if needed.

A limitation of Selenium is that it only works with a browser. In other words, without the presence of an open browser, Selenese – Selenium commands – cannot be executed. Also, it is necessary to use a graphical user interface to perform these actions. Indeed, when an action, such as hovering the mouse to reach another element that is not visible without this hover, should be performed, the presence of a graphical user interface is a necessity. In order to avoid this type of action being disrupted, it is highly possible that it will be necessary to use a graphical user interface entirely dedicated to Selenium. As a result, to use Selenium as a robot, it is necessary to have a platform entirely dedicated to it so that it is not disrupted in its performance

of tasks. Besides, in the specific context of the use of RPA in a web browser, one of the main disadvantages that can be encountered is the presence of NoCAPTCHA reCAPTCHA. The principle of CAPTCHA is to differentiate a computer from a human through an automated Turing test. It is therefore difficult to pass through the NoCAPTCHA reCAPTCHA with Selenium.

In general, in the scope of RPA, Selenium would be used on web pages whose code has been written by third parties. It must therefore be kept in mind that the code of this website may change over time, due to maintenance or modification of the graphical user interface. It may sometimes be necessary to search for elements not by their identifier, but by using the DOM[3] or other attributes of web elements. In this case, the modification of the DOM can then have significant consequences for the robotization of the process, making it impossible to find these elements and thus to move to the next step. Similarly, the website used must not be undergoing maintenance, or the execution of the robotic process must not take place during it.

One of the main advantages of Selenium is its price. It is a free and open source set of tools. Therefore, regardless of the number of developers using Selenium, the cost will remain zero, and therefore unchanged. Besides, Selenium is compatible with most browsers – including Mozilla Firefox, Internet Explorer, Google Chrome, Safari, and Opera – and environments – Windows, Mac OS, and Linux.

Since Selenium's robot can open a browser to automate and perform actions, it has been considered to use Selenium as an alternative to RPA tools for automating actions on web pages. Indeed, by recording the actions performed using Katalon Automation Recorder IDE, the code can be generated, then used with Selenium WebDriver in the source code. Then, the code can be modified or improved. However, instead of being used at the test level, as is usually done when using Selenium for automation web testing, the code will be written in the source code.

[3] See Glossary

4 CASE STUDY: A SYSTEM FOR TERMINATING INSURANCE POLICIES

With the French Hamon law's entry into force, in January 2015, insurance companies became aware that the number of people asking for terminating their insurance would significantly increase. According to the insurance company asking for the RPA system, the number of hours spent by the employees on terminating policies had grown as the amount of incoming notices of termination increased. The reason is that because of legal pressure, these notices must be processed without delay. As a result, the time spent on doing this became progressively more significant, thereby pushing high added value tasks into the background. So, people were hired for the simple exclusive purpose of processing these notices, thus leading to a cost rise for the company. Desirous of reducing these costs and saving time, the insurance company wished for a system that could terminate the policies automatically. Therefore, Atos met this demand by suggesting the use of an RPA system that would use the already existing internal resources of the company's IT system.

4.1 OBJECTIVE OF THE PROJECT

The main objective of the system is to reproduce the interactions usually done by the employees on the already existing internal-use software of the insurance company. Consequently, the role of the implemented robot is to process the notices by using what currently exists. Specifically, the implemented functionalities should not be business functionalities so that they did not interfere with those implemented by the software used at present. For instance, the exception cases concerning policy termination must be handled by already present software.

To summarise, the RPA system is developed with the aim of using the internal-use software of the company but not replacing it. Besides, the way the robot operates should be identical to what is currently done to terminate a policy.

4.2 REQUIREMENTS ANALYSIS SUMMARY

Four stakeholders were identified during the requirements analysis. First, developers and maintainers must operate the system while complying with the functional and non-functional specifications that have been required. The employees of the insurance company who usually terminate the contracts are the persons benefiting indirectly – since they do not have to interact directly – from the system. The funds are provided by the insurance company, which therefore determines the budget for the implementation of this project, thus

making it the major stakeholder. Finally, the director and project managers of the insurance company's IT department provide the IT tools.

The main constraints imposed by the stakeholders are the implementation of the system with a restricted budget, since it is a POC, and the use of technologies already used by the client's IT department, in order to facilitate the integration of the system.

In a nutshell, the technologies used to design the software must be inexpensive and match those already used in the IT department. The system must be a software robot that can be launched several times a day and is able to read emails, extract information from a PDF file, and terminate a contract on the already existing intranet. In other words, the robot must be able to repeat the contract termination process described in the following subsection.

4.3 PROCESS DESCRIPTION

Once a day, the email box receiving the notices of termination is opened and the arrival of new emails is controlled. Cancelling a policy is the result of several actions performed by the employees in charge of terminations. The diagram presented on Figure 9 illustrates this process by detailing the steps that an employee goes through to perform the task.

Processing the notices of termination begins with their receipt. The termination requests are made by email: the letters are sent via an electronic registered letter (ERL) using a third party, called A in the diagram, in charge of the delivery. The ERL's are sent to a specific email address that is supposed to receive only termination requests. In fact, this email address is a folder shared by many email addresses belonging to different employees. It is dedicated to the reception of all the notices of termination, regardless of the type of the policy. Therefore, an ERL received can contain several letters and only some of them concern Hamon's law. When receiving the email from the third party, the link is clicked on and the process to download the ERL begins. Different third parties exist to deliver an ERL, and the process to go through in order to acknowledge the receipt of the electronic letter differs from one to another. Depending on the third party, a link or a password can be used to access and obtain the ERL, then afterwards it can be downloaded as many PDF files or as a zip file containing all the PDF files. In this case study, the third party chosen –called "A" – sends two emails. The first email contains a link leading to a web page where the terms and conditions (T&Cs) must be accepted.

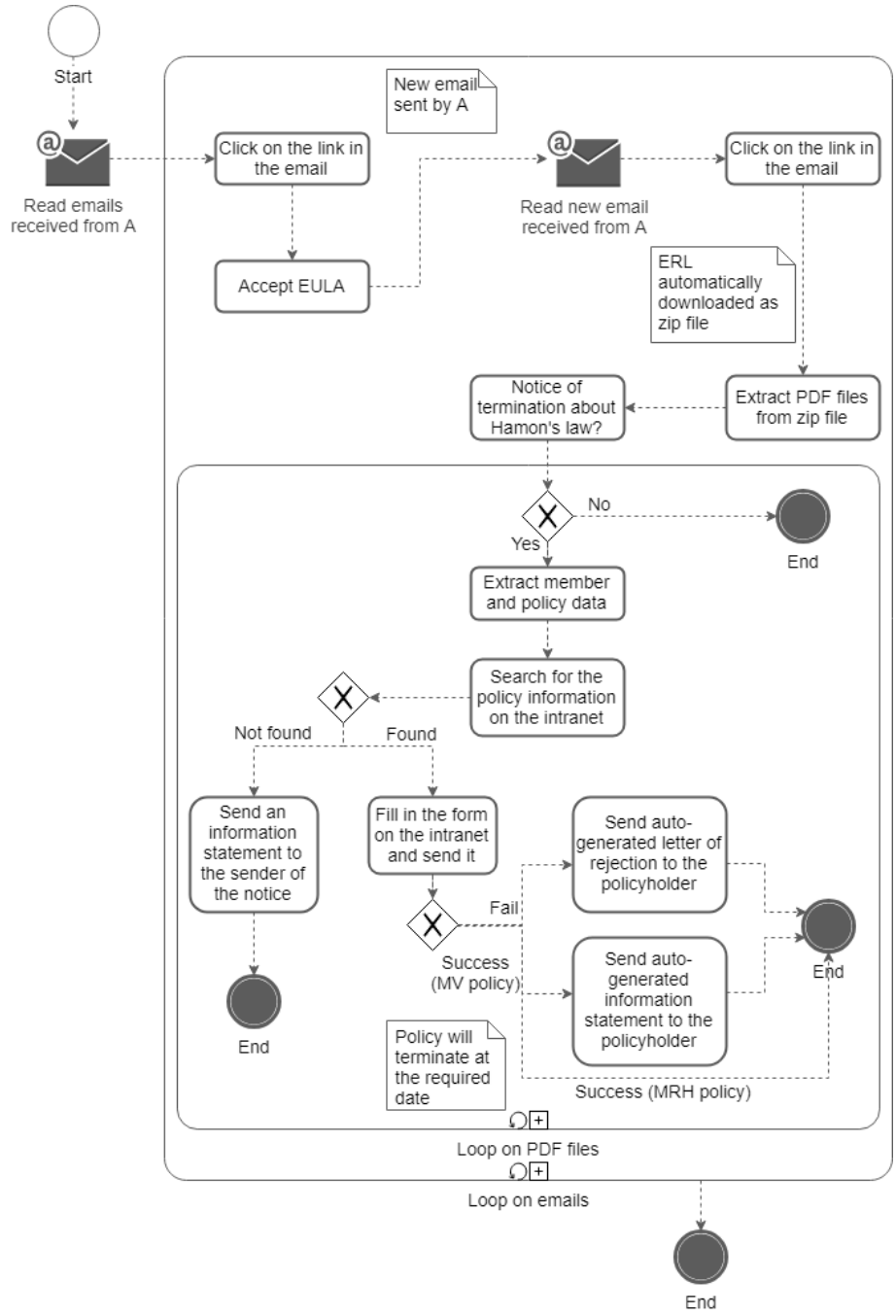


Figure 9: Diagram of the human process for terminating policies

The second email is received once the T&Cs have been accepted and it contains a link for instant download. In every case, every cancellation letter is linked to a proof of delivery. Once the ERL is downloaded, each letter is opened and processed. Focusing on the notices concerning Hamon's law, they should all contain the necessary information for the termination of the policy. In a first place, it must contain the name "Hamon" and the article number "L113-15-2". Data are extracted from the PDF file, such as the first and last names of the

policyholder, the contract number[4], the insurance policy number, and the type of policy that is to be terminated. Besides, the registration number of the vehicle can be found if the notice of termination is about a car insurance policy, and the address of the premises must be explicitly written if it concerns an MRH insurance policy. Then, with the extracted information, a search is performed on the intranet to find the corresponding policy. When it is found, the notice of cancellation and its proof of delivery are uploaded on the document management system (DMS) of the company, so it is persistent. Afterwards, the termination process on the intranet begins. The form is completed using the data extracted from the letter, then sent. The following step is to download the auto-generated information statement document, a letter rejecting the request or nothing – depending on the outcome of the termination – from the link that is given on the intranet. Finally, the downloaded document must be sent by mail to the policyholder.

[4] In French, the contract number is called "numéro de contrat". It differs from the "numéro de police" which is the actual insurance policy number. A contract has a number and consists of one or more policies, with a maximum of one active policy. The contract number is unique in its category (e.g. MRH or MV, etc.)

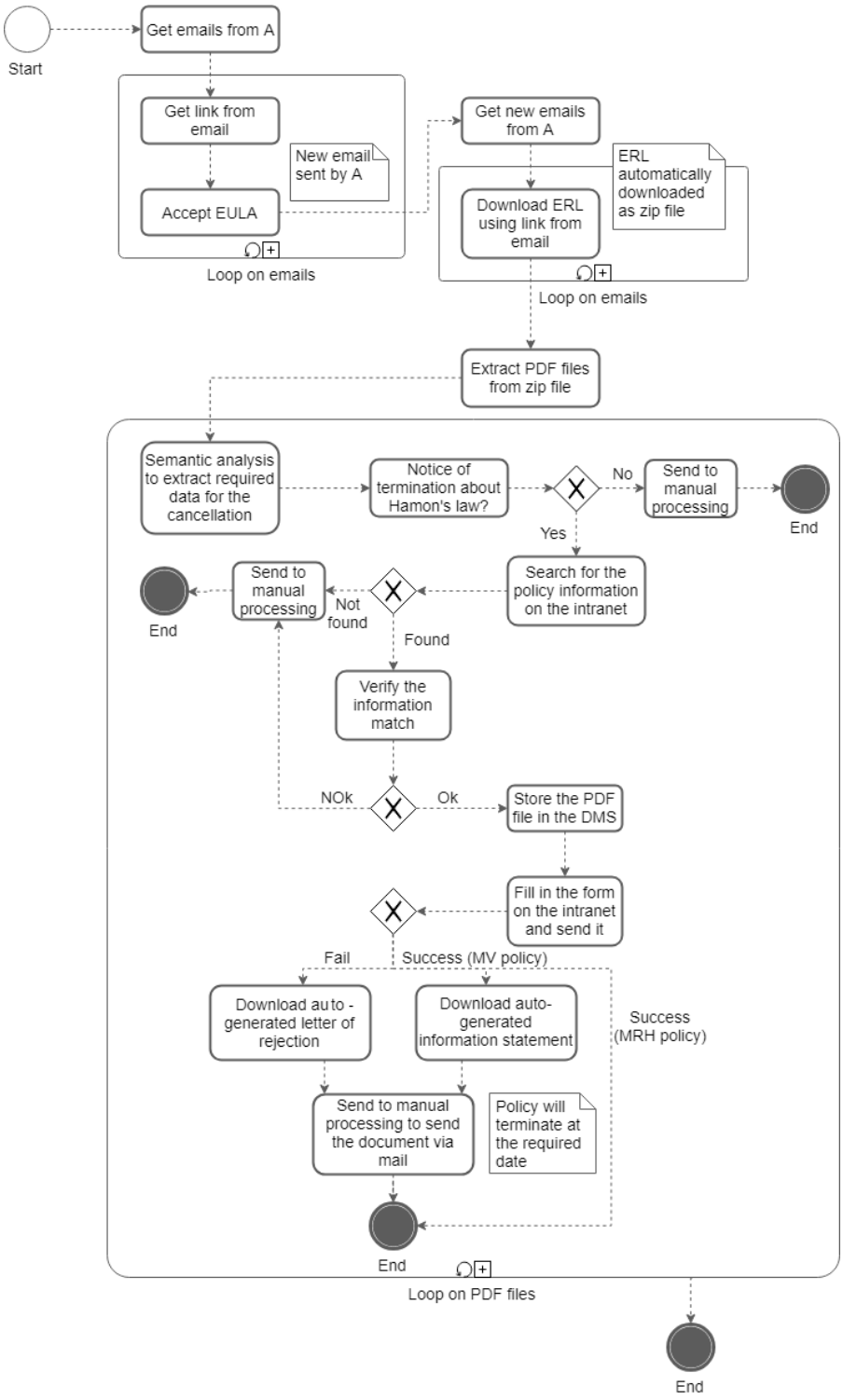


Figure 10: Diagram of the robot process for terminating policies

The process is slightly different for the robot, considering it can use tools such as web services. The main idea of the process followed by the robot is illustrated on Figure 10. Basically, as soon as the situation cannot be handled by the robot, the request is sent for manual processing. In these cases, employees themselves must intervene and process the notices of cancellation.

4.4 TECHNOLOGIES USED

In this section, a general overview of the technologies used and the links between them are given, then the reasons for the choice of each technology are detailed in the next subsections.

4.4.1 General overview

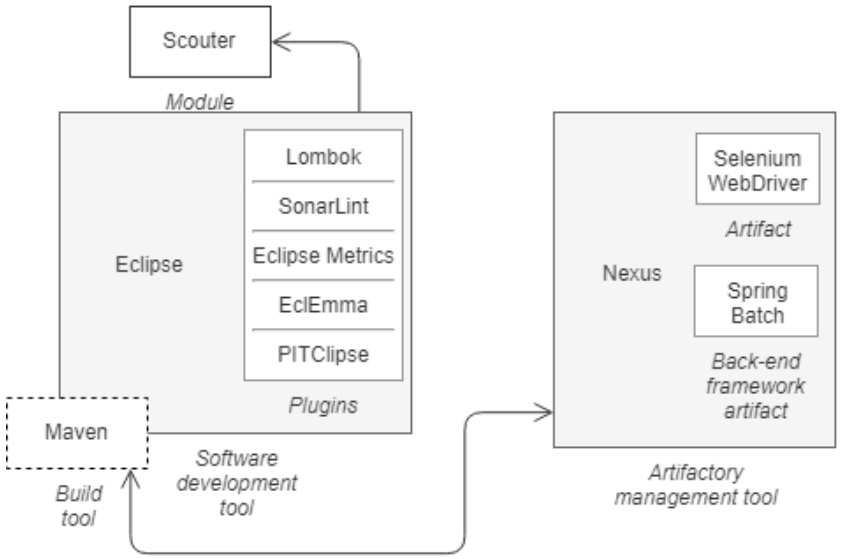


Figure 11: Technologies and tools used for the implementation

The robot is a batch process that is supposed to be executed several times a day. The programming language used to implement this solution was Java. While the Spring Batch framework was used to implement this batch, the role of Selenium WebDriver was to perform the task on the intranet of the client. Besides, a semantic analysis engine developed by Atos, called "Scouter", was used to analyse the letters. Other tools were used to help during the implementation of the batch to analyse code quality. The relations between all the used technologies are illustrated in Figure 11.

4.4.2 Maven

Maven is a build automation and dependency management tool used for Java, using POM[5] files for configurating projects. In this file, the libraries needed are specified as dependencies of the project. Therefore, using Maven for a project greatly facilitates the use of jar dependencies, as Maven handles their import. As the client owns a Nexus[6] repository manager, where all the dependencies along with the parent POM files needed for the project can be

[5] See Glossary

[6] See Glossary

found, it was decided to use Maven to implement the software solution. In this way, it is easier to integrate the project into the client's IT system.

4.4.3 Spring Batch framework

Spring Batch is a Java backend framework for batch implementation. This framework can be difficult to configure, but when it is mastered, it facilitates the implementation of batches written in Java. This open source solution is highly useful when batch processing is required, which is the case for this case study. In addition, its use makes batch maintenance easier as the concepts and principle are common between batches, thanks to the classes and interfaces around which the software is organised. Indeed, each batch consists of Job, Step, Chunk, or Tasklet, etc., which are therefore common elements that are found in batch applications implemented using Spring Batch. Besides, most of the batch solutions of the client are implemented using this framework, making it a good candidate for the implementation of the software solution.

4.4.4 Selenium WebDriver

The final step of the contract cancellation process is to perform the cancellation on the intranet, by using a browser. So, the robot must perform on the intranet the actions usually performed by an employee. Thus, several RPA tools can meet this need and be considered. Automation Anywhere, Blue Prism, and UiPath, described in section 3.3.2, all have the ability to automate tasks on a browser. However, their cost is rather high, and one of the main requirements is to use low-cost tools.

Selenium WebDriver, originally developed with the aim of testing web applications, has been chosen for automating the tasks on the web interfaces. Indeed, Selenium's primary role is to automate tests for web applications. However, since Selenium's robot can open a browser to automate and perform actions, it has been considered to use Selenium as an alternative RPA tool to automate actions on one or more web pages, from one or more browsers, usually performed by humans.

Within the range of tools offered by Selenium, Selenium WebDriver seems to be suitable enough for the RPA needs of this project. Selenium has certain limitations that are known and have been considered before launching the project. For instance, a computer with a GUI will be dedicated to this software system. Considering this project is a POC, this free tool capable of performing tasks on websites is sufficiently efficient for the actions that are to be done.

4.4.5 Scouter: semantic analysis engine

Scouter is a tool developed by Atos [26] that can analyse the semantics of a text, i.e. it is aimed at understanding the meaning of a text. It is a tool based on natural language processing[7] (NLP) with a wide range of functionalities, such as optical character recognition[8](OCR) analysis and data extraction. Since the software solution requested by the customer only requires the retrieval of predefined information (e.g. last name, first name, address, policy number), only the module of data extraction of Scouter was used.

4.4.6 Tools

In this section, the different tools used to improve and measure the software quality are described.

4.4.6.1 Lombok

Lombok is a free Java tool, aiming at reducing the number of lines of the source code. It can be added to a project as a Maven dependency, and enables, among other things, auto-generating getters, setters, as well as constructors, but also loggers. Indeed, this kind of code may seem repetitive to write, or may even extend the size of the classes. The functionalities of Lombok differ from the option of auto-generation of code offered by the IDE by using annotations: the code will be generated automatically when the project is built, but the code will not appear in the source code. In other words: getters and setters are invisible, but they exist and appear in the class outline; they are automatically named in accordance with the Java code conventions. Visually, this makes the source code more readable as the classes will be shorter.

4.4.6.2 SonarLint

SonarLint is a tool developed by SonarSource, aiming at helping the developer when coding, so (s)he can write better code. It is a free and open source IDE extension which underlines the code quality defects that it has detected. Besides, the documentation furnished for a mistake is rich. The severity level of the mistake is given, so that developers remember what practices are the best when coding. It also gives examples of compliant and non-compliant code, with compliant code being code that complies with Java conventions and best practices. Installed from the Eclipse IDE marketplace, this tool has been used during the code implementation phase.

[7] See Glossary

[8] See Glossary

4.4.6.3 SonarQube

Developed by the same company as SonarLint, SonarQube is an open source tool that analyses the quality of the code more or less in detail, depending on the edition used (community or commercial). SonarQube supports several programming languages, including Java, and is used to analyse the code in order to report information on the level of documentation and comments, code duplication, potential bugs, compliance with programming rules, code complexity, and more. For example, SonarQube can also compute the technical debt, based on SQALE [27] [28], a language-independent generic source code evaluation method. More generally, SonarQube helps at determining the security and maintainability of the code. In the end, this tool was considered a good candidate for helping during the development of the solution, so the community edition was used when working on the source code of the project.

4.4.6.4 Eclipse Metrics

Eclipse Metrics is a plugin for the Eclipse development platform that calculates different metrics of a program written in Java. The McCabe's cyclomatic complexity of methods, the number of lines of code, and the number of lines of code per package, per class, and per method are metrics that are calculated by this plugin. The average and the maximum value of some of these metrics can be obtained as well. This plugin was used with the purpose of using the results as a complement to the results provided by SonarQube.

4.4.6.5 EclEmma and PITClipse

EclEmma and PITClipse are both free Eclipse plugins that are used to evaluate the quality of the tests of a Java project. While EclEmma evaluates only the code coverage by the unit tests of a Java project, PITClipse evaluates also the mutation coverage. The principle of mutation testing is to modify the code by creating mutations. If the code is changed, the results must be different, so the unit tests must fail. Therefore, mutations must be killed by the tests. If a mutation survives a test, it means that the unit tests are not good enough. So, as the code and mutation coverage by unit tests increases, the chances of project compilation failure after code modification are higher. Indeed, when the developer in charge of software maintenance alters the code, (s)he should not introduce new errors in the code. If a code is widely covered by the tests, the error will be easier to detect, thereby helping to maintain the software correctly.

In a nutshell, the software solution described in this thesis was implemented using Maven, as build automation tool, Spring Batch, for batch implementation, Selenium WebDriver, for performing tasks on the web browser, and Scouter, for semantic analysis of PDF documents. This implementation was assisted using tools such as SonarLint, SonarQube, EclEmma, and PITClipse to maintain the software quality. In the following, the architecture of the system is detailed.

5 IMPLEMENTATION OF A MAINTAINABLE SYSTEM

This section describes the implementation of the system that has been implemented to perform the contract termination by interacting between the different existing systems. First, the code rules to be followed are given. Subsequently, the specifications, architecture, and implementation of this system are described.

5.1 CODING RULES FOLLOWED

To implement the system and make it as maintainable as possible, some coding rules were followed. First, the given names in the code must be explicit, and the Java code conventions must be respected. Writing methods and classes that are too long must be avoided [9]: ideally, the size of a method should be less than twenty lines and the length of a class should be smaller than two hundred lines. If it is impossible, a method must be written in about forty lines, and the length of a class must be less than four hundred lines. Also, no more than two indentation levels – nested blocks – must be used in a method, and the code must be easily scannable by a human. Then, when a portion of code is used more than twice, a method must be created to gather the similar code and reduce the class or method length. Every class must be easy to test. With this aim in mind, the methods must implement as few functionalities as possible, ideally only one or two. Indeed, a class difficult to test probably does too many things. Last but not least, the code must be well documented and commented: Javadoc must be updated, and the comments must describe what the code does.

5.2 SPECIFICATIONS

The functional and technical specifications were defined by the client and Atos, then validated by both parties. They give a main idea of the software solution that needs to be implemented, and the constraints that need to be considered and respected.

5.2.1 Main idea of the software solution

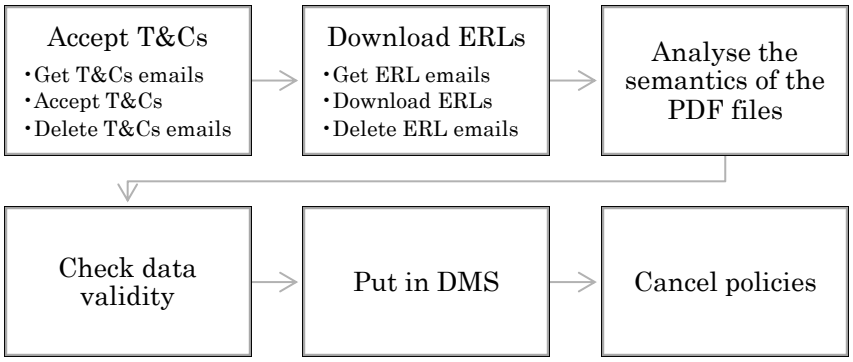


Figure 12: Main idea of the process

The software solution consists of six key phases, illustrated on Figure 12: validating the Terms and Conditions (T&Cs), downloading the electronic registered letter, extracting data from the PDF files, verifying the validity of the data, putting the termination request letters in the DMS, and finally cancelling the contract on the intranet.

First, emails from "no-reply@thirdpartyname.com" with a subject line containing "You are the recipient of a registered letter" must be retrieved. The link starting with the pattern: "https://www.thirdpartyname.com/receive/acceptTCs/" is extracted from it, then used by Selenium for the validation of the T&Cs.

Next, emails from "no-reply@thirdpartyname.com" whose subject contains "Download - Registered letter" must be retrieved. The link starting with the pattern "https://www.thirdpartyname.com/receive/download/" is extracted from the email body, then directly used to download the electronic registered letter.

Then, each PDF file corresponding to a cancellation request letter is analysed by the semantic analysis engine. The key data are extracted from it: the motive for termination (Hamon or not), the first and last names of the policyholder, the address, the address of the risk location, the vehicle registration, the type of policy to be terminated, the number of the policy to be terminated, and the name of the new insurance company of the client. These data are then stored to be reused in the rest of the process.

Following this analysis, the extracted data are checked. To do this, a RESTful[9] web service allowing the recovery of valid contracts and developed by the client is used. The following pieces of information of the policy returned by the web service are verified one by one: last name, first name, and address.

Once the verification is complete, the termination request letter and its proof of delivery are put into the DMS using another web service.

Finally, the last step is to perform the policy cancellation on the intranet using Selenium. A refusal letter must be downloaded if the termination is not possible. If the policy to cancel was an MV policy and the termination was successful, then an information statement must be downloaded. If a document has been downloaded, it will be sent to the policyholder by an insurer.

Throughout these phases, reporting files are generated or completed, so that insurers can be informed of the policies that have been terminated, as well

[9] See Glossary

as of the cases that could not be handled and that they must manage by themselves.

5.2.2 Constraints and solutions

The sources from which the constraints come vary from one project to another, but in any software project, constraints come from the tools, software, or environment that are used. Nevertheless, since this project includes a company and its client, additional constraints are imposed by these two parties. Thus, these factors impose constraints for which alternatives to the initial solutions had to be found.

Firstly, the use of Selenium requires that the computer on which the batch will be run must have a graphical user interface and be entirely dedicated to this batch in order to prevent any possible interference. It also implies that the batch application should not be started several times at the same time, and two instances must never overlap in order to avoid any interruption. Therefore, when the client will schedule the launching times for the batch application, the time between each launch should be much longer than the average batch execution time. Secondly, the non-use of a database was a constraint imposed by the client for the implementation of the POC. This caused a problem for data persistence during the batch execution. Therefore, alternative solutions had to be used to transmit data: the data required for the rest of the process was either stored in the context of the batch application or serialised in flat files. Then, since the project was only a POC, it was decided not to make this project multi-modular. As a result, only Scouter was used as a module, and the rest of the code was divided into packages. Finally, the creation of a .jar dependency, containing all the Selenium commands, external to the main project was imposed by Atos. Consequently, this implied the creation of a second Java project to be deployed so that it could be used as a library.

5.3 SYSTEM ARCHITECTURE

The system architecture is based on the functional and non-functional specifications and constraints that were defined. Thus, in order to determine the structure of a project, it is necessary to understand what the links between the different elements composing the system are, and the process that will be implemented.

5.3.1 General system architecture

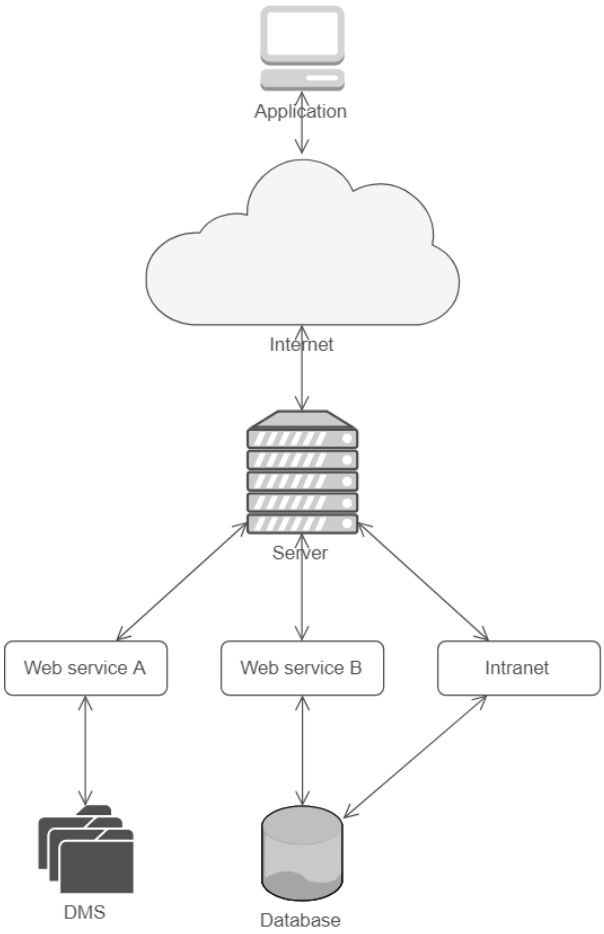


Figure 13: Architecture of the system

The system architecture is illustrated on Figure 13. The main components of this architecture are the batch application, Internet, the server, two web services, the intranet, and the company's database and DMS. The database can be accessed either via the intranet or by using web service B, while access to DMS is provided only through web service A. Thus, in order for the application to have access to the DMS and the data of the database, the application needs to use Internet to have access to the server on which these web services are hosted.

5.3.2 General process

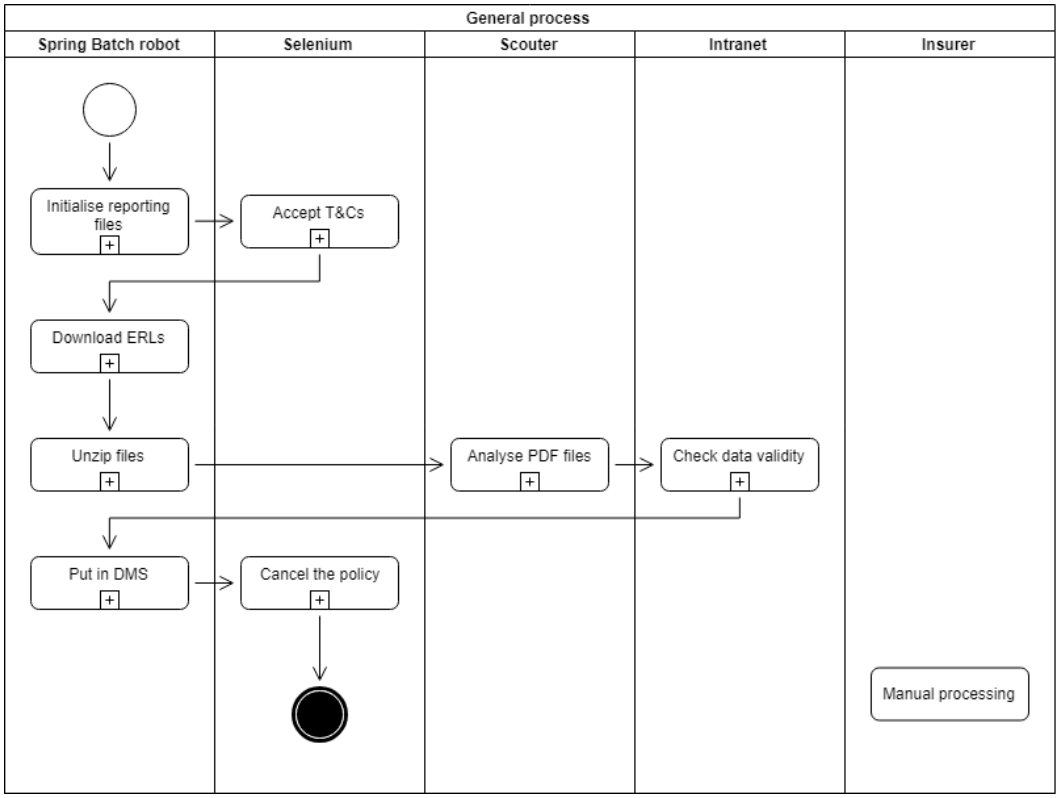


Figure 14: BPMN of the general process

The general process as explained in section 5.2.1 is represented by the BPMN diagram on Figure 14. This diagram illustrates the different entities participating in the process: the batch application (Spring Batch robot), Selenium WebDriver, Scouter, the intranet, and the insurer. There are six distinct functional phases, and every entity of this software solution is in charge of one or several of these subprocesses.

When the batch application is launched, the acceptance of the T&Cs is performed by Selenium. After that, the batch downloads the electronic registered letters. The PDF analysis is then performed by Scouter, followed by the checking of the validity of the data through the intranet. The letters of cancellation request are next put into the DMS by the batch, then Selenium performs the termination of the policy. However, it can happen that the batch contains an unhandled exception. When the batch is not able to handle a case, the cancellation letter is sent to the insurer for manual processing.

In order to implement this process, a code structure had to be determined and is detailed in the following section.

5.3.3 General structure

To implement this system and ensure that the project is as maintainable as possible, it was decided to structure the batch so that it could be reused for other similar projects. Therefore, the structure of the batch application has been chosen so that it can be easily modified.

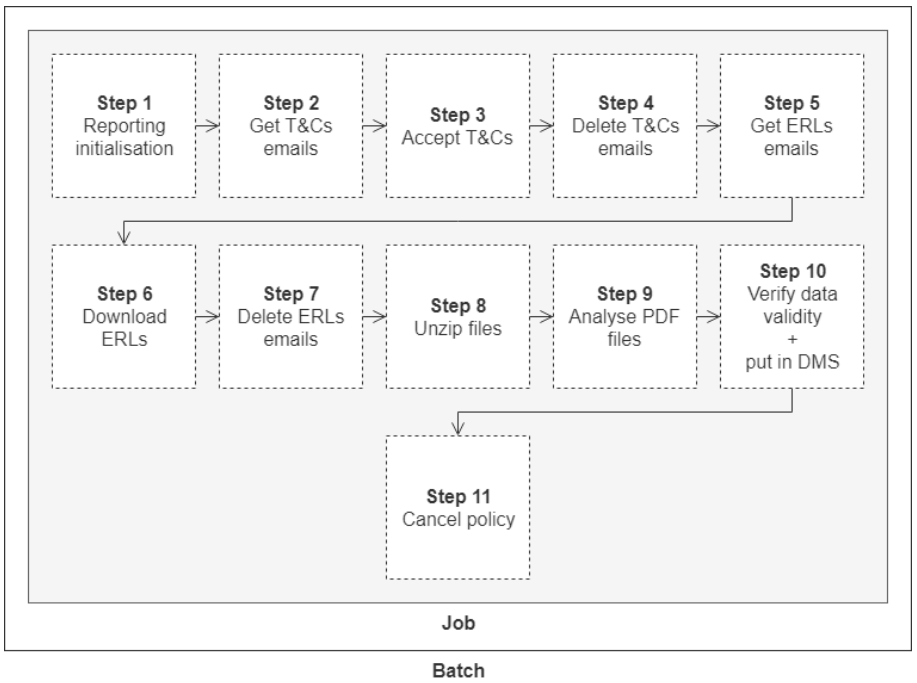


Figure 15: Job and Steps of the batch

Since the data were stored in the context so that they could be used throughout the batch, it was necessary to create a single Job. Thus, this Job was divided into several Steps. In total, eleven Steps result from the partitioning, each with its own well-defined task. Figure 15 illustrates the splitting of the Job into Steps: after initialising the files for reporting (Step 1), the T&Cs emails are obtained (Step 2) to enable the acceptance of the T&Cs (Step 3) afterwards. These emails are then deleted (Step 4). Next, the ERL emails are retrieved (Step 5). This is followed by the downloading of the ERL's (Step 6), and the deletion of the processed ERL emails (Step 7). Subsequently, the downloaded files are unzipped (Step 8) so that the PDF files they contain are accessible for the PDF analysis (Step 9). The validity of the extracted data is then verified, and the pdf files are put into DMS (Step 10). Finally, the termination on the intranet is performed (Step 11).

In addition, due to the absence of a database, processed PDF files are moved from directory to directory after each Step (cf. Appendix I and Appendix II). In this way, it makes it possible for the batch to resume, for each file, at the

Step at which the batch could have stopped previously without creating interference or having to start over from the beginning.

Although the batch structure is an essential part of the project implementation, the package organisation is equally important as it plays a major role in improving the reusability of the project.

5.3.4 Package overview

The project consists of two Maven projects: a batch as the main Maven project, and a subsidiary Maven project for the RPA part performed by Selenium. Therefore, each project has its own packages.

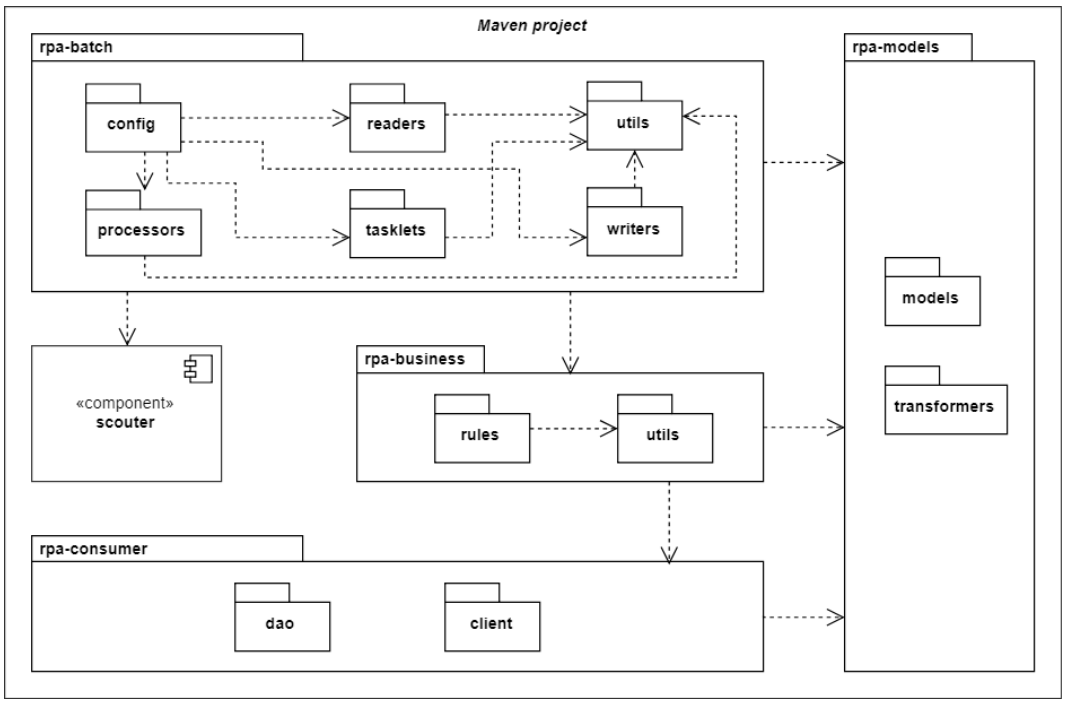


Figure 16: Package diagram of the main project

The main project was organised into different packages and contains the Scouter module (cf. Figure 16). There are four main packages: *rpa-models*, *rpa-batch*, *rpa-business* and *rpa-dao*. The *rpa-models* package contains, in its *models* package, all the model objects used for the project, and, in its *transformers* package, the objects used for transforming an object into another object. All the other packages depend on the *rpa-models* package. The *rpa-consumer* package corresponds to the data access layer and therefore contains the classes used for accessing the email box or the database containing the contracts. The classes implementing the business rules are in the *rpa-business* package, depending itself on the *rpa-consumer* package. Finally, the *rpa-batch* package corresponds to the presentation layer and contains all the classes necessary to build the batch and uses the *rpa-business* package. The *config*

package contains all the configuration classes where the Beans are declared. The package depends on the *readers*, *processors*, *writers*, and *tasklet* packages which contain all the classes needed to configure the different Steps of the batch.

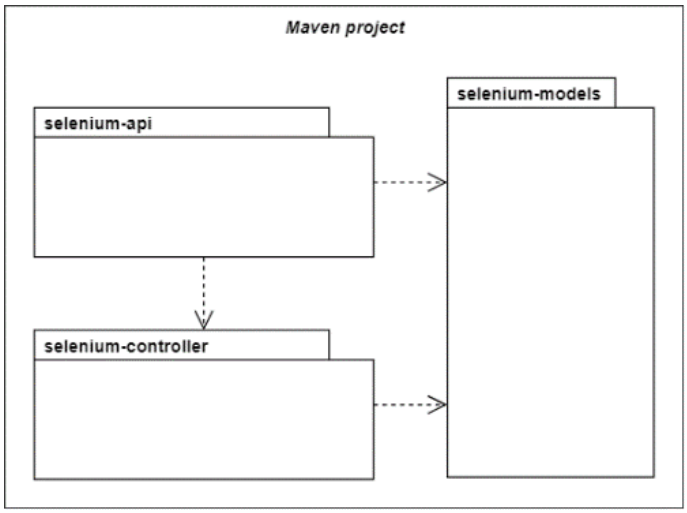


Figure 17: Package diagram of the Selenium project

The subsidiary project is used as a dependency by the main project and it consists of three packages (cf. Figure 17). The *selenium-controller* package is the controller layer of the project. It contains the classes implementing the different actions that can be performed on the intranet. The *selenium-api* package, which corresponds to the *view* layer of the project, depends on the *selenium-controller* package and contains all the classes directly used by projects using this one as a library. In the end, these two packages both depend on the *selenium-models* package, which contains the model classes.

5.4 IMPLEMENTATION OF THE FUNCTIONALITIES

The implementation of the six distinct functionalities was divided in many technical Steps. Thus, one functional phase does not necessarily correspond to one technical Step. The functionalities, along with how they were implemented, are described in this section.

5.4.1 Initialisation of the reporting files

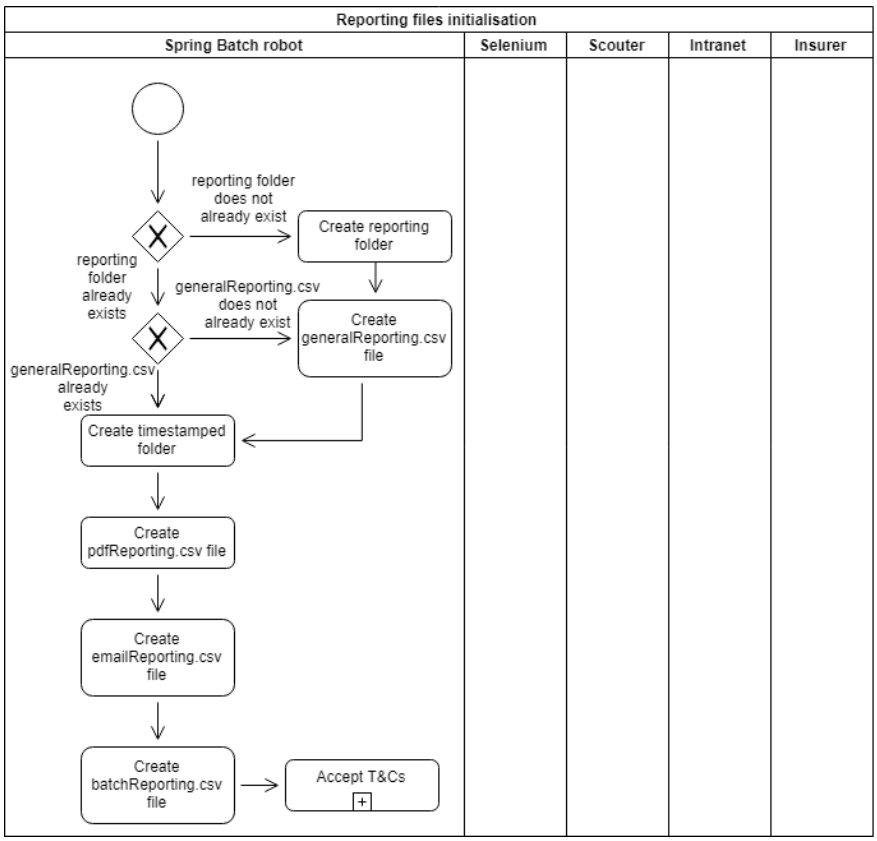


Figure 18: BPMN of the reporting files initialisation

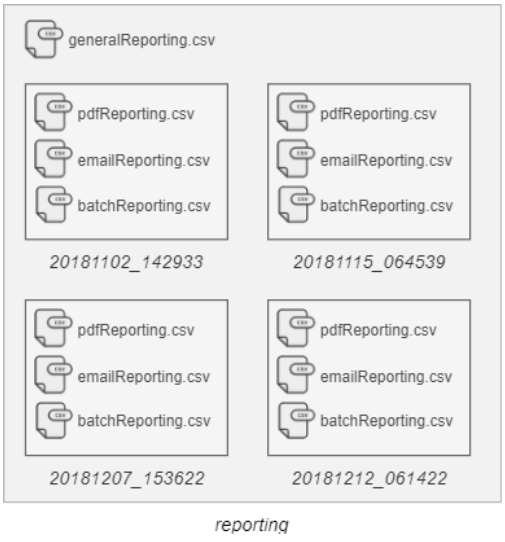


Figure 19: Example of a reporting folder

Before implementing any of the main functionalities of this application, the first Step of the batch is to initialise the reporting files. It consists of creating a reporting folder named “reporting” if it does not already exist, and a folder with a name based on its creation date, using the date format “yyyyMMdd_HHmmss”, in the “reporting” directory. Then, a general reporting csv file – “generalReporting.csv” – is created if it does not already exist in the “reporting folder, and reporting csv files – “pdfReporting.csv”, “emailReporting.csv” and “batchReporting.csv” – are created in the folder named by timestamp. Throughout the batch, new lines are added to the csv files. Figure 18 illustrates the process of initialisation of these files, and Figure 19 gives an example of what can be obtained.

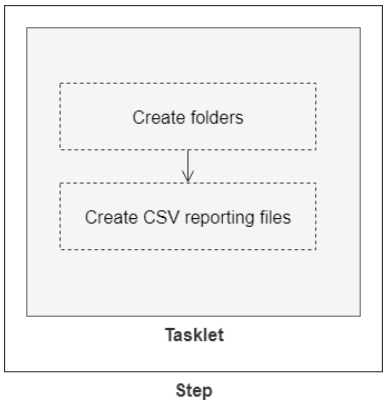


Figure 20: Step of the reporting files initialisation

In the end, initialising reporting files consists of a simple sequence of tasks to create folders and files (cf. Figure 20). For this reason, a Step composed of a simple Tasklet has been implemented, in which folders are first created, then csv files.

5.4.2 Acceptance of the Terms and Conditions

The first functional phase is the acceptance of the T&Cs detailed on Figure 21. An attempt is made by the batch application to connect to the email box receiving termination requests. If the connection fails, then the batch application stops, and an error is logged. When the connection is established, emails from the third-party sender whose subject indicates a T&Cs email are retrieved and stored in a list. The connection to the mailbox is then closed. For each email in the list, the presence of a link is verified. If no link is present, the email cannot be processed and is therefore deleted from the list. On the other hand, if at least one link is present, any link that corresponds to the link pattern leading to the validation of the T&Cs is extracted from the body of the email. If there is only one single link, it is stored with the corresponding email for the rest of the process.

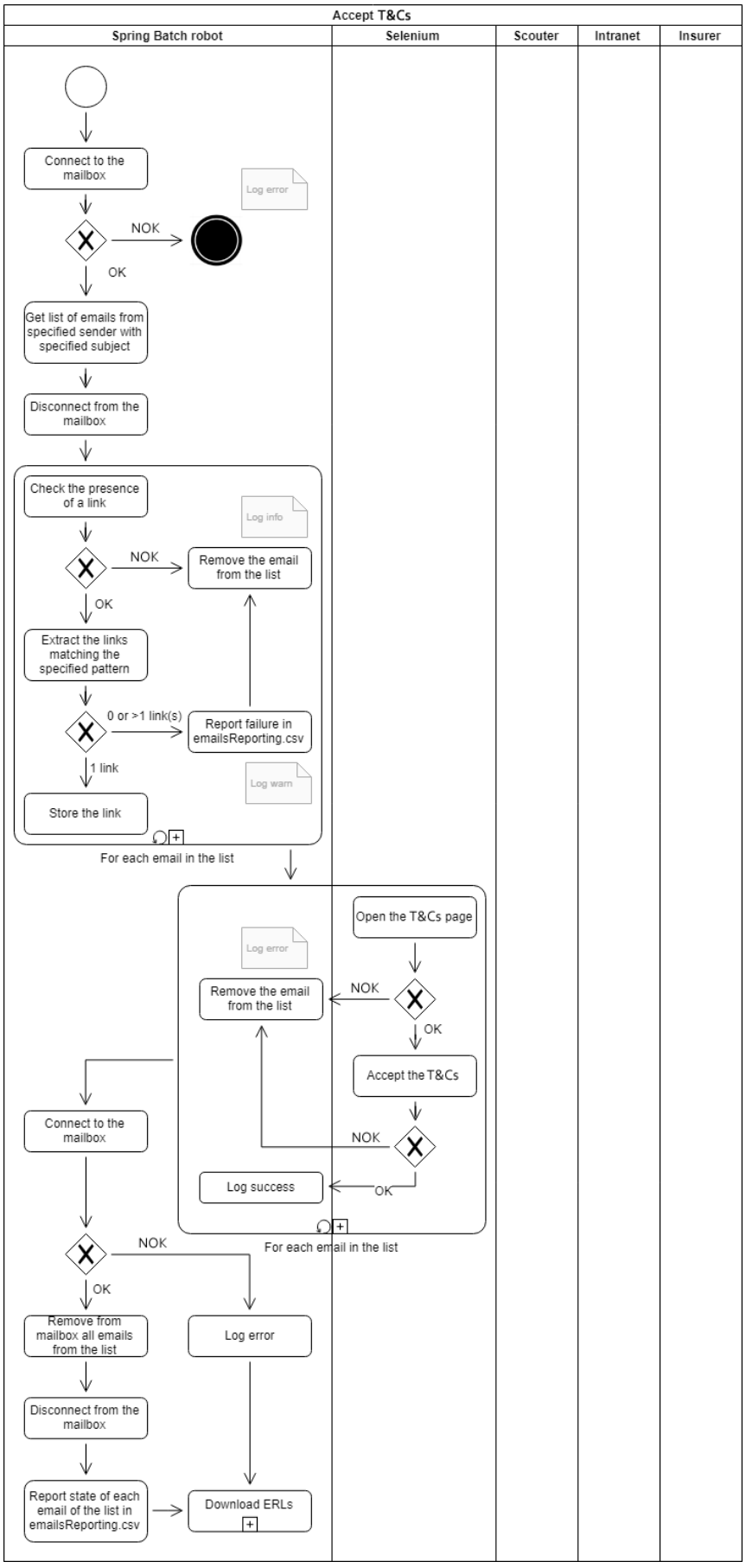


Figure 21: BPMN of the process of acceptance of T&Cs

Again, if several links result from this extraction, it is impossible for the application to determine which link should be used. The email is therefore impossible to handle and is removed from the list. A new loop is then run for the remaining emails in the list. The page of acceptance of the T&Cs is opened using Selenium WebDriver. If the displayed page is the one expected, the T&Cs are accepted, next the success of this operation is logged. If the page fails to open, the link leads to an unexpected page, or the acceptance of the T&Cs fails, the email is no longer considered manageable by the batch and is therefore deleted from the list. Once every email has been processed, a connection to the mailbox is made, so that the emails for which it was possible to validate the T&Cs – in other words, all the emails remaining in the list – are deleted. After closing the connection to the mailbox, the status of each email is given in the email reporting file with an informative message. If the connection to the mailbox failed, the emails cannot be deleted: an error message is therefore logged to inform of the situation. The electronic registered letters can then be downloaded.

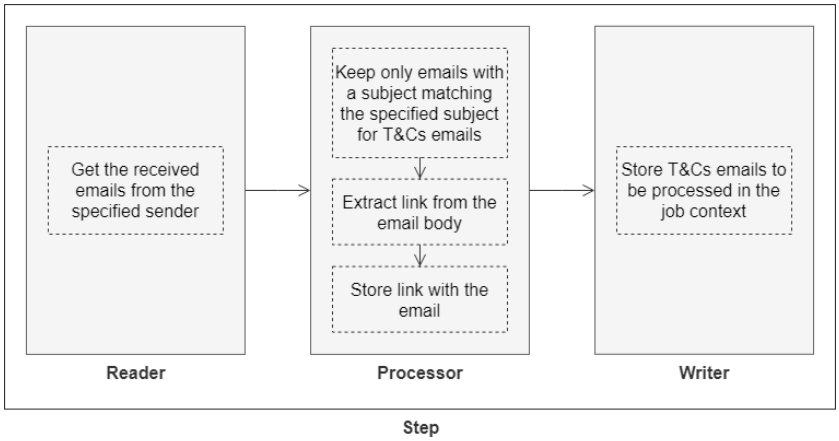


Figure 22: Step for obtaining the T&Cs emails

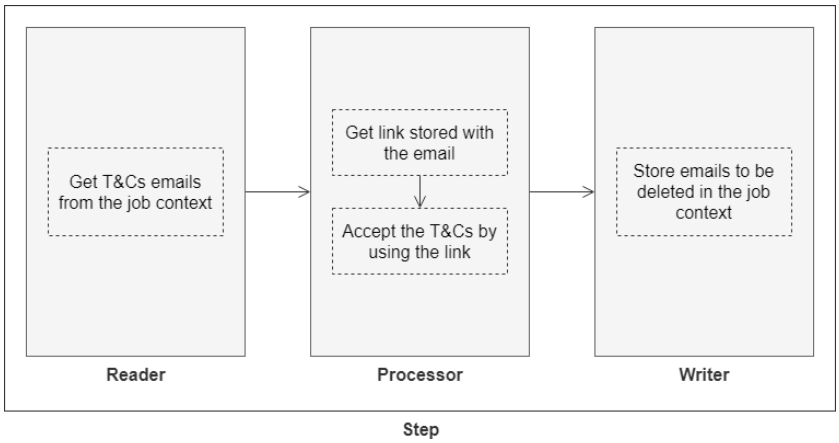


Figure 23: Step for accepting the T&Cs

Figure 24: Step for deleting the T&Cs emails

The implementation of this functional phase was divided into three Steps: a first Step for obtaining the emails (cf. Figure 22), a second Step in charge of accepting the T&Cs (cf. Figure 23), and finally a third Step performing the deletion of processed emails (cf. Figure 24).

The Step for obtaining emails sent by the third-party has been implemented with the use of a reader, a processor, and a writer. The emails are retrieved in the reader, then the processor keeps only the emails concerned with the acceptance of the T&Cs. It extracts the links from these emails, and stores them with the emails, which are finally stored in the job context by the writer.

After that, the acceptance of the Terms and Conditions begins with the retrieval by the reader of the emails stored in the job context. The processor gets the link, which is then used to accept the T&Cs. Finally, the emails to be deleted after the acceptance of the T&Cs are stored in the job context.

Then, the deletion of these emails is handled by a last Step, which is a Tasklet. It retrieves the emails to be deleted from the job context, deletes them, and informs about the status of each email – whether it could be deleted or not – in the csv email reporting file. Afterwards, the Steps for downloading the ERL's are executed.

5.4.3 Download of the ERL's

The second functional phase consists in downloading the electronic registered letters (cf. Figure 25) and is very similar to the previous phase. After getting a list of emails from the appropriate sender with the appropriate subject, and extracting the link from the email body, the batch application proceeds to the download of the ERL's. Next, the emails that have been processed and for which the download of the ERL's was successful are deleted.

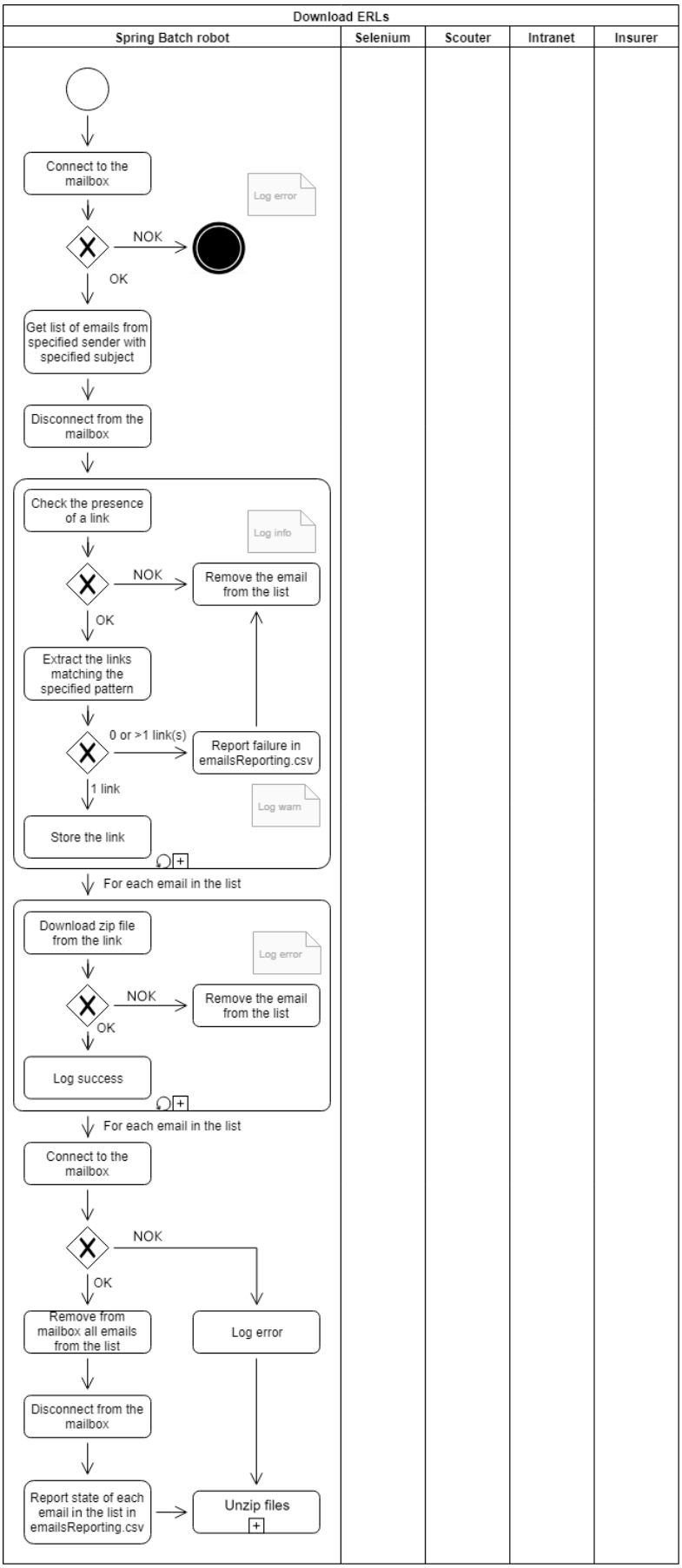


Figure 25: BPMN of the process of download of ERL's

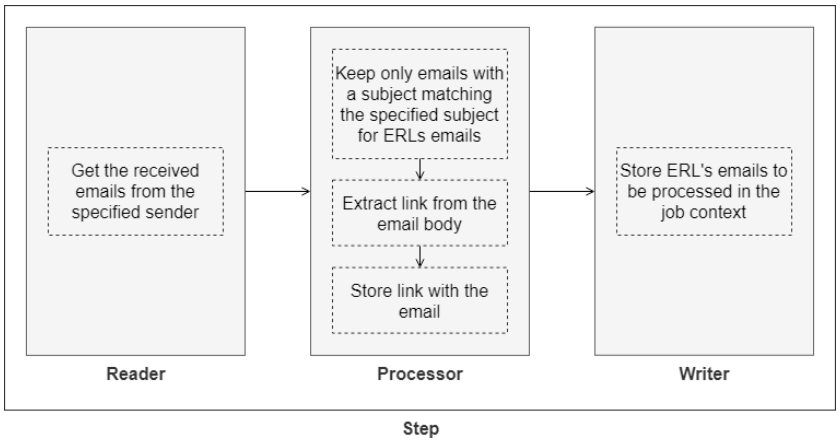


Figure 26: Step for obtaining the ERL emails

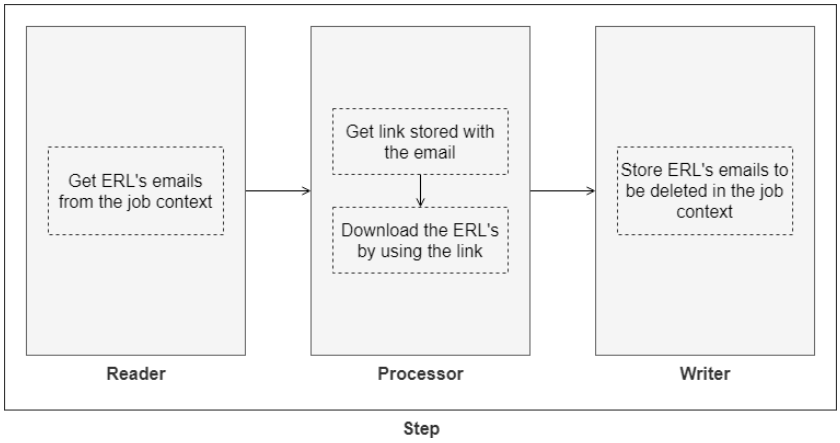


Figure 27: Step for downloading the ERL's

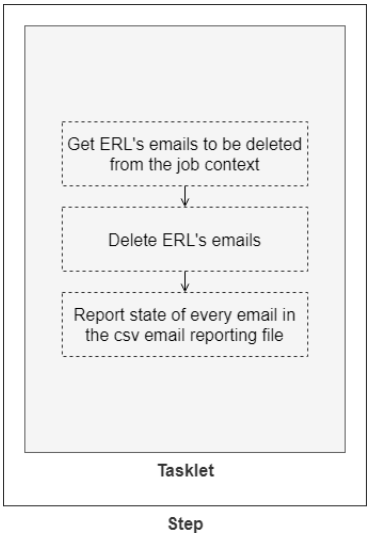


Figure 28: Step for deleting the ERL emails

As the previous functional phase, this phase was divided into three: a first Step for retrieving the emails (cf. Figure 26), a second Step for downloading the ERL's (cf. Figure 27), and finally a third Step for deleting the processed emails (cf. Figure 28). Thus, the implementation of the Steps of email retrieval and

email deletion are almost identical, while the download Step uses the link to directly download the file before proceeding to the unzipping of the downloaded files.

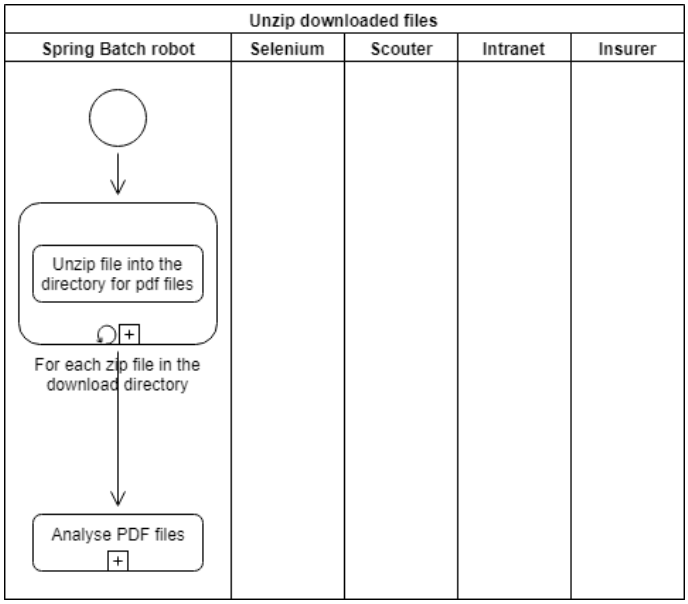


Figure 29: BPMN of the process of file unzipping

As illustrated in Figure 29, each zip file in the download directory is unzipped, and the PDF files are extracted and sent directly to the storage directory for the PDF files.

Figure 30: Step for unzipping downloaded files

Considering unzipping the previously downloaded files (cf. Figure 30) consists of a simple task, it was performed by using a Tasklet for the Step. After unzipping the files, those that were successfully unzipped are deleted. Afterwards, the following phase is the analysis of the extracted PDF files.

5.4.4 Analysis of PDF files

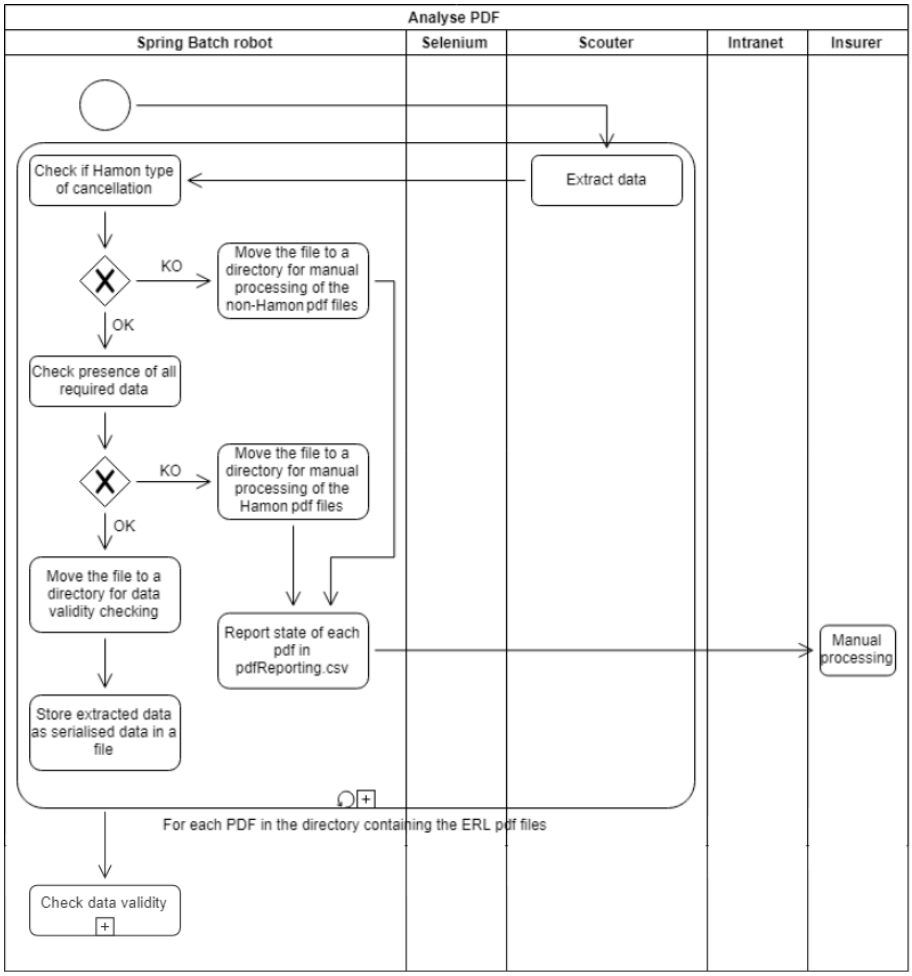


Figure 31: BPMN of the process of analysing PDF files

The process for analysing PDF files is illustrated on Figure 31. For every pdf file in the folder containing files that were previously extracted and whose name does not end with "_deliveryProof", Scouter extracts the following data: last name and first name of the policyholder, contract number, policy number, policy type, address, and license plate. Besides, "Hamon" or the name of the article of law "L113-15-2" must necessarily be mentioned in the request, and the last name and first name of the policyholder must be present. If the letter concerns a Hamon termination request and all the necessary data are present, the pdf file can be moved to the directory used by the next Step: checking of data validity. The data collected during the analysis of the file is serialised in order to be stored in a flat file with the pdf file and proof of delivery. If the analysis reveals that it is not a Hamon cancellation request, then the file is moved to a specific directory for manual processing of non-Hamon cancellation request letters. Also, if all the necessary data is not present, the file is moved to the directory for manual processing for Hamon cancellation request letters.

When a letter is sent to manual processing, it is reported in the csv reporting file of pdf files.

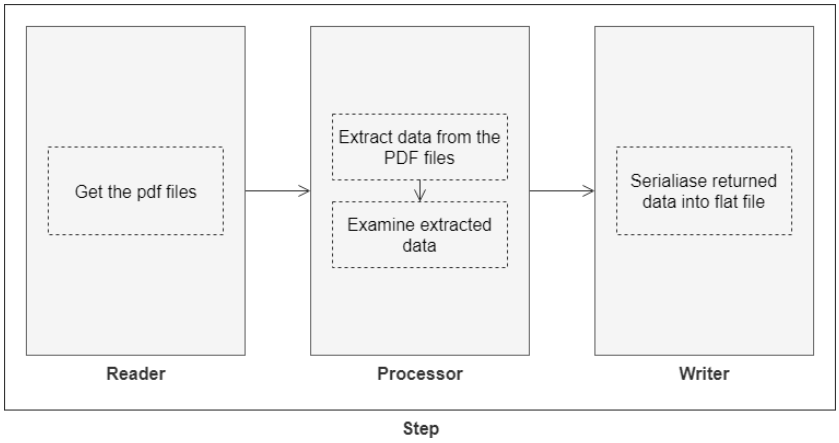


Figure 32: Step for analysing PDF files

This Step has been implemented using a reader, a processor, and a writer (cf. Figure 32). The PDF files are retrieved in the reader. Subsequently, in the processor, the files are analysed, and examined: a non-Hamon termination request is sent to non-Hamon manual processing, a Hamon termination request for which data is missing is sent to Hamon manual processing, and the other requests are moved to a folder to be processed in the next Steps of the batch. Finally, the data that have been extracted are serialised into a flat file by the writer.

Afterwards, the validity of the data that have been extracted can be verified.

5.4.5 Checking data validity and putting in DMS

After every file has been analysed, and data has been extracted from them, these data are used to retrieve a contract. This contract is used afterward to verify that the data are correct. The process at this phase is illustrated by the diagram on Figure 33. Checking the validity of the data begins by reading the serialised data from the file created after the analysis of the PDF files. The following data are required to retrieve a contract: contract number, policy number, contract type, and license plate if it is an MV contract. These data are used by the web service so that it can perform a search which returns a list of zero, one, or more contracts. If zero or more than one contracts are returned, then the case is considered unmanageable by the robot, and is therefore sent for manual processing. If only one contract is returned, then the policyholder's first and last names are verified. After that, if the cancellation request concerns an MRH contract, the address of the location of risk is checked. In the event that any of the data required for the search via the web service is missing or

incorrect, either the search will return zero insurance contracts, or the verification of the data will fail.

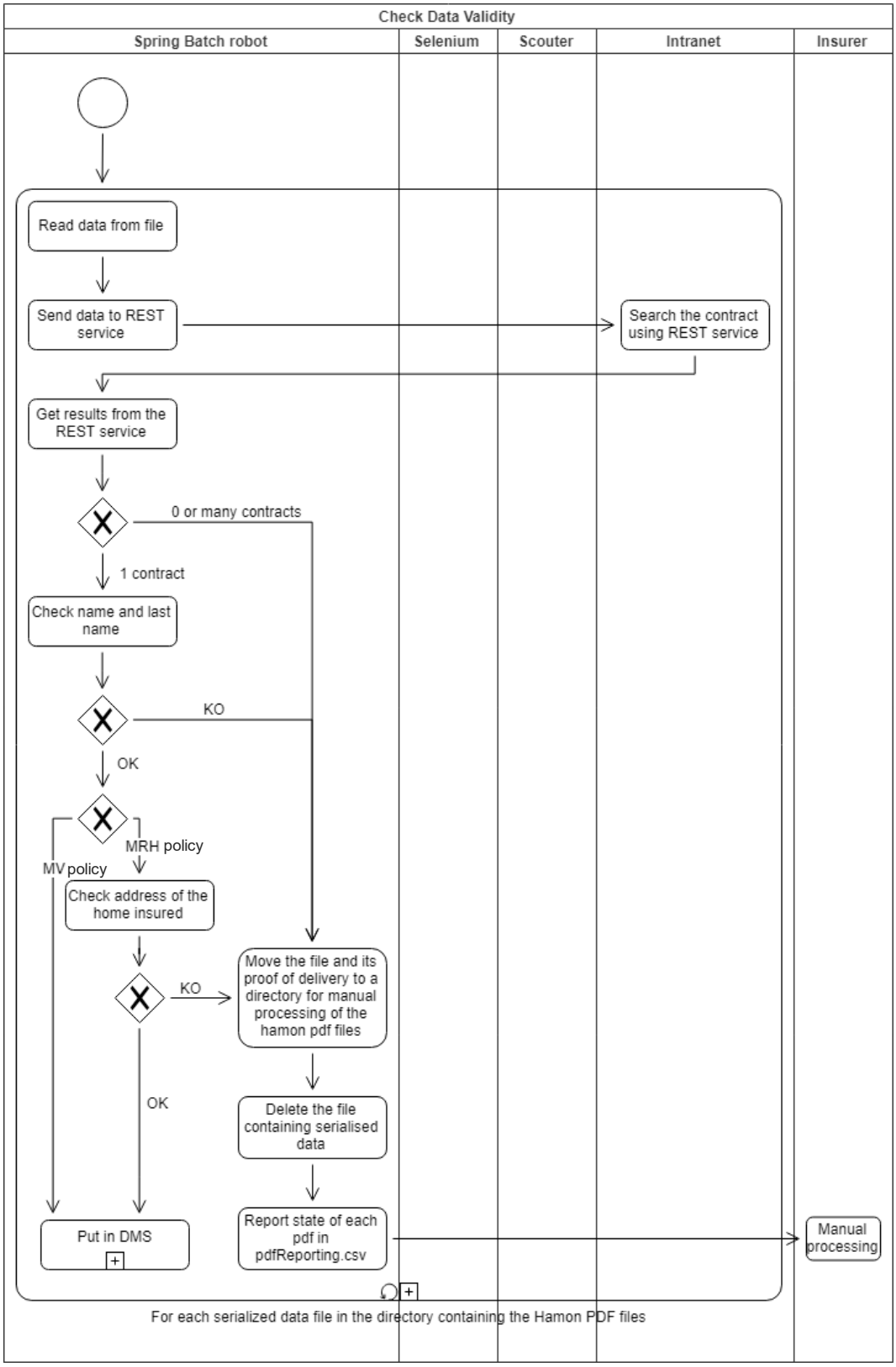


Figure 33: BPMN of the process of checking data validity

In all cases, the letter will be sent for manual processing, as it will be if the last name, first name, or address are incorrect. When a termination request letter is sent for manual processing, it is moved with its proof of delivery to the folder intended for the retrieval of files for manual processing. The file containing the serialised data associated with the letter is deleted. Once the data has been verified, the letter is sent to the DMS.

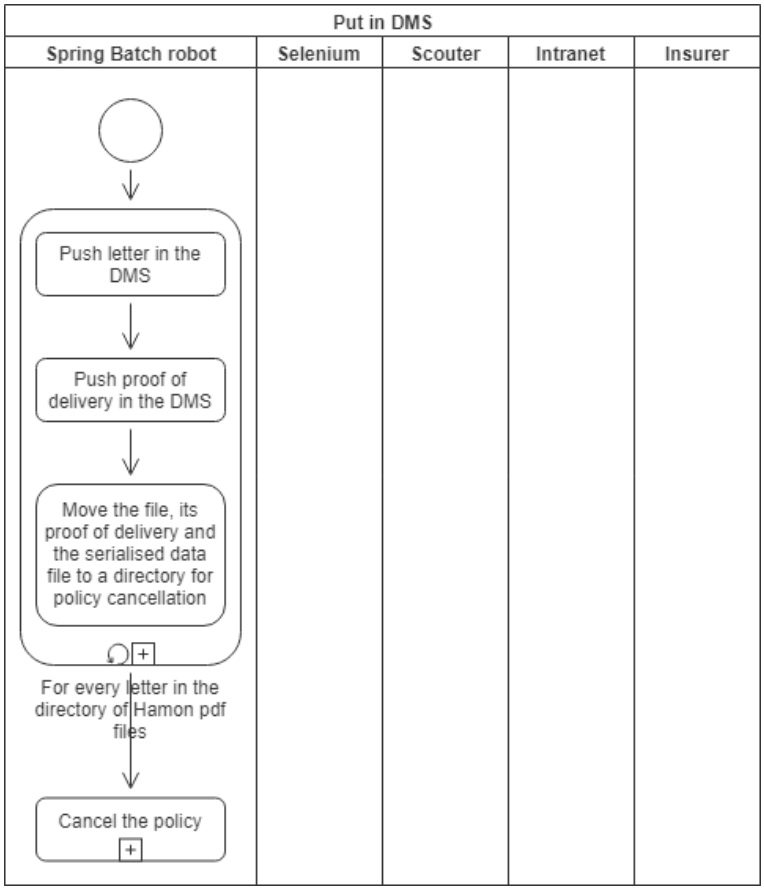


Figure 34: BPMN of the process of putting documents in DMS

If the data are valid, then the letter is pushed into DMS, as well as its proof of delivery. The files are also moved to the directory containing the pdf files that have already been put in the DMS, so that they can be retrieved for the cancellation of the policy. This process is illustrated on Figure 34.

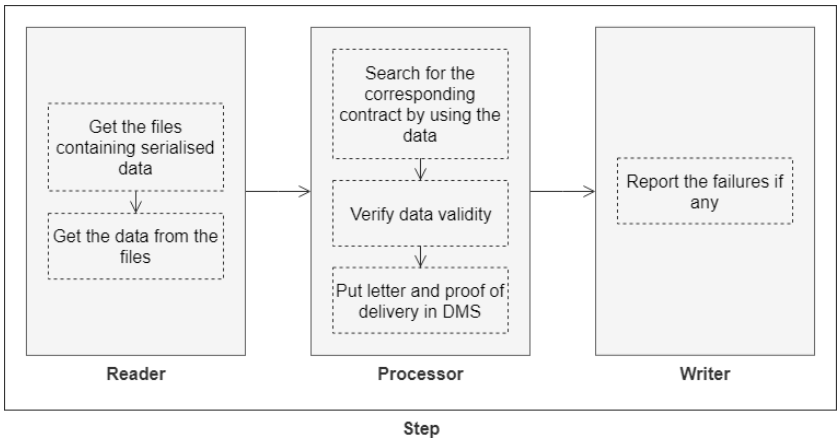


Figure 35: Step for putting documents in DMS

The verification of the validity of the data as well as the process for putting documents into the DMS have been implemented in a single step consisting of a reader, a processor, and a writer (cf. Figure 35). The reader retrieves the files containing the serialised data, then extract the data from the file. Then, the processor uses these data to retrieve the corresponding policy, verify the data, and finally put in the DMS the letter and proof of deposit linked to these data. Finally, the writer reports failures if there have been any.

The process of cancelling insurance policies can then start.

5.4.6 Cancellation of the policies

The cancellation of the policy is mostly performed by Selenium. For each file containing serialised data, the cancellation process described on Figure 36 is performed. Thus, the Selenium driver opens the web browser and accesses the intranet login page. The credentials that have been provided to the batch application are used to login to the intranet. Then, the contract cancellation page is accessed. The Hamon topic is selected, and the form is completed based on the data that was collected during the analysis of the letter, then submitted. If the cancellation fails, then a rejection letter is downloaded, the browser is closed, and the letter and its proof of deposit are moved to the directory for manual processing. If the cancellation is successful, the browser is closed, and the letter and its proof of deposit are moved to the directory containing letters that have been successfully processed. An information statement document is downloaded before closing the browser if the policy was an MV policy. In every case, the file containing the serialised data is deleted. After all the files are processed, the batch application stops running.

Appendix III to Appendix VI are mock-ups representing the cancellation page and the different pages that can appear after submitting the form.

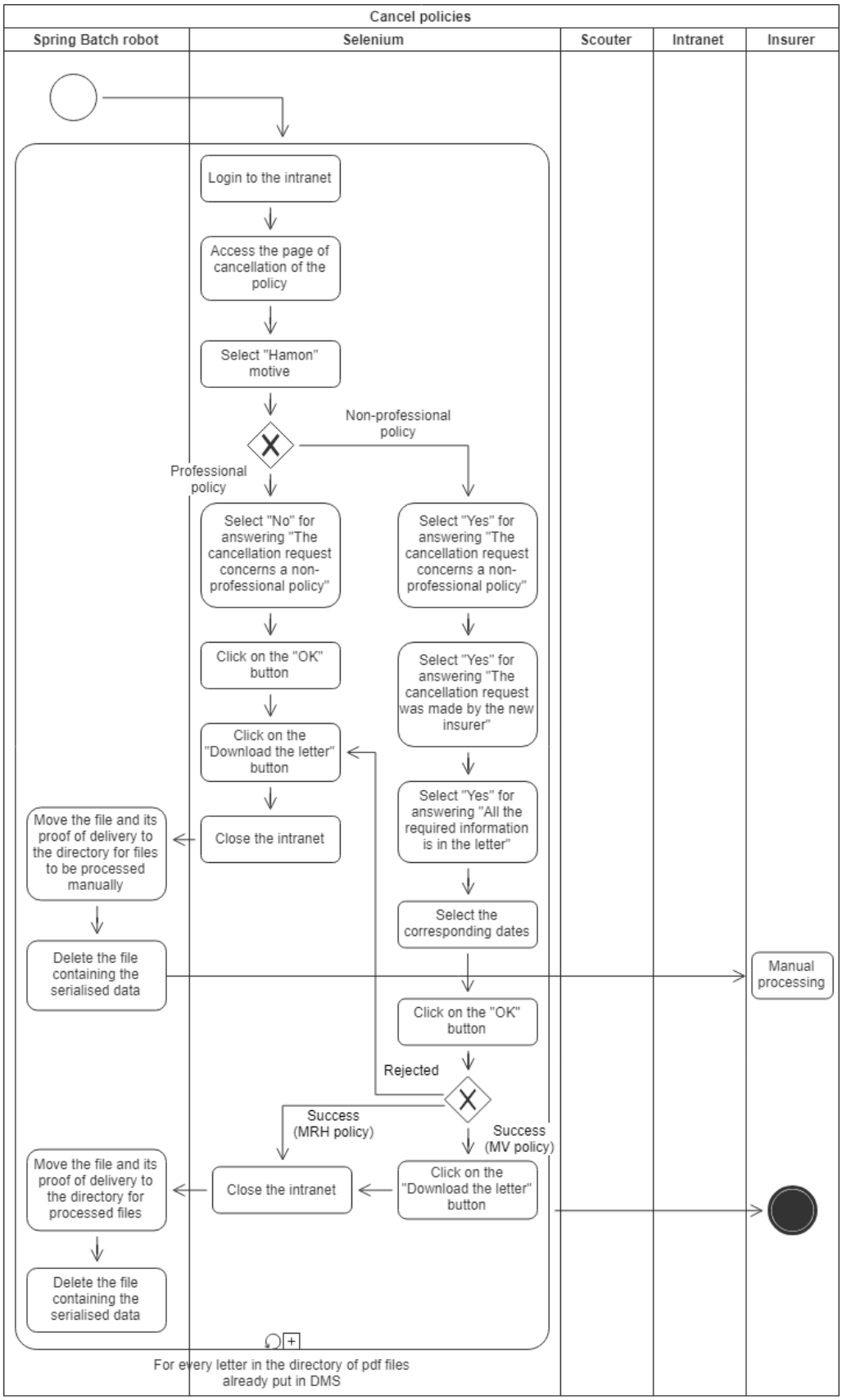


Figure 36: BPMN of the process of cancelling policies

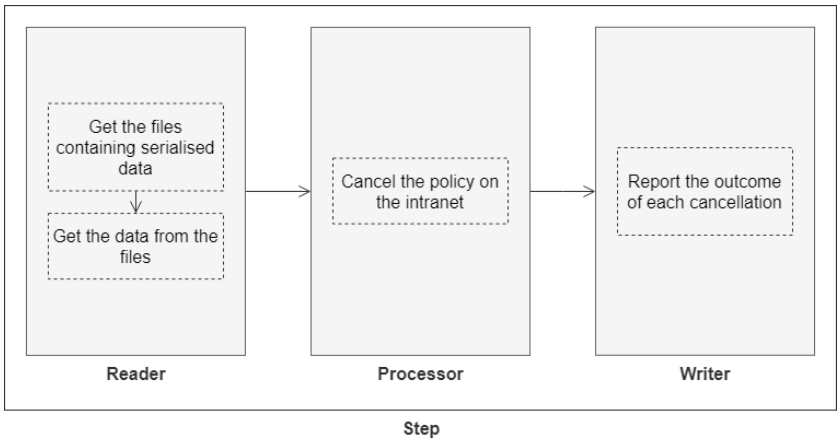


Figure 37: Step for cancelling policies

The Step for the cancellation of policies is illustrated on Figure 37. The reader retrieves the files containing the serialised data, and then deserialises the data from this file. By using these data, the processor performs the cancellation on the intranet. Finally, the writer reports the outcome of each cancellation.

This Step is the last Step executed by the batch application. After this, there should be files left only in the following directories: directory of non-Hamon cancellation requests, directory of non-handled Hamon cancellation requests, and directory of handled Hamon cancellation requests. Also, the reporting csv files contain the report of the outcomes of the processing of every pdf file that was processed by the batch.

6 RESULTS AND ANALYSIS

Following the implementation of this project, an analysis was performed in order to determine the maintainability of the global project. The two parts of the project were analysed separately: on the one hand, the batch project, and on the other hand, the Selenium project which was used as a jar dependency. Even though it was used as a module in the main project, the code of Scouter was not analysed, since it had been developed before this thesis. SonarQube and Eclipse Metrics were the tools used to analyse the code.

6.1 CODE SIZE, COMMENTS, AND NESTED BLOCK DEPTH

Data such as the size of the source code, the proportion of comments in the code, and the nested block depth of the methods were collected. So, the average number of logical lines of code, as well as the proportion of comments, per class and per method of each project, were collected and reported in tables and histograms for the purpose of analysing the distribution of each metric in the code.

Table 6: Number of classes and methods – main project

Package	Number of classes	Number of methods	Mean number of methods per class
Batch	56	221	4
business	12	37	3
consumer	5	33	7
models	58	319	6
TOTAL	**131**	**610**	**4.65**

Table 7: Logical lines of code and class logical lines of code – main project

Package	Number of lines of code	Mean number of lines of code per class	Max number of lines of code per class
Batch	3,021	54	150
business	590	54	123
consumer	398	80	216
models	1,377	24	69
TOTAL	**5,386**	**41**	**216**

As shown in Table 6, the main project, i.e. the batch application, is divided into four packages, and has 131 classes and 610 methods. Table 7 shows that the *batch* package has the highest number of LLOC's. This is twice the number of LLOC's in the *models* package, which has the second largest number of LLOC's and the lowest mean number of LLOC's per class. While the *consumer*

package has the lowest number of LLOC's, the mean number of LLOC's per class is the highest. In total, the project has 5,386 LLOC's, the average number of LLOC's per class is 41, and the maximum number of LLOC's per class reached is 216.

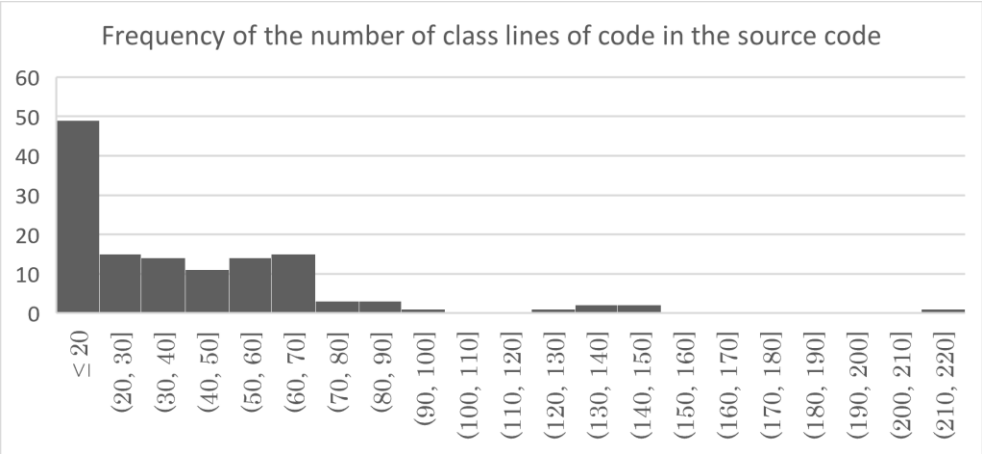


Figure 38: Histogram of the number of class lines of code – main project

As illustrated by the histogram in Figure 38, most classes have between 20 and 70 LLOC's, and a slightly smaller number of classes of the source code has less than 20 LLOC's. Very few classes have more than 150 LLOC's.

Table 8: Method logical lines of code – main project

Package	Number of method lines of code	Mean number of lines of code per method	Max number of lines of code per method
batch	1,251	6	37
business	306	8	20
consumer	181	5	32
models	517	2	38
TOTAL	2,255	4	38

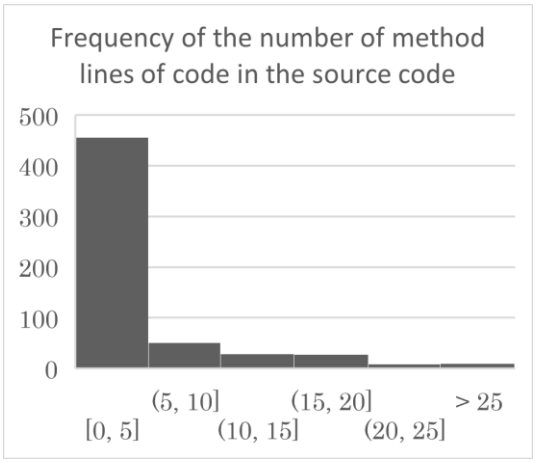


Figure 39: Histogram of the number of method lines of code – main project

Out of 5,386 lines of code, 2,255 are method LLOC's. As shown in Table 8, a method contains on average four LLOC's and the maximum number of LLOC's reached in a method is 38. In addition, as shown by the histogram in Figure 39, a large majority of the methods have less than five LLOC's, and only a very small number of the methods have more than 20 LLOC's.

Table 9: Lines of comments – main project

Package	Total number of lines of comments	Proportion of lines of comments (%)
batch	933	23.6
business	267	31.2
consumer	143	26.4
models	805	36.9
TOTAL	2,148	29.5

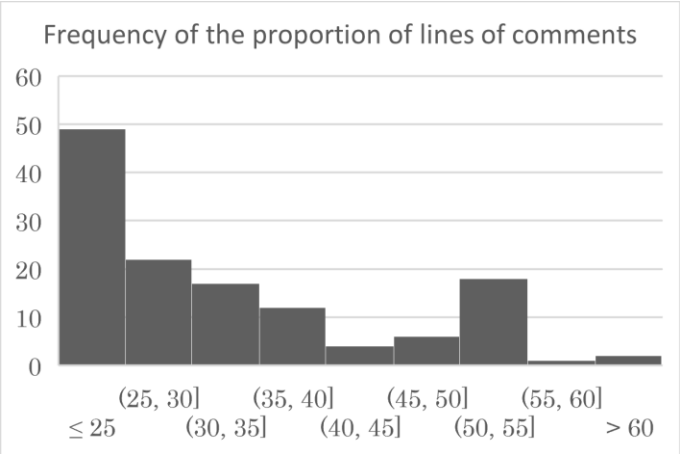


Figure 40: Histogram of the proportion of lines of comment per class – main project

The source code contains 2,148 lines of comments (CLOC's), which represents 29.5% of the non-blank physical lines of code (LLOC's added to CLOC's) of the whole project (see Table 9). In addition, as shown in Figure 40, more than half of the project classes contain more than 25% CLOC's.

In total, the main project contains 10,100 lines of code (LOC's), 5,386 LLOC's, 2,148 CLOC's and therefore 2,566 NCLOC's.

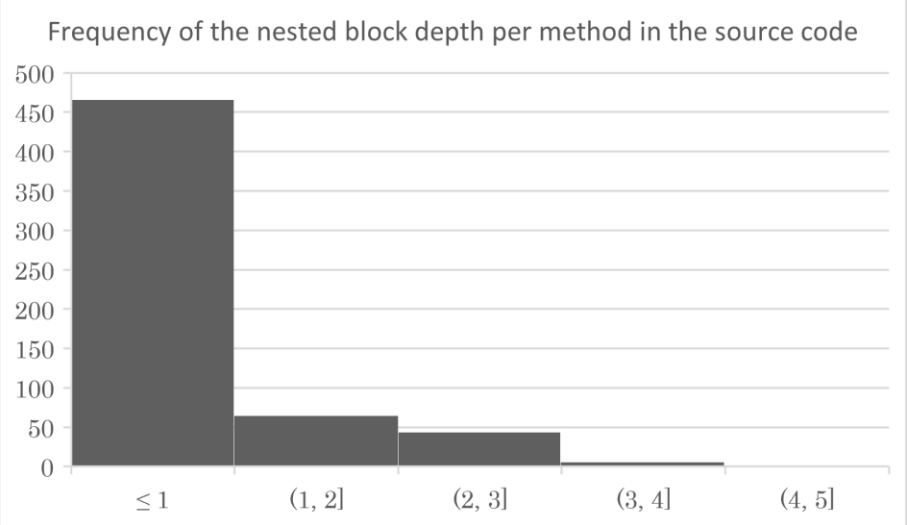


Figure 41: Histogram of the nested block depth per method – main project

To end with, as shown in Figure 41, most methods of the main project have a nested block depth lower than or equal to one, and the maximum value reached is five. Also, about 9% of the methods have a depth higher than or equal to two.

Table 10: Number of classes and methods – Selenium project

Package	Number of classes	Number of methods	Mean number of methods per class
api	2	10	5
models	15	135	9
controller	6	38	6
TOTAL	23	183	9

Table 11: Logical lines of code and class logical lines of code – Selenium project

Package	Number of lines of code	Mean number of lines of code per class	Max number of lines of code per class
api	171	86	119
models	333	22	57
controller	711	194	221
TOTAL	1,215	53	221

The Selenium project, i.e. the subsidiary project used to automate the tasks on the browser, is divided into three packages, and has 23 classes and 183 methods (cf. Table 10). As shown in Table 11, the *controller* package contains more than half the total amount of LLOC's of the project, and contains a class with 221 LLOC's, which is the highest mean number of LLOC's reached of all the packages. In total, the project has 1,215 LLOC's, and an average of 53 LLOC's per class.

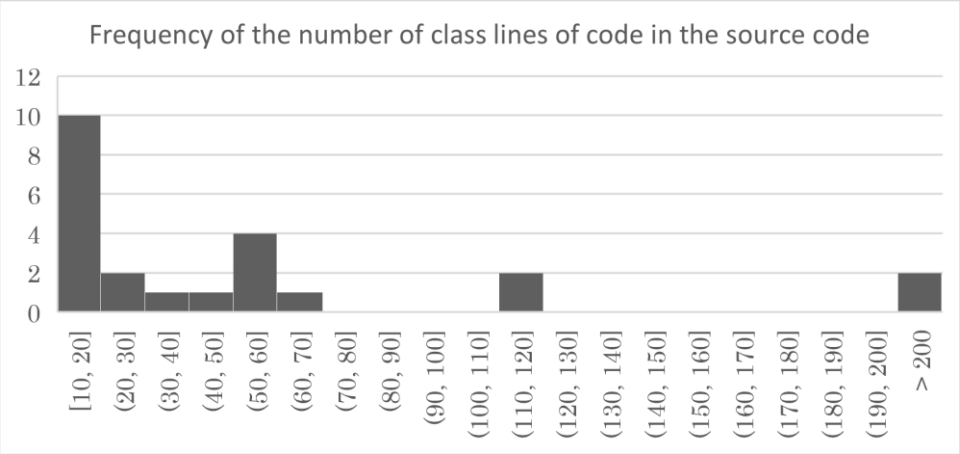


Figure 42: Histogram of the number of class lines of code – Selenium project

As illustrated by the histogram in Figure 42, most classes have between ten and one hundred LLOC's, two classes have between one hundred and two hundred LLOC's, and another two classes have more than two hundred lines of code.

Table 12: Method logical lines of code – Selenium project

Package	Number of method lines of code	Mean number of lines of code per method	Max number of lines of code per method
api	103	9	23
models	164	1	18
controller	455	12	45
TOTAL	**722**	**4**	**45**

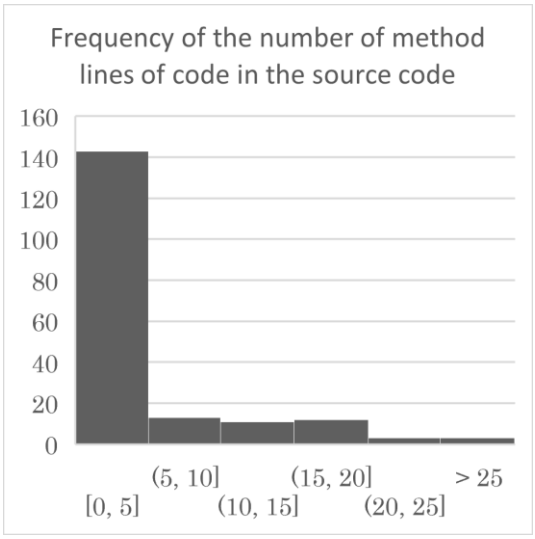


Figure 43: Histogram of the number of method lines of code – Selenium project

In total, there are 722 method LLOC's out of 1,215 logical lines of code in the source code. As stated in Table 12, the mean number of LLOC's per method is four, and the maximum value reached per method is 45. Besides, the histogram in Figure 43 shows that most methods have between zero and 20 lines of code.

Table 13: Lines of comments – Selenium project

Package	Number of lines of comments	Proportion of lines of comments (%)
api	110	39.1
models	264	44.2
controller	321	31.1
TOTAL	**695**	**36.4**

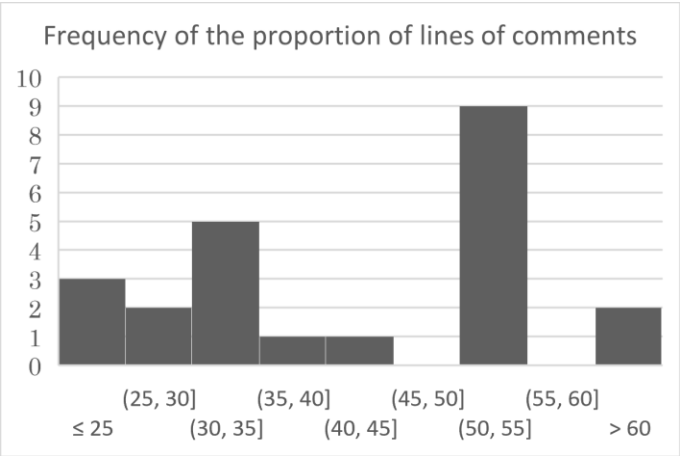


Figure 44: Histogram of the proportion of lines of comment per class – Selenium project

As specified in Table 13, there are 695 lines of comments (CLOC's) in the source code of the Selenium project; thus, 36.4% of the non-blank physical lines of code are comments. Besides, most classes have more than 25% of comments, as illustrated by the histogram in Figure 44.

The Selenium project has 2,705 lines of code (LOC's), 1,215 LLOC's, 695 CLOC's and 795 NCLOC's.

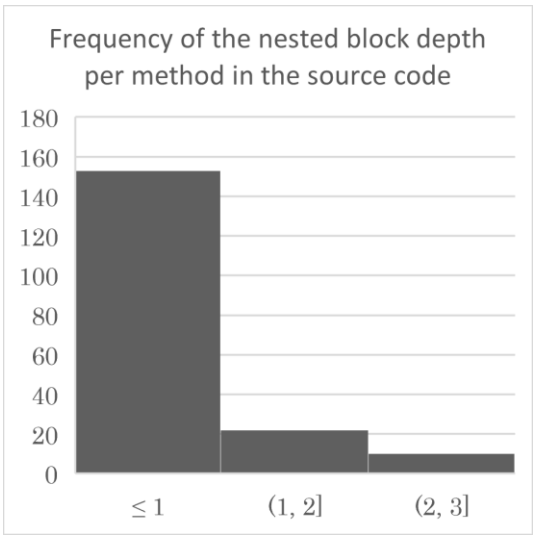


Figure 45: Histogram of the nested block depth per method – Selenium project

Lastly, as illustrated in Figure 45 for the Selenium project, the nested block depth per method is lower than or equal to three. The depth is mainly less than or equal to one, and only 6% of the methods have a depth higher than two.

To obtain easily readable and understandable code for future developers or maintainers, several indications concerning the lines of code that are to be followed were given in sections 2.3.1, 2.3.5, and 5.1. Thus, it was recommended to write methods containing less than twenty LLOC's and classes with less than two hundred LLOC's. Also, it was recommended to have a source code with a proportion of comments between 30% and 70%, and to have a nested block depth per method strictly below three in order to improve the readability and understandability of the code.

The results reported above indicate that, except for a few classes and methods, the majority have the desired number of LLOC's. Besides, although the maximum depth of nested blocks reached is five, more than 90% of the methods have a depth of less than two. The readability of the code can therefore be considered optimised as the small size of the classes and methods, as well as the low value of nested block depth, facilitates the reading and

understanding of the code. Besides, the percentage of comments in the code reaches the required lower limit, being 30%. Even though the quality of the content of the comments cannot be measured, the proportion of comments in the code corresponds to the requirement criteria. Thus, assuming that the comments are clear and concise, and therefore of good quality, they should be sufficient for understanding the code while going through it.

6.2 McCabe cyclomatic complexity

For the methods and classes of each project, the McCabe complexity was calculated, thereby giving the number of independent linear paths of the project.

Table 14: CC per method – main project

Package	Mean	Maximum
batch	2	9
business	3	7
consumer	2	5
models	1	8
TOTAL	2	9

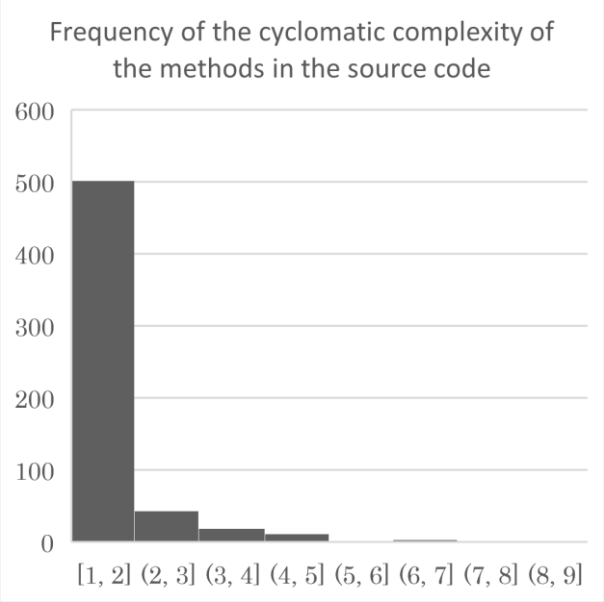


Figure 46: Histogram of the CC of the methods – main project

On average, the methods in the main project have a cyclomatic complexity of 1.89, and the maximum complexity $v_{G_{max}}$ reached by a method is nine (cf. Table 14). As shown by the histogram in Figure 46, the CC of the methods is mainly lower than five, and 80% of methods have a CC value v_G lower than or equal to two.

Table 15: CC per class – main project

Package	Mean	Maximum
batch	6	22
business	8	20
consumer	13	39
models	6	35
TOTAL	**8**	**39**

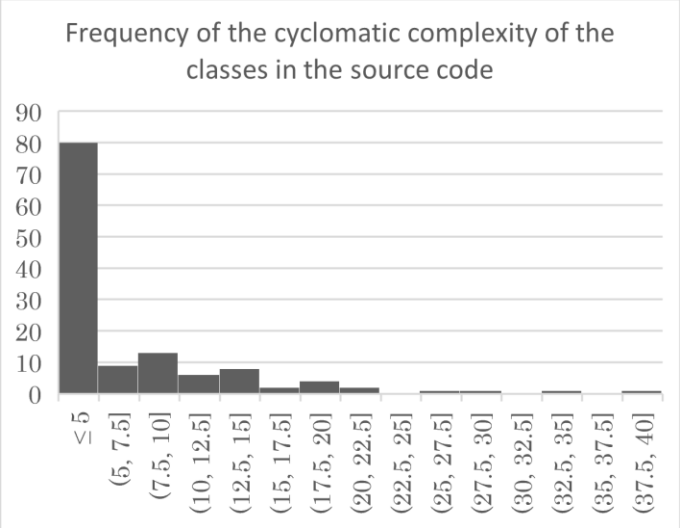


Figure 47: Histogram of the CC of the classes – main project

As shown in Table 15, the average cyclomatic complexity per class is eight, and the maximum value $v_{G_{max}}$ reached is 39. Even though 20% of the project classes have a cyclomatic complexity v_G greater than 12.5, most have a CC value lower than five (cf. Figure 47).

Table 16: CC per method – Selenium project

Package	Mean	Maximum
api	2	4
models	1	3
controller	3	8
TOTAL	**2**	**8**

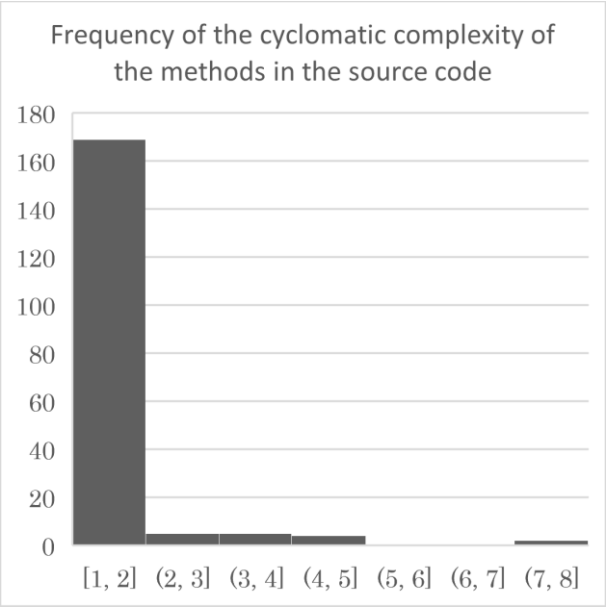


Figure 48: Histogram of the CC of the methods – Selenium project

Table 17: CC per class – Selenium project

Package	Mean	Maximum
api	10	12
models	9	34
controller	16	33
TOTAL	11	34

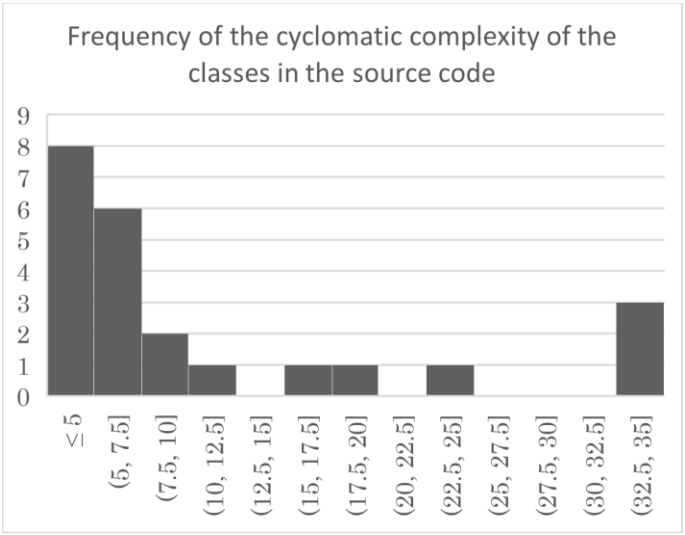


Figure 49: Histogram of the CC of the classes – Selenium project

The average cyclomatic complexity is estimated at one for the methods (cf. Table 16), and 11 for the classes (cf. Table 17), and the maximum value $v_{G_{max}}$ reached is eight for the methods, and 38 for the classes. Also, Figure 48 and

Figure 49 show that 85% of the methods have a CC v_G lower than or equal to two, and 65% of the classes have a CC below 12.5.

Cyclomatic complexity indicates the number of independent linear paths. Thus, high cyclomatic complexity indicates that the element to be tested requires a number of unit-tests at least equal to the CC, and is therefore more difficult to test. The results show that 80% of the methods have a CC of less than two, so most methods are easily testable. In addition, only 25% of the classes have a CC of more than 12.5, which means at least thirteen unit-tests per class must be written in order to have a satisfactory code coverage.

6.3 CODE SMELLS AND TECHNICAL DEBT

As explained in section 2.3.7, the technical debt incurs additional costs to the original cost of the project. Thus, the technical debt of the project is unprofitable since no benefit can be derived from it unless it has been intentionally created. Sonar estimates the debt that will have to be repaid over the life of the project by determining the quantity of code smells and the technical debt each of them causes. For example, defining a local constant rather than hard writing the value directly into the code will prevent many future errors. Therefore, not doing so could increase the technical debt.

Figure 50: Sonar technical debt results – main project[10]

Figure 51: Sonar technical debt results – Selenium project

The analysis of each project was performed by SonarQube. The technical debt accumulated for the main project is estimated at nine days and four hours for 363 code smells, while for the Selenium project, it is estimated at six days for 199 code smells. The most common code smells in both projects are related to Java convention. SonarQube uses the following formula to calculate the debt ratio:

[10] where "j" stands for "jours" in French

$$Debt\ ratio = \frac{Remediation\ cost}{Cost\ to\ develop\ 1\ line\ of\ code \times Number\ of\ lines\ of\ code}$$

Formula 15: Technical debt ratio

where the cost to develop a line of code is evaluated to 30 minutes [29]. Therefore, the debt ratio for the main project is estimated at 3.2%, and at 7.5% for the Selenium project.

In summary, for each project, the technical debt is lower than 10%. Given the size of each project, the current technical debt can be considered acceptable. Indeed, although it is estimated at seventeen days, the technical debt of the projects is low enough to not be considered a threat to the cost of the project. However, it will have to be controlled so that it does not increase too much when changes are made.

6.4 DUPLICATIONS OF CODE

The number of duplicated blocks of code in the source code is analysed to determine how easily the code can be modified. Each code block that has been duplicated may have a different number of lines of code. Based on the number of lines of duplicated code, SonarQube calculates the proportion of duplicated lines in the source code.

Figure 52: Sonar code duplication results – main project

Figure 53: Sonar code duplication results – Selenium project

The results given by the SonarQube analysis showed that there are ten duplicated blocks in the main project and that the proportion of code duplications in the source code is 2.7%. In the Selenium project, six code blocks are duplicated, and the source code contains 8.4% of code duplications. Thus, this represents 145 duplicated logical lines of code for the main project, and 114 LLOC's for the second project.

Avoiding code duplication is a way to facilitate any future code modification. Refactoring any duplicated code will facilitate modifications of the code, since it will be necessary to change it in only one place. The closer the percentage of duplicated code is to 0%, the easier it will be to modify the code.

The results indicate a proportion of duplicated blocks of less than 10%, which is satisfactory. For projects of this size, i.e. with less than ten thousand lines, this duplication rate is correct, since it represents only about two hundred lines.

6.5 DISCUSSION

The purpose of this section is to analyse further the results reached above. Indeed, the performed analysis helped to determine how maintainable the system is; yet, some things could have been done differently or could be improved in order to obtain better maintainability.

To start with, the projects contain 30% of comments, thereby making the source code readability sufficient. However, readability could have been improved by adding more comments and Javadoc in the code. Indeed, a proportion of 30% of comments is theoretically enough to ensure that the code is well understood. Yet, this proportion corresponds to the recommended lower limit for the percentage of comments. Even though having a high proportion of comments does not necessarily help to understand the code – for instance, if the comments are not good, are unclear, or do not explain what the code does – a higher proportion of comments in the code increases the chances of making it more comprehensible. As these comments are supposed to make the code more readable and understandable, it would be useful to have more.

Concerning the system testability, it could have been improved by simplifying some classes and methods. Even if the average cyclomatic complexity of one method is two, several methods in both projects have a cyclomatic complexity that is too high. This complexity makes it tedious to write tests for the classes containing these methods. The refactoring of these methods should therefore be considered in order to make those classes more testable as well as readable.

Additionally, the analysis showed that the code has less than 10% of duplicated code, which is low for projects of this size. Although it would be preferable to have a proportion closer to 0%, this rate is good enough to prove that the probability of introducing anomalies while doing any code modification is reduced. The system modifiability is therefore improved by the low rate of duplicated code in the project.

However, the project's modularity and reusability could be improved. Indeed, it would have been preferable to divide it into several modules, each of them having its own business functionalities. Therefore, implementing this batch as a multi-modular Maven project could be a solution. This project could be composed of four modules, each of which would be a batch application. The first module could aim at retrieving the ERL's: retrieving T&Cs and ERL's emails, validating T&Cs, downloading ERL's and deleting T&Cs and ERL's emails could be performed. Indeed, the process studied in this thesis is specific to a third party and might vary from one such to another, so it could be isolated. In this way, the batch application of this module could have as many different

jobs as there are different third parties. Then, the second module could be used to analyse the PDF files and extract data. Validating data and putting the ERL's in the DMS could be performed in the third module. Finally, the last module could be responsible for performing the termination on the intranet. Thus, the modules would not be interdependent, and could be reused for other similar projects. For example, since this project was implemented for a third party A, it could be adapted for a third party B by modifying only the first module.

Finally, any modification made in order to improve the maintainability of the system will have consequences on the technical debt, but also on each subcharacteristic – reusability, analysability, modularity, modifiability, and testability – of maintainability. A balance must be kept so that a project whose maintainability would be good enough while respecting the client's needs could be obtained.

7 CONCLUSION

The main objective of this thesis was to evaluate the maintainability of an RPA-based system performing the termination of insurance contracts. To achieve this, a batch application was created to process incoming termination request letters. The Selenium WebDriver free tool was used as an alternative to the RPA tools available on the market to automate tasks on the web browser. To obtain a system with good maintainability, tools such as Eclipse Metrics, SonarQube, and SonarLint were used for guidance and analysis.

The analysis of the implemented system has proven the good maintainability of the system on different points. First, some classes of the source code can be adapted to other projects. Although the validation of the T&Cs is specific to the selected third-party provider, other actions are more general and can be shared. Most classes can be shared and configured to allow generalisation, hence making some parts of the project reusable. Besides, the number of LOC's per class and per method, the depth of nested blocks, and the proportion of comments in the code meet the requirements given in section 5.1 and imposed in order to obtain a readable code. Consequently, the results concerning the LOC's and comments showed a good readability and understandability of the source code. Also, the quantity of duplicated code blocks proved that the probability of introducing errors due to an omission is reduced, thereby making the code easily modifiable. Then, the average number of independent linear paths per method was estimated at two, which means that, on average, a minimum of two unit-tests are required to test a method and have good code coverage. Because there are few unit tests to write, it is easier to determine which tests should be performed on a method. This reflects the ease with which the code can be tested, and therefore shows that the system has acceptable testability. Finally, the technical debt of each project shows that the code smells found in the source code should not interfere with future modifications if they remain under control. Overall, this analysis demonstrated that the maintainability of the system is good enough for a project implemented under industrial conditions.

The system implemented as part of this thesis is the outcome of an Atos offer made to a client. It was therefore conducted in a non-academic context, with customer constraints. This project was implemented under real conditions of use of the software. Therefore, what could have been, theoretically, a solution to improve the maintainability of the system was sometimes impossible to achieve in practice given the constraints. Indeed, the customer requirements were to obtain a low-cost system with a low development cost, so it was necessary to reduce the development period, not to mention that the use of

some of the client's internally developed software was necessary in the process: not having control over that was sometimes an obstacle in keeping a good maintainability of the system. Consequently, time restriction and not having control over some internally developed code had a significant impact on the writing of the source code and its maintainability. Another limitation of this thesis is due to the use of Selenium: although using Selenium WebDriver was challenging at first, the tasks were successfully performed in the end. However, its behaviour could sometimes be unpredictable and unexpected, although it was eventually controlled. To limit the risk of errors, it could be envisaged using this tool only on extranets, and use web services on the internal system. To conclude about Selenium WebDriver, its use as an alternative RPA tool in this case study was conclusive.

Due to the limited time to work on this thesis, and since the developed system is for a client and has been developed on the client's internal system, this thesis only focused on the maintainability of the software system. Future research could focus on Selenium and show to what extent it can be used as an alternative to existing RPA tools, and under what conditions.

REFERENCES

[1] G. G. Sreeja. and M. Shanthini, "Robotic Process Automation for Insurance Claim Registration," *International Journal of Advanced Research (IJAR),* vol. 6, no. 4, pp. 23-27, April 2018.

[2] S. Aguirre and A. Rodriguez, "Automation of a Business Process Using Robotic Process Automation (RPA): A Case Study," in *Applied Computer Sciences in Engineering*, Cartagena (Colombia), Springer, Cham, 2017, pp. 65-71.

[3] Capgemini Consulting and Capgemini Business Services, "Robotic Process Automation - Robots conquer business processes in back offices," 2016. [Online]. Available: https://www.capgemini.com/consulting-de/wp-content/uploads/sites/32/2017/08/robotic-process-automation-study.pdf. [Accessed 20 September 2018].

[4] "ISO/IEC-24765 – Systems and software engineering — Vocabulary," ISO, Geneva (Switzerland), 2017.

[5] "ISO/IEC-25010 – Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," ISO, 2011.

[6] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, 3rd ed., New York (USA): Addison-Wesley Professional, 2012.

[7] Network for Business Sustainability, "Long-term Thinking in a Short-term World: A Guide for Executives," Network for Business Sustainability, 2015.

[8] C. Chen, R. Alfayez, K. Srisopha, B. Boehm and L. Shi, "Why is it Important to Measure Maintainability, and What are the Best Ways to Do it?," in *2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion*, Buenos Aires (Argentina), 2017.

[9] K. Lambertz, "Complexité et qualité – Comment mesurer la complexité d'un logiciel," *MSCoder,* 2007.

[10] C. Ebert and J. Cain, "Cyclomatic Complexity," *IEEE Software,* vol. 33, no. 6, pp. 27-29, Nov./Dec. 2016.

[11] R. Niedermayr, “Why we don’t use the Software Maintainability Index,” CQSE, 2 March 2016. [Online]. Available: https://www.cqse.eu/en/blog/maintainability-index/. [Accessed 25 November 2018].

[12] D. I. Sjøberg, B. Anda and A. Mockus, “Questioning Software Maintenance Metrics: A Comparative Case Study,” in *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Lund, Sweden, IEEE, 2012, pp. 107-110.

[13] S. Counsell, X. Liu, S. Eldh, R. Tonelli, M. Marchesi, G. Concas and A. Murgia, “Re-visiting the ‘Maintainability Index’ Metric from an Object-Oriented Perspective,” IEEE, Funchal (Portugal), 2015.

[14] H. Artaza, N. C. Hong, M. Corpas, A. Corpuz, R. Hooft, R. C. Jiménez, B. Leskošek, B. G. Olivier, J. Stourac, R. Svobodová Vařeková, T. Van Parys and D. Vaughan, “Top 10 metrics for life science software good practices,” *F1000RESEARCH,* vol. 5, 2016.

[15] M. Fowler and K. Beck, “Bad Smells in Code,” in *Refactoring: Improving the Design of Existing Code*, Pearson Education, 1999.

[16] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord and I. Gorton, “Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt,” ACM, Bergamo (Italy), 2015.

[17] X. Blanc, “La dette technique expliquée !,” 2016. [Online]. Available: https://promyze.com/wp-content/uploads/ladettetechnique.pdf. [Accessed 23 November 2018].

[18] P. Kruchten, R. L. Nord and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Pratice,” *IEEE Software,* vol. 29, no. 6, pp. 18-21, November/December 2012.

[19] J.-L. Letouzey and M. Ilkiewicz, “Managing Technical Debt with the SQALE Method,” *IEEE Software,* pp. 44-51, 2012.

[20] M.-E. Bobillier-Chaumon et P. Sarnin, «Les conditions de travail,» chez *Manuel de psychologie du travail et des organisations*, 1 éd., De Boeck, 2012, pp. 97-99.

[21] M. Wall, “Adapt or die: How to cope when the bots take your job,” 16 March 2018. [Online]. Available: https://www.bbc.co.uk/news/business-43259906. [Accessed 25 November 2018].

[22] K.-F. Lee, “Kai-Fu Lee: How AI can save our humanity | TED Talk,” TED, 2018. [Online]. Available: https://www.ted.com/talks/kai_fu_lee_how_ai_can_save_our_humanity. [Accessed 7 January 2019].

[23] Blue Prism, “Blue Prism - Robotic Process Automation,” Blue Prism, [Online]. Available: http://www.blueprism.com/. [Accessed 12 December 2018].

[24] UiPath, “Robotic Process Automation | UiPath,” UiPath, [Online]. Available: https://www.uipath.com/. [Accessed 12 December 2018].

[25] S. Kappagantula, “RPA Tools List and Comparison | Leaders in RPA Software | Edureka,” 25 December 2018. [Online]. Available: https://www.edureka.co/blog/rpa-tools-list-and-comparison/#Types%20of%20RPA%20Tools. [Accessed 15 January 2019].

[26] B. Belabbess, M. Bairat, J. Lhez, Z. Khattabi, Y. Zheng and O. Curé, “Scouter: A Stream Processing Web Analyzer to Contextualize Singularities,” in *Proceedings of the 21st International Conference on Extending Database Technology (EDBT)*, Bezons (France), 2018.

[27] J.-L. Letouzey and T. Coq, “The « SQALE » Analysis Model: An analysis model compliant with the representation condition for assessing the Quality of Software Source Code,” *2010 Second International Conference on Advances in System Testing and Validation Lifecycle,* pp. 43-48, 2010.

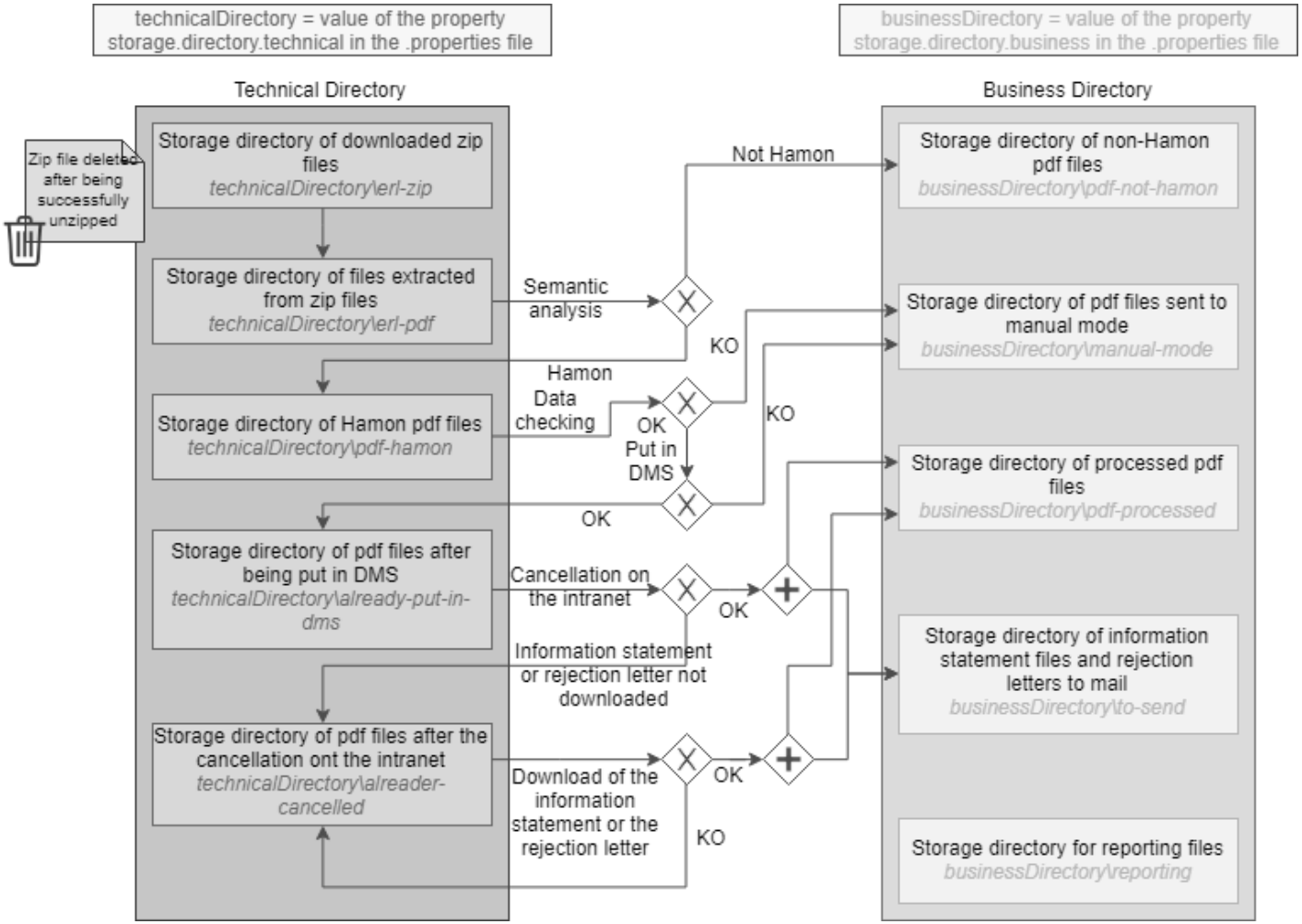
[28] J.-L. Letouzey, “The SQALE Method, Definition Document,” 31 March 2016. [Online]. Available: http://www.sqale.org/wp-content/uploads/2016/08/SQALE-Method-EN-V1-1.pdf. [Accessed 10 January 2019].

[29] SonarQube, “Metric Definitions,” SonarSource, [Online]. Available: https://docs.sonarqube.org/latest/user-guide/metric-definitions/. [Accessed 10 02 2019].

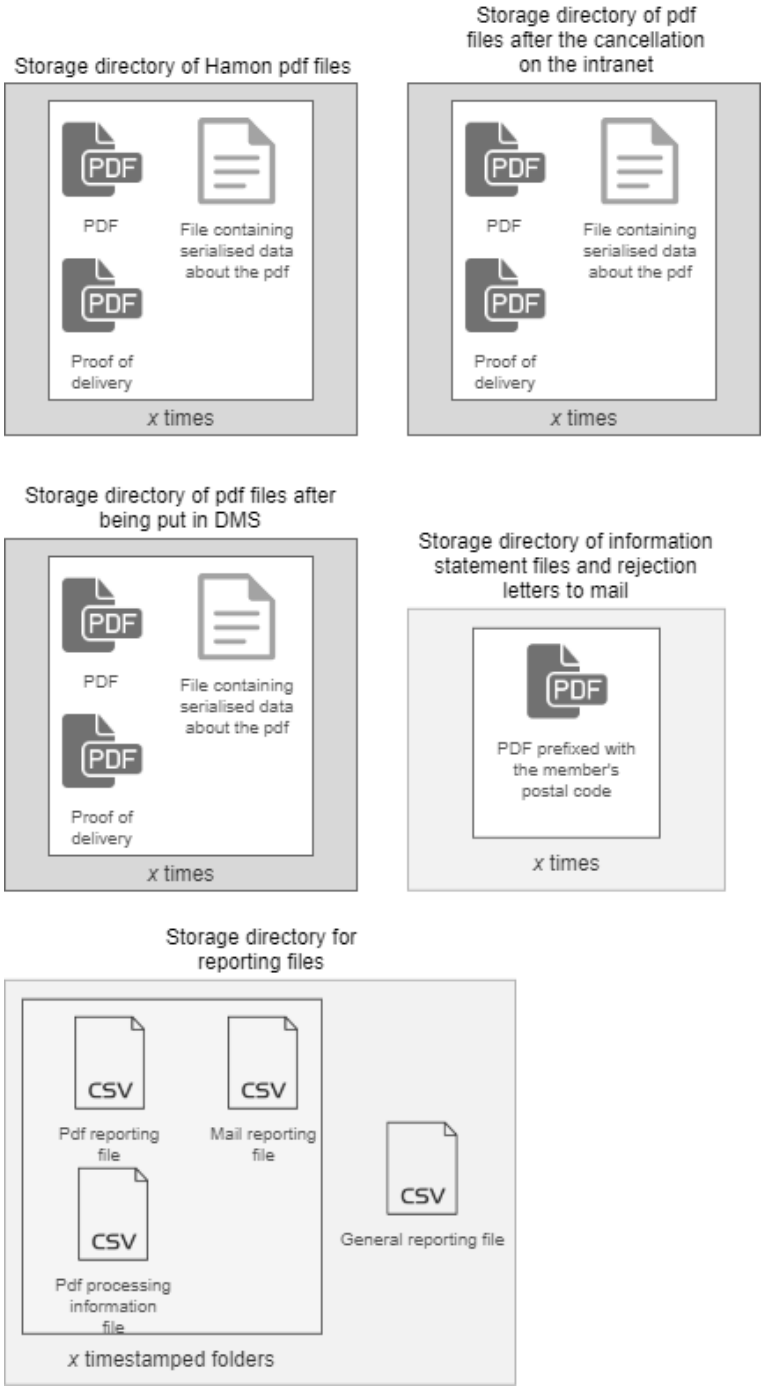
[30] Thoughtonomy, “Robotic Process Automation - 6 real world use cases (white paper),” 2016.

[31] A. Van Deursen, “Think Twice Before Using the “Maintainability Index”,” August 2014. [Online]. Available: https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/. [Accessed 25 November 2018].

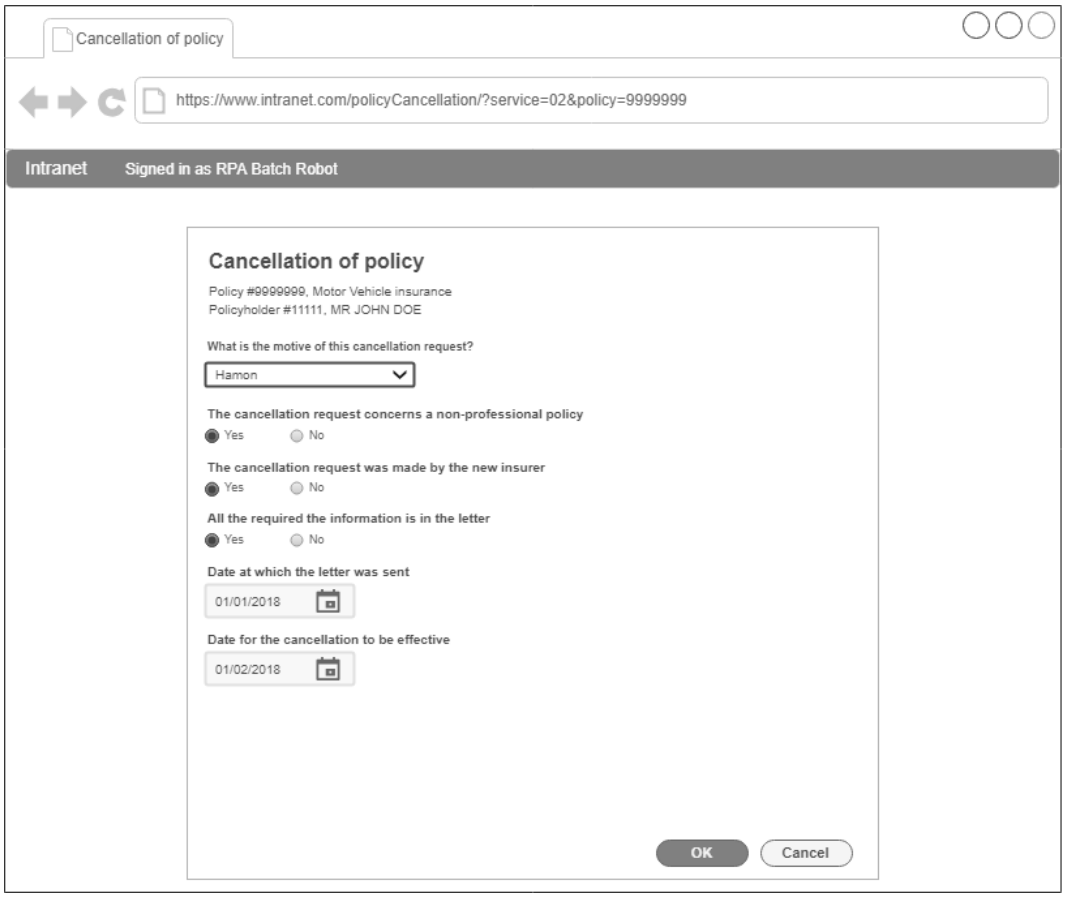
APPENDICES



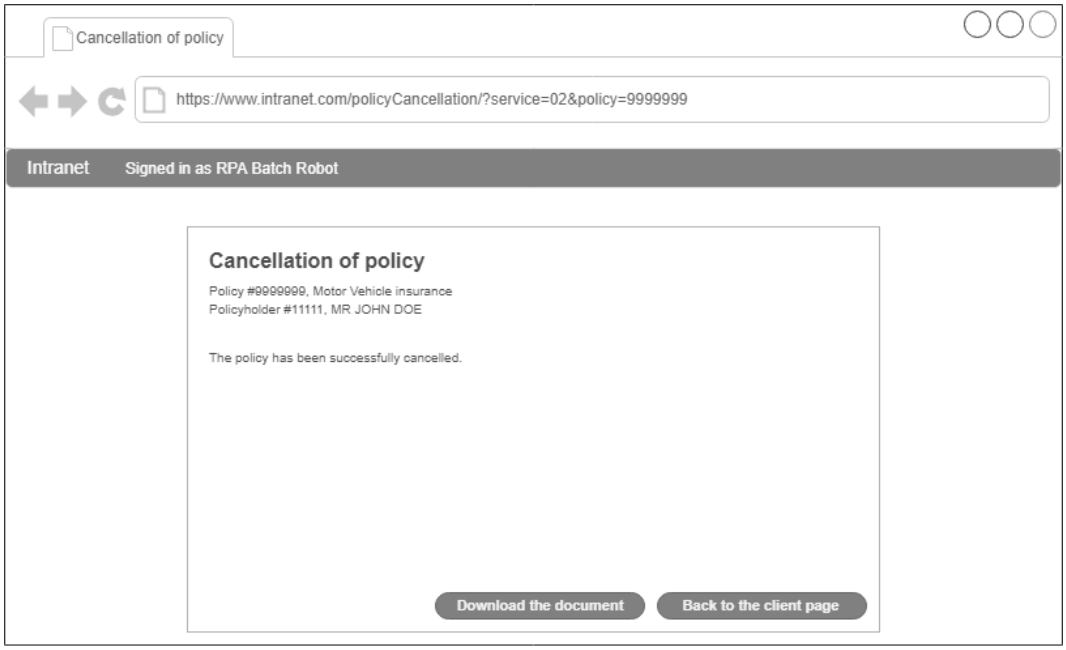
Appendix I: Organisation of the storage directories



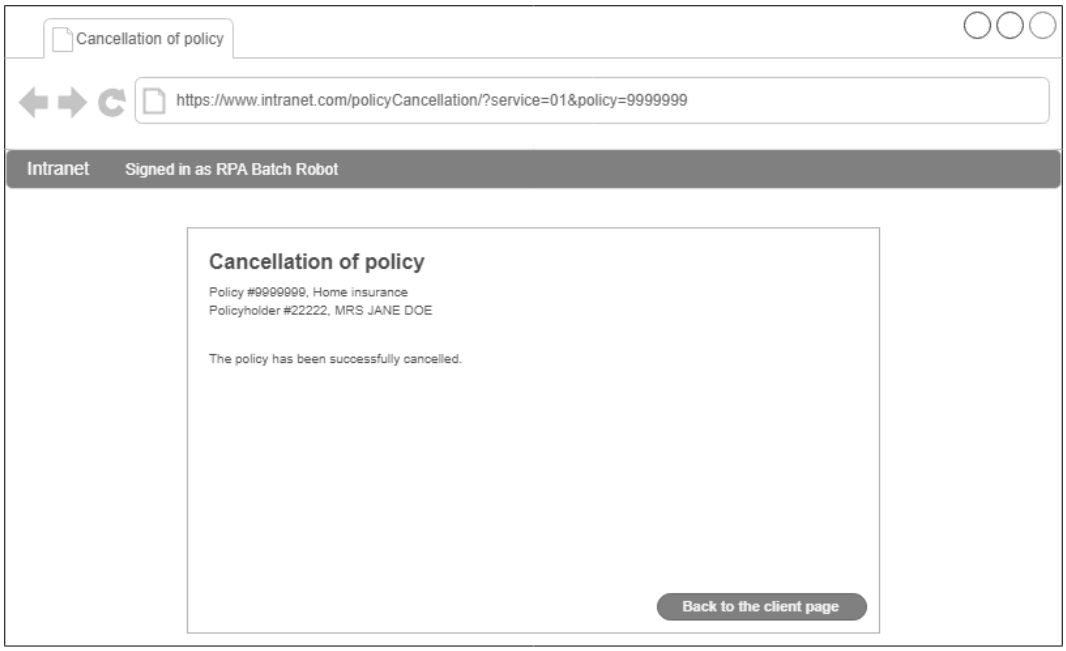
Appendix II: Content of the folders



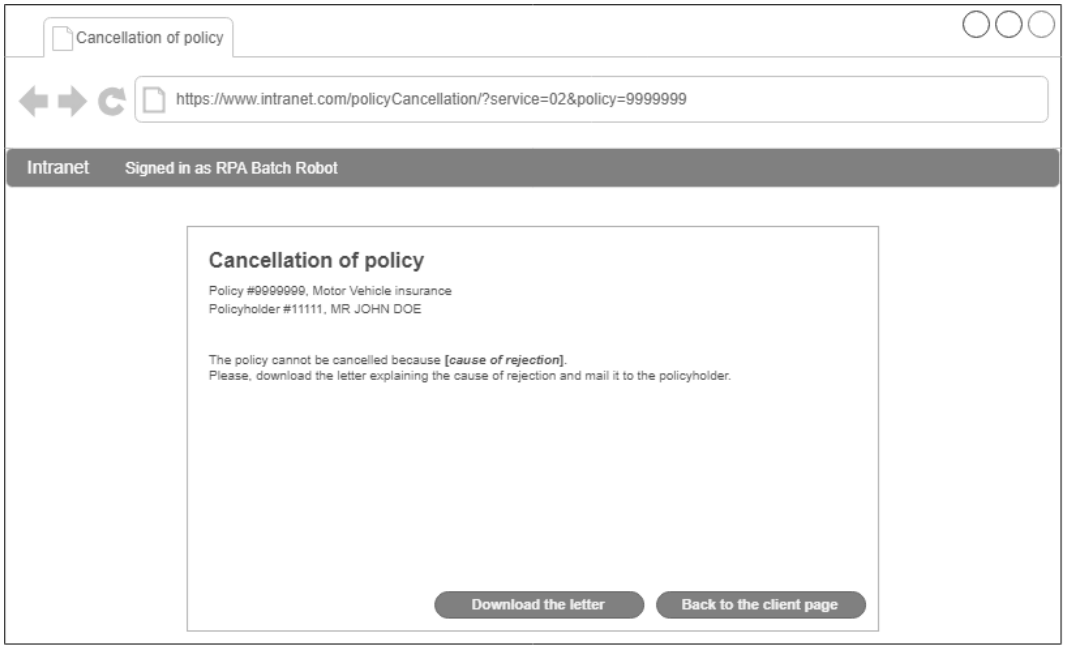
Appendix III: Web Page “Cancellation of policy": Hamon motive selected, non-professional policy, request made by the insurer and presence of all the required information



Appendix IV: Web Page “Cancellation of policy": success of the cancellation of a motor vehicle insurance policy



Appendix V: Web Page “Cancellation of policy": success of the cancellation of a home insurance policy



Appendix VI: Web Page “Cancellation of policy": rejection of the cancellation of an insurance policy