



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
AUTOMATION AND CONTROL DEPARTMENT

Maria Laura Brzezinski Meyer

Artificial intelligence algorithms application to the problem of automatic selecting integration tests

Toulouse, France

2020

Maria Laura Brzezinski Meyer

Artificial intelligence algorithms application to the problem of automatic selecting integration tests

Monography submitted to the Federal University of Santa Catarina (Universidade Federal de Santa Catarina) as a requirement to the course approval **DAS 5511: Project for End of Study** from the course of Control and Automation Engineering.
Supervisor: Fernand Cuesta
Co-supervisor: Joni da Silva Fraga

Toulouse, France
2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Meyer, Maria Laura Brzezinski

Artificial intelligence algorithms application to the
problem of automatic selecting integration tests / Maria
Laura Brzezinski Meyer ; orientador, Fernand Cuesta,
coorientador, Joni da Silva Fraga, 2020.

51 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia de Controle e Automação,
Florianópolis, 2020.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Inteligencia
Artificial. 3. Teste de Softwares. 4. Data Science. 5.
Data Analysis. I. Cuesta, Fernand. II. da Silva Fraga,
Joni. III. Universidade Federal de Santa Catarina.
Graduação em Engenharia de Controle e Automação. IV. Título.

Maria Laura Brzezinski Meyer

Artificial intelligence algorithms application to the problem of automatic selecting integration tests

This internship report was evaluated in the context of the discipline DAS5511: Project for End of Study and APPROVED in its original last version by the course of Control and Automation Engineering.

Fernand Cuesta
Local Supervisor
Renault Software Labs
Toulouse, France

Toulouse, 16 of July of 2020 .

ACKNOWLEDGEMENTS

The author would like to thank Renault Software Labs for the resources necessary and also the brazilian university UFSC (Universidade Federal de Santa Catarina), the french college INP - ENSEEIHT (Institut National Polytechnique - École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications) and the brazilian institute CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior). Thanks also to the entreprise tutor Fernand Cuesta, who guided me in this work, to Joni da Silva Fraga that oriented me from Brésil, and César Slogo and my family for the support.

*"I believe that the attempt to make a thinking machine will help us greatly in finding out how we think ourselves."
(Alan M. Turing, 1951)*


RESUMO

As mudanças que o modelo de gestão de projetos vem sofrendo traz grandes vantagens às empresas, porém novas problemáticas são inseridas. Nos métodos clássicos de desenvolvimento, um projeto é realizado na forma de *cascata*, ou seja, primeiro o modelo é construído, depois é realizado o desenvolvimento e por último o projeto é validado por testes. Esse modelo é custoso em termos de validação, pois as falhas são encontradas apenas na fase final de produção, tornando a correção mais difícil e cara. Atualmente o modelo dito *ágil* é utilizado, onde diversos ciclos de planejamento-desenvolvimento-testes são realizados, tornando o processo mais rápido e flexível.

Contudo, esse modelo inviabiliza a execução de todos os testes necessários na produção de um veículo, pois não há tempo hábil para a execução deles, além de não ser necessário testar certas funções em todas as etapas do projeto. Por exemplo, testar o Bluetooth do carro antes mesmo do sistema multimídia ser implementado é inconcebível. Visto a necessidade de selecionar uma quantidade limitada de teste para cada etapa do projeto, o estudo de como selecionar de forma automática e segura é necessário. O objetivo do projeto aqui apresentado é de estudar formas de selecionar testes de acordo com as funcionalidades do software, bem como a relação entre a confiabilidade do sistema em relação aos testes realizados.

O estágio foi realizado na empresa Renault Software Labs em Toulouse, uma divisão de pesquisa e desenvolvimento do grupo Renault. Tal projeto é uma proposta de tese de doutorado, portanto durante o estágio de 6 meses, as fases iniciais foram desenvolvidas: coleta e preparação de dados. Os dados coletados são referentes à descrição dos testes do catálogo da Renault, ao histórico de execução desses testes e também aos defeitos encontrados por eles durante a fase de validação de softwares para veículos conectados. Além disso, foi realizado um estudo bibliográfico inicial a fim de identificar as diferentes abordagens e metodologias existentes. Por fim, foram realizadas análises das informações adquiridas e da relevância de cada uma em relação ao resultado dos testes.

Palavras-chave: Testes de Softwares. Testes de Integração. Seleção. Método Ágil. Integração Contínua. Validação de Software. Inteligência Artificial. Segurança de Funcionamento. Ciência de Dados.


 **Atenção:** Os dados apresentados nesse documento não condizem à realidade. Por questões de confidencialidade, os nomes dos testes, as escalas e os valores exatos foram retirados ou modificados.

ABSTRACT

The changes that the project management model has undergone bring great advantages to companies, but new problems are introduced. In the classic methods of development, a project is carried out in the form a *V* form, that is, first the model is built, then development is carried out and finally the project is validated by tests. This model is costly in terms of validation, as the flaws are found only in the final stage of production, making the correction more difficult and expensive. Currently, the so-called *agile* model is used, where several planning-development-testing cycles are performed, making the process faster and more flexible. However, this model makes it impossible to carry out all the necessary tests in the production of a vehicle, as there is no time to execute them, and it is not necessary to test certain functions at all stages of the project. Given the need to select a limited amount of test for each stage of the project, the study of how to select automatically and safely is necessary. The objective of the project presented here is to study ways to select tests according to the software's functionalities, as well as the relationship between the reliability of the system in relation to the tests performed.

The internship took place at Renault Software Labs in Toulouse, a research and development division of the Renault group. Such a project is a doctoral thesis proposal, so during the 6-month internship, the initial phases were developed: data collection and preparation. The collected data refer to the description of the tests in the Renault catalog, the history of the execution of these tests and also the defects found by them during the software validation phase for connected vehicles. In addition, an initial bibliographic study was carried out in order to identify the different existing approaches and methodologies. Finally, analyzes were performed on the information acquired and the relevance of each in relation to the test results.

Keywords: Software Testing. Integration Testing. Selection. Method Agile. Continuous Integration. Software Validation. Artificial Intelligence. Dependable Computing. Data Science.


 **Attention: The data presented in this document do not correspond to reality. For reasons of confidentiality, the names of the tests, the scales and the exact values have been removed or modified.**

RÉSUMÉ

Les changements de modèle de développement de projet logiciel apportent de grands avantages aux entreprises, mais se heurtent à de nouveaux problèmes. Dans les méthodes classiques de développement, un projet est développé en suivant le process en cycle dit « V », c'est-à-dire que le produit est d'abord spécifié, puis le développement est effectué et ce n'est qu'à la fin de ce cycle que tout le produit est validé par des tests. Ce modèle est coûteux en termes de validation car la détection des fautes s'effectue qu'à la fin du processus de production, rendant la correction plus difficile et plus coûteuse. Actuellement, les modèles appelés *agiles* commencent à être utilisés, où plusieurs cycles de planification-développement-test sont effectués rendant le processus de développement plus rapide et plus flexible. Cependant, l'exécution de tous les tests nécessaires au test d'un véhicule devient impossible dans les temps impartis par les cycles courts et successifs de ces méthodes agiles. Compte tenu de la nécessité de sélectionner une quantité limitée de tests pour chaque étape du projet, il est nécessaire d'étudier une méthode de sélection automatiquement de tests en prenant en compte les risques liés à la détection de défaut. L'objectif du projet présenté ici est d'étudier les modalités de sélection de tests en fonction des fonctionnalités du logiciel, ainsi que la relation entre la fiabilité du système par rapport aux tests effectués.

Le stage s'est déroulé au sein de l'entreprise Renault Software Labs à Toulouse, une division de recherche et développement du groupe Renault. Le projet, dans son ensemble, est une proposition de thèse. Le stage de 6 mois, préliminaire à cette thèse, s'est concentré sur les phases initiales du projet : la collecte et la préparation des données. Les données collectées sont tirées de la description des tests dans le catalogue de test Renault, de l'historique d'exécution de ces tests ainsi qu'aux défauts constatés lors de leur exécution durant les phases de validation logicielle des véhicules connectés. Dans un même temps, une première étude bibliographique a été menée afin d'identifier les différentes approches et méthodologies existantes sur le thème de ce stage. En fin de rapport, des analyses ont été réalisées sur les informations acquises, ainsi que sur la pertinence de chacune par rapport aux résultats des tests.

Mots-clés : Test de Logiciels. Test d'Intégration. Sélection. Méthode Agile. Intégration Continue. Validation des Logiciels. Intelligence Artificielle. Sûreté de Fonctionnement. Science des Données.

 **Attention** : Les données présentées dans ce document ne correspondent pas à la réalité. Pour des raisons de confidentialité, les noms des tests, les échelles et les valeurs exactes ont été supprimés ou modifiés.

LIST OF FIGURES

Figure 1 – Alliance logo.	13
Figure 2 – Renault Software Labs logo.	13
Figure 3 – V model.	14
Figure 4 – Agile methodology.	15
Figure 5 – Continuous Integration.	17
Figure 6 – Integration testing cycle.	19
Figure 7 – Test cycle example.	20
Figure 8 – Test cycle redistribution.	20
Figure 9 – Changed based test selection.	21
Figure 10 – Reinforcement learning schema.	23
Figure 11 – Project overall.	25
Figure 12 – Data filtering process.	26
Figure 13 – Project pipeline.	28
Figure 14 – API usage.	29
Figure 15 – Data Frame example.	30
Figure 16 – Issues from Jira Project Settings.	31
Figure 17 – Main issues types.	32
Figure 18 – Jira data structure.	32
Figure 19 – Structure of the Jira data collection code.	33
Figure 20 – Silk Central data structure with example.	35
Figure 21 – Time spent by data scientists.	36
Figure 22 – Automated and manual tests.	38
Figure 23 – Tests case status.	38
Figure 24 – Tests runs status.	39
Figure 25 – Clusters PASS/FAIL.	40
Figure 26 – Distribution of executions duration.	40
Figure 27 – Top 10 tests cases for bugs founded.	41
Figure 28 – Feature importance for all issues linked.	42
Figure 29 – Feature importance for defects related issues.	42
Figure 30 – Tests case most used.	43
Figure 31 – Number of steps distribution.	44
Figure 32 – Feature importance for test case feature.	45
Figure 33 – Feature importance for test executions tags.	46

LIST OF TABLES

Table 2 – Approaches comparison.	24
Table 3 – Tests main parameters.	37
Table 4 – Vectors descriptive statistics.	44

LIST OF ABBREVIATIONS AND ACRONYMS

APFD	Average Percentage of Faults Detected
API	Application Programming Interfaces
CFG	Control Flow Graph
CI	Continuous Integration
JSON	JavaScript Object Notation
MLP NN	Multi-Layer Perceptron Neural Network
REST	Representational State Transfer
RSWLT	Renault Software Labs Toulouse
SIT	System Integration Team
SOAP	Simple Object Access Protocol
XAI	Explainable AI
XML	Extensible Markup Language

CONTENTS

1	INTRODUCTION	13
1.1	ENTERPRISE PRESENTATION	13
1.2	SUBJECT PRESENTATION	14
2	PROJECT CONTEXT	16
2.1	SOFTWARE TESTING	16
2.2	CONTINUOUS INTEGRATION DEVELOPMENT	17
2.3	TEST SELECTION PROBLEM	18
3	STATE OF ART	21
3.1	CHANGES BASED APPROACH	21
3.2	BLACK BOX APPROACH	22
3.3	REINFORCEMENT LEARNING	22
3.4	COMPARISON	23
4	PROJECT OVERALL	25
5	DATA COLLECTION	29
5.1	DATA FORMAT	29
5.2	FIRST DATA SET - JIRA	30
5.3	NEW DATA SET - SILK CENTRAL	34
6	DATA PREPARATION	36
6.1	THE DATABASE	37
6.2	FEATURES	39
6.2.1	Test run Status	39
6.2.2	Test execution Duration	39
6.2.3	Issues funded	41
6.2.4	Tests case Usage	41
6.2.5	Tests case Complexity	43
6.2.6	First vectors	43
6.3	CATEGORICAL DATA	44
7	CONCLUSION	47
7.1	FUTURE WORK	47
7.2	FINAL CONSIDERATIONS	47
	BIBLIOGRAPHY	48
	APPENDIX A – JIRA’S DATABASE FEATURES	51

1 INTRODUCTION

The purpose of this paper is to describe the project done during the final year internship by the student Maria Laura Brzezinski Meyer. The work was done at Renault Software Labs in Toulouse - France, starting on February 17 through August 17, 2020.

⚠ Attention: The data presented in this document do not correspond to reality. For reasons of confidentiality, the names of the tests, the scales and the exact values have been removed or modified.

1.1 ENTERPRISE PRESENTATION

Figure 1 – Alliance logo.



Source: (RENAULT–NISSAN–MITSUBISHI... , 2020)

The Renault group is a company responsible for the manufacture of vehicles, founded in 1898 by Louis Renault. It is present in 134 countries and sold nearly 3.8 million vehicles in 2019 (GROUPE RENAULT... , 2020). The group is formed by five brands: Renault, Dacia, Renault Samsung Motors, Alpine and LADA. In 1999, a partnership was signed between the Renault group and the Japanese company Nissan. Then Mitsubishi joined it in 2017, creating the Renault–Nissan–Mitsubishi Alliance, responsible for more than 1 of 9 vehicle sell's in the world (RENAULT–NISSAN–MITSUBISHI... , 2020).

Figure 2 – Renault Software Labs logo.



Source: (RENAULT... , 2020)

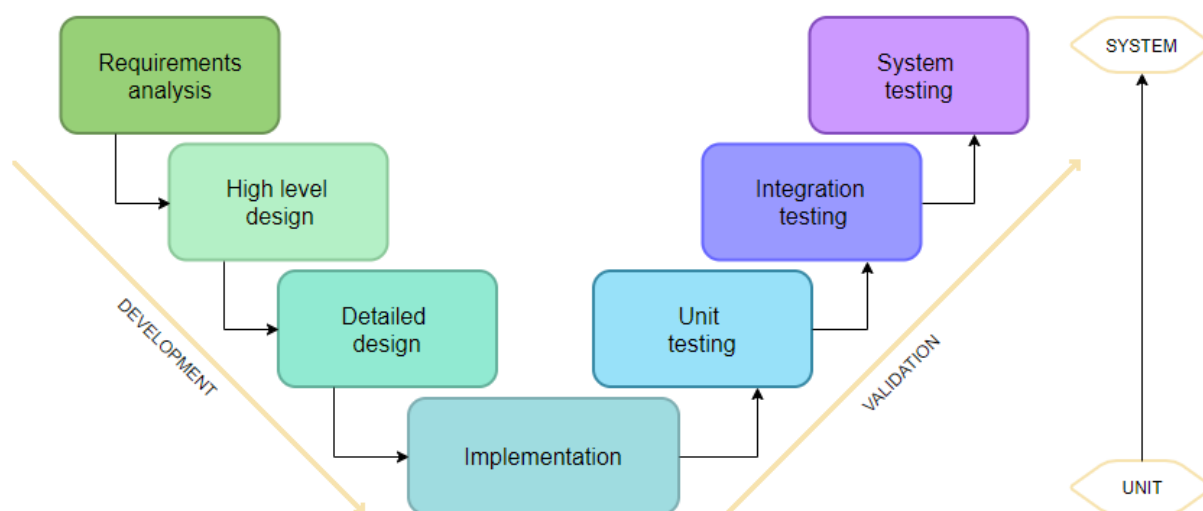
Renault Software Labs (RENAULT... , 2020) is a branch of Renault group since 2017, grouping more than 400 engineers divided into two places, Sophia-Antipolis and Toulouse. These are innovations centers of Research and Development (R&D) responsible for software's architecture and embedded systems that are integrated into

connected, autonomous and electric vehicles, strengthening the development of the new software generation incorporated in its vehicles, capable of offering personalized services, updating remotely, independently and in real-time. Renault Software Labs Toulouse (RSWLT) defines its expertise on several domains, like embedded systems architecture, integrated system and software, continuous integration and platforming, eco-energy, cybersecurity, modeling and simulation, multi-media, mobile products and internet of things, cloud, data management, artificial intelligence, machine learning and security. This project is part of the System Integration Team (SIT), a group that is in charge of multimedia and connectivity tests.

1.2 SUBJECT PRESENTATION

The change in the project management model brings great advantages, but some flaws. In the classic methodology, a project was carried out following the “V” model process, that is, first the model is specified, then development is carried out and, lastly, the system is validated (figure 3).

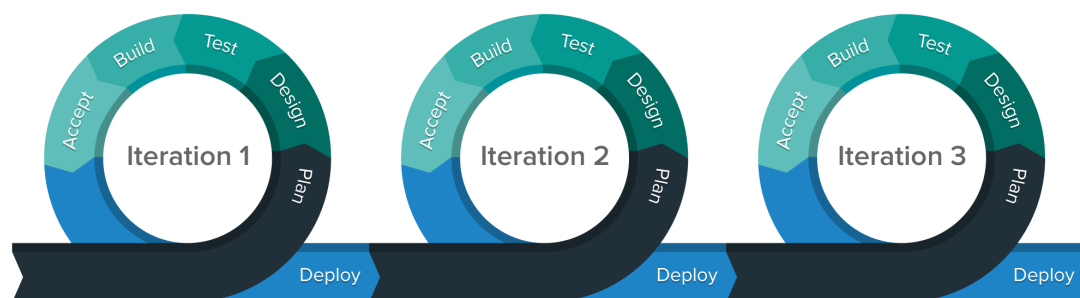
Figure 3 – V model.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

This model is expensive in terms of validation because it is done at the end of the product development, so if some flaws are found in the final production stage, their corrections become highly difficult and expensive. Currently, an agile model named continuous integration is used, where several planning-development-testing cycles are performed, making the validation much easier for each cycle and the integration of all subsystems became more flexible to construct. Figure 4 exemplifies three interactions in agile methodology.

Figure 4 – Agile methodology.



Source: (PIVOTAL TRACKER, 2020).

Renault Software Labs uses this method, which will be explained in more detail in the next chapter, for integrating software code. This technique makes it possible to verify in each version of the source code if the modifications result does not produce a regression in the developed system. Thus, a piece of code by piece of code, the system is developed and validated in incremental steps. This concept, mentioned for the first time by Grady Booch (BOOCH, 2004), aims to detect problems related to software integration at the earliest time in the project development cycle. Problem detection is made possible by the automation of test batteries that are triggered with each new version of the software.

However, with the increase of complexity of new systems such there are been seen in Automotive sector, the continuous integration model makes it impossible to carry out all the tests necessary in a vehicle production with a reasonable time frame delivered for the daily testing. Besides, during the development, some features are not yet integrated, so there is no need to test it. For example, testing the car's Bluetooth before the multimedia system is even implemented is inconceivable.

Given the need to select a limited amount of tests for each stage of the project, the study of how to select it automatically without compromise systems safely is necessary. The goal of the project is to study ways to select tests according to the software's functionalities, as well as the relationship between the system's reliability in relation to the tests performed. The results of tests already performed will be used as a base to select new tests for futures projects. To do so, the feasibility of implementing artificial intelligence algorithms will be studied.

2 PROJECT CONTEXT

In order to understand the context of the project, definitions of where, how and why the test selection algorithm will be developed will be explicit in this chapter.

2.1 SOFTWARE TESTING

The project domain is software testing, a crucial step in the development of any product. The aim of a test is to certify that all functional and non-functional requirements of the project are being respected. Therefore, tests are used to reduce software failure risks, and to do so, they help to find and eliminate errors.

Functional requirements are linked to the system definition, describing the functions that a software must perform, that is, inputs, behavior, and outputs. Functional software requirements help to capture the intended behavior of the system, whether expressed by the functions, services, or tasks that the system must perform. Non-functional requirements are related to the software quality, being essential to guarantee the usability and effectiveness of the entire system.

For example, in a car digital speed counter, it is necessary that a speed value is shown in the display, this is its functionality. But it is also important to ensure that the value shown matches the real car speed, so the software is delivering the correct work, this is a non-functional requirement.

A test can be either manual, when a person is needed to execute testing actions (like click on the Bluetooth button to check if a green light turns on), or automatic, when a script is written to perform testing actions faster and repeatedly. There are three methods to execute a test:

- Black box: it reviews only the application's functionalities, behaving as a user, what happens internally does not matter, only exits are verified.
- Grin box: a tester gives the system an input, verify if the result is like expected, and also checks through which process this result was obtained. In this case, the role of the system, its functionalities and its internal mechanisms are known, however there is no code source access.
- White box: all software's internal components are tested through its source code in this method, so here tests have a developer view.

Tests are also differentiated by their degree of granularity, depending on the development stage. Unit testing is used to verify individual components of a software, then integration testing analyses the interaction between these components. Increasing more the granularity degree, the next is system testings, where a complete and integrated software is tested. Finally, acceptance testing is the process phase where the

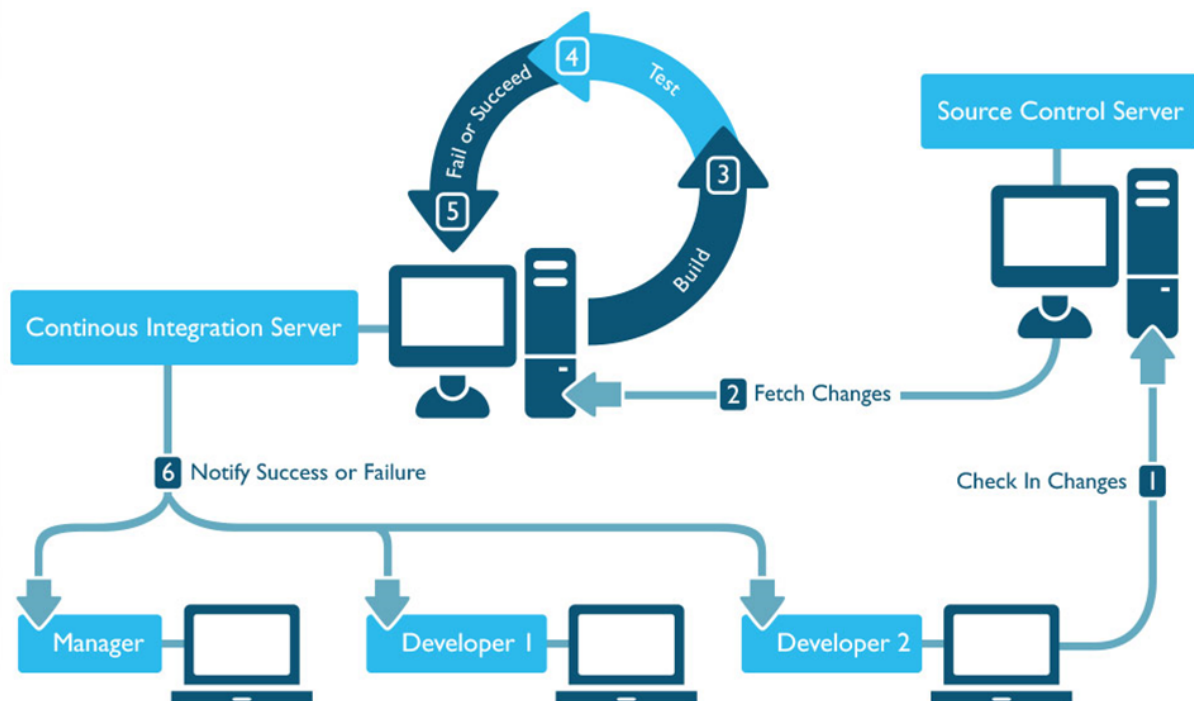
system is tested by a black box test before its release to verify if all users case scenarios are covered, that why it is also called validation testing. This project is concentrated in the integration software testing stage, where the entire build is created.

2.2 CONTINUOUS INTEGRATION DEVELOPMENT

In software engineering, a new way of development is been implemented, it is an agile method called Continuous Integration (CI). The term was described in Martin Fowler's blog publication in 2000 (FOWLER, 2006) and explored by many authors like (ZHAO *et al.*, 2017), it is a software development practice where work is frequently integrated and validated.

The diagram in figure 5 shows how the software development process is done using continuous integration practice. First, a development is done by one or many developers, these new features or code changes are sent to a source control server (normally a git repository), then the code is built and tested. These validation results are sent to the development team, so issues can be fixed and then a new cycle can begin.

Figure 5 – Continuous Integration.



Source: (FOO, 2016).

This process makes it easier to find and remove code bugs, because it searches for fails in each change made. Besides, bugs are cumulative, so it is not fixed before

continuing development, it can affect the new part of the code and generate more errors.

2.3 TEST SELECTION PROBLEM

Test selection problem is a well-known and central topic in software development with the introduction of agile techniques that are increasingly reducing development cycles. Continuous integration, in this context, faces this problem directly. Although test selection has already been addressed in the literature, and many research projects have gone as far as industrial deployment in this field, this remains a subject far from being completed. As an example Test case prioritization is an issue widely studied in regression testing (ROTHERMEL *et al.*, 2001), this means that a test set is selected among a group of already executed test cases to validate existing functionalities. This problem is described mathematically as: given a test suite TS , a test set PT formed by all possible permutation of TS , and a function Q that measures a test set performance; a permutation TS' of TS need to be founded such that $Q(TS')$ is maximized, that is,

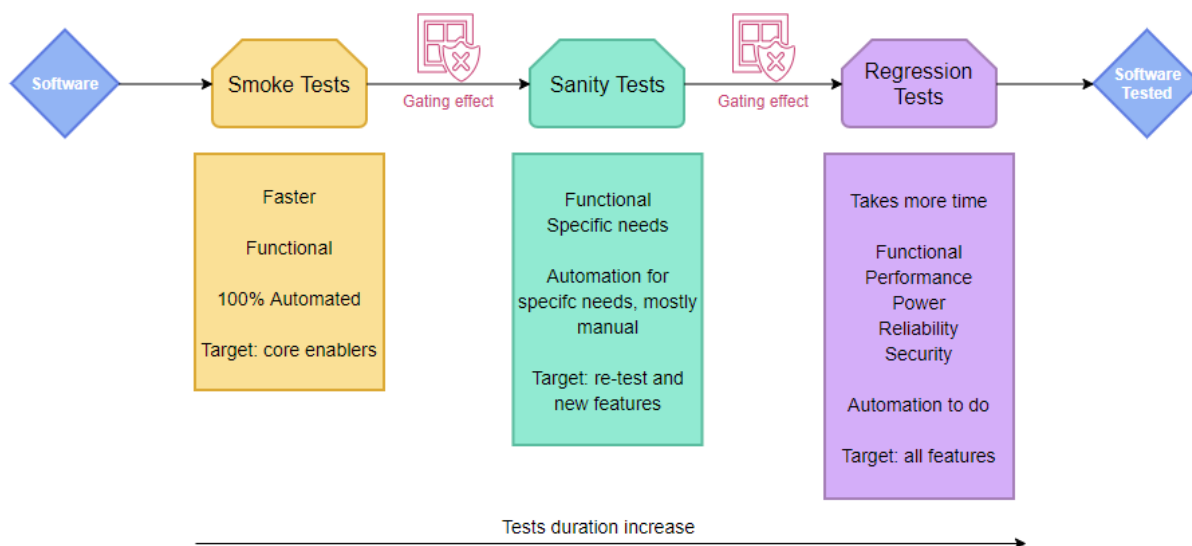
$$\forall TS \in PT: Q(TS') \geq Q(TS) \quad (1)$$

Although the prioritization problem is important, it is not the only one faced in this project. Here the aim is to select tests among all test cases, so it can be implemented more generically. Given a software to test, a test plan needs to be created in order to find the largest number of failures as soon as possible. The integration test is performed at Renault Software Labs in 3 stages like in figure 6: smoke, sanity and regression test. The first stage called **Smoke Tests** is a small set of tests targeting enabler's functionalities to gate testing of strongly flawed software. Smoke Testing is a kind of Software Testing performed after the software build to ascertain that the critical functionalities (enablers) of the program is working fine. Smoke tests must be performed on each integration cycle. This applies to new development and major and minor releases of the system. This can applied also at each patch submitted by the developers.

The second stage is called **Sanity Tests**. Is also a small set of tests but selected according to the integration plan and debug/fix activities. Sanity testing is a type of software testing performed after receiving a software compilation to check for bug fixes and new features added in the integration cycle. The goal is to determine that the proposed functionality works roughly as expected. Sanity tests will change on each build (needs to be planned) by targeting bug fixes and new features. In this stage tests can be automatized or manual.

The last stage is the **Regression Tests**. This stage is an extract of the full test plan to cover all the required tests for the integration cycle. The aim here is to find if the new software parts (all patches include) generates a functional regression on the

Figure 6 – Integration testing cycle.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

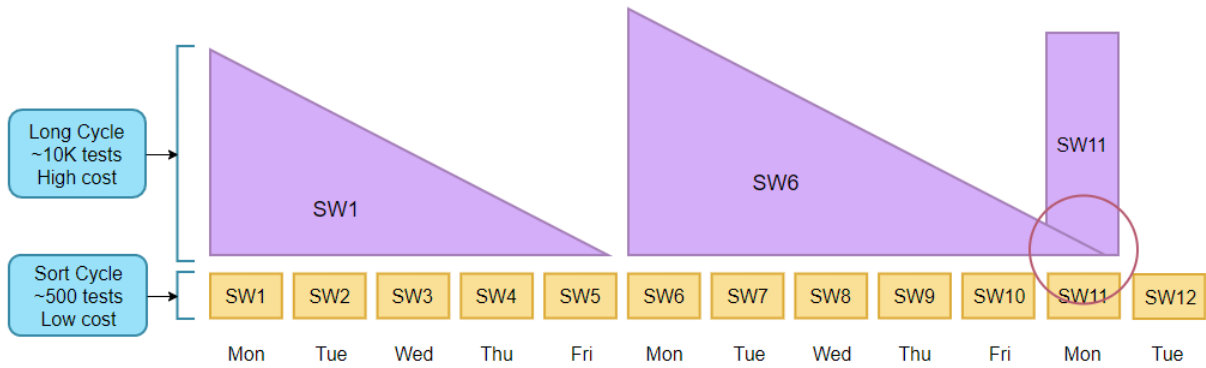
system. The number of regression tests became high according to the life cycle of the system. More the system becomes complex, more the number of regression tests is high and the duration to perform this amount of tests is long. The test duration and complexity increase in each step, it means that regression tests take more time.

Generally, the execution timing of these tests sets (called test campaign) depend on the complexity and the time reserved to validate the system. Renault Software Labs clocks these test campaigns like this: every night smoke tests and automatic sanity tests are launched. A small part of the automatic regression test is performed. These nightly tests are called short cycle test campaign. These tests are performed on the latest build generated at 1:00 AM. This short cycle takes 1 to 3 hours and the cost to perform them is reasonable.

A longer test campaign is performed more sparsely, like at each start of the week, including all the system's test catalog. This long cycle takes time to perform (some days) because it aims to validate that all system's parts, which became more and more complex, doesn't have any flaws. So, several numbers of test is performed, it can be approximately 10000, so it may not have ended when a new cycle should begin. The cost is also high because, not only the time to perform is long, but also this kind of test includes more complex testing material, such as GPS simulator, multi-system interactions, Multimedia actuators, and others. Figure 7 shows a clocking example for these two cycles:

The idea of selecting tests is to find the right regression test that can be performed into earlier steps in aim to catch errors as soon as possible and to reduce the numbers of long cycle test campaign. For the previous example, one long cycle every

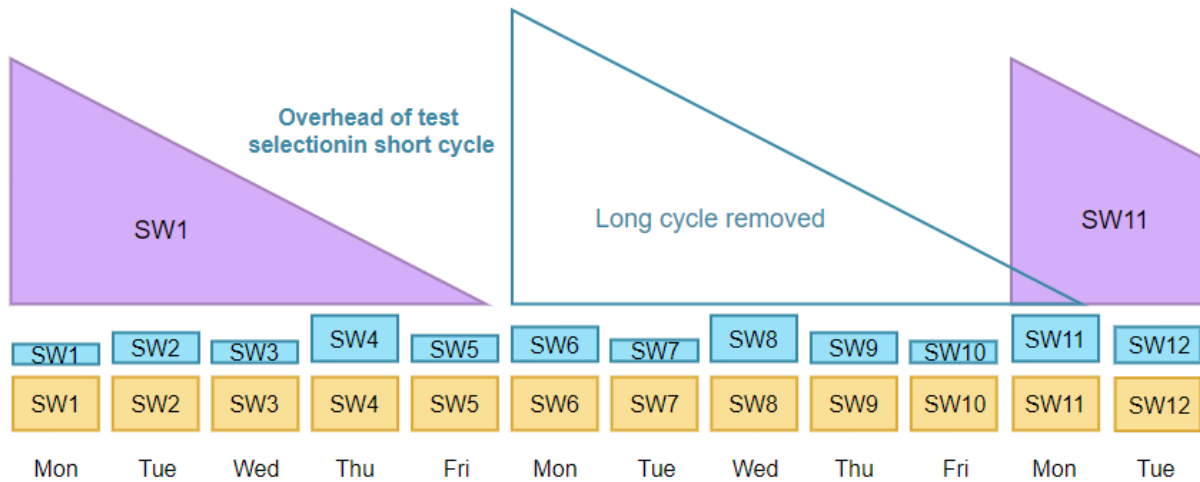
Figure 7 – Test cycle example.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

two can be removed as shown in figure 8. Doing this, a long cycle schedule can be more sparse and it lets more time to perform some complex multi-systems tests that cannot be done in just one week.

Figure 8 – Test cycle redistribution.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

In continuous integration software development, as the system is built, more integration tests are added, keeping the development and test cycles shorter. However, the system to be tested becomes more and more complex and the test times are longer. This is the challenge faced in this project.

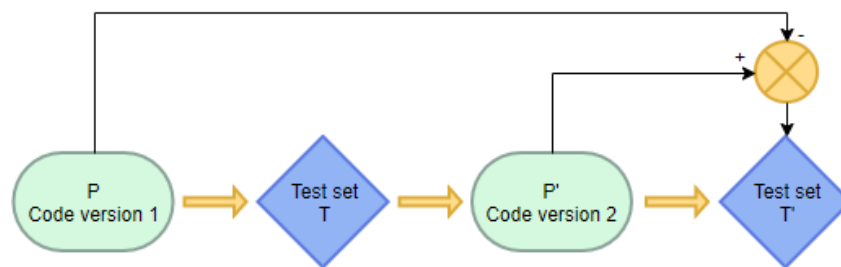
3 STATE OF ART

A study on test selection theme was carried out for the beginning of the research and three main approaches were founded: a changed based, a black box and a reinforcement learning approach. They will be explored in this chapter, thereby related works can be analyzed and compared.

3.1 CHANGES BASED APPROACH

The most common approach in the literature is a changed based test selection for regression testing. It is based on the analysis of what has been changed in the software that is being tested in comparison with the precedent commit. Let's call P the state of the software in the first sprint, where a test set T was applied. In the second sprint, the state is changed to P' , so the algorithm will analyze the difference between P and P' to select a new set T' to be executed, like the scheme of figure 9.

Figure 9 – Changed based test selection.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

A solution for the test prioritization problem, mentioned in chapter 2, is the mean objective in this case. However, the order of testing is also important, introducing here a sorting problem. In (RUTH *et al.*, 2007), an Control Flow Graph (CFG) was used to determine the relations between changes, where nodes represent code entities (statements, methods, or components) and edges represent the control flow between entities. With the graph constructed, it is possible to determine which test case covers which entity by testing it. A comparison between P and P' is made so dangerous edges can be highlighted, that is, entities that may behave differently in P' . Based on this information, tests case are selected to be rerun.

This approach has good results when applied in unit tests or in a test that validates an accurate and known functionality. However, for integration testing, this technique requires knowing the order in which tests are performed and, therefore, re-testing the campaigns several times to determine the optimal order for a given software patch. This is very expensive at test run times and it is not always possible to change

the order of test executions, so it can not be well adapted in Renault Software Labs. That is why it won't be much explored at the moment.

3.2 BLACK BOX APPROACH

Another way to select tests is based on past tests runs results, however, a lot of data is required to do so. For a specific code that needed to be tested, machine learning algorithms can select a test set that had a good result when used to test a similar code in the past. This type of analysis is called *black box* here because the intern process of selection is not known, only inputs and outputs are explicit. Although the use of machine learning algorithms is not yet widely explored in this field, it is a promising approach, since computers don't think like human beings, thus they can find patterns and links that an expert could not.

This method is widely required since the source code is not always available in the testing phase. Supervised machine learning and natural language processing are used in (LACHMANN *et al.*, 2016) to prioritize test cases to be executed in regression testing. Test experts decisions are used to rank a training set of test cases and then machine learning tries to imitating the expert's behavior. The words frequently found in the description of steps to be taken in a test form a dictionary, thus information about each test case can be learned, such as requirements coverage, revealed failures (failure count, age and priority) and test execution cost. However, for each new code functionality, a new training data need to be created to learn a new classifier and experts are needed in creating its labels.

In (GÖKÇE; EMINLI, 2014), tests case are classified using a Multi-Layer Perceptron Neural Network (MLP NN) in aim to determine which test cases are more critical and need to be executed first, increasing the possibility of finding faults earlier. Five classes are defined to cluster test cases using a fuzzy scale, they are: very high priority, high priority, middle priority, low priority, very low priority. A test set is used to train the neural network, and then another set is used to test if it can cluster test cases by itself.

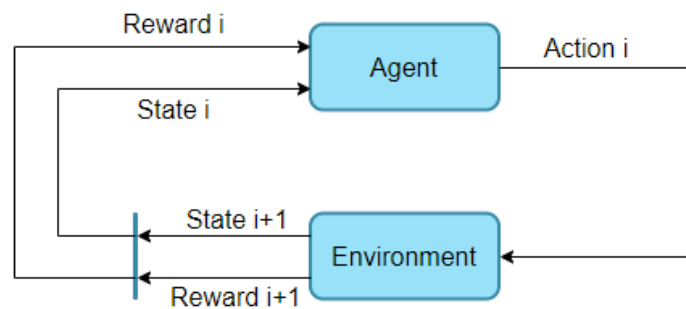
There are also heuristic approaches based on historical data that can be classified as *Black Box* methods because they don't require access to the source code. One example is the work developed in (NOGUCHI *et al.*, 2015), where an ant algorithm is used to prioritize tests case by their historical performance. With a fitness function, test cases can be classified, and then a heuristic method can be applied to select the best combination in aim to maximize the test set fitness value.

3.3 REINFORCEMENT LEARNING

Reinforcement learning is an artificial intelligence method based on oriented objectives that learns by experience feedback. It can begin with randoms actions to learn

what works and what doesn't and this learning is done by incentives. So the algorithm receives a reward every time it makes a decision considered correct and is punished for erroneous actions. Figure 10 shows the five main terminology in reinforcement learning: an agent that will learn in the process; an environment where the agent's actions will be performed; an environment state that will serve as input information for the agent; a reward that will encourage or discourage the agent's decisions; and an action (MONI, 2019).

Figure 10 – Reinforcement learning schema.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

This approach takes more time to adapt, it will do a decision, execute the tests selected and then take feedback to adjust its previous decision. It was used to prioritize and select tests case in a continuous integration development by the authors of (SPIEKER *et al.*, 2018). As the code evolves over the CI cycles, they propose to solve an adaptive test case prioritization problem applying a reward function that evaluates a test schedule's performance created by the agent. Their method is model-free and doesn't require access to the source code of the program to be tested, however, it takes time to adjust and a significant amount of data needs to be stocked.

3.4 COMPARISON

As seen in the previous sections, there are several ways to approach the test selection problem, which is a subject to be studied in more depth and careful way. To sum up, a comparison between these three approaches can be explicit in a table format to a better understanding, see table 2 below.

As seen in the previous chapter, the last two methods will be further explored. For that, a history of Renault's test executions must be collected, which is the subject of the next chapter.

Table 2 – Approaches comparison.

Approach	Data required	Feedback	Methods	Referances
Changes Based	Source code	No	Statistic and optimization	(ROTHERMEL <i>et al.</i> , 2001), (RUTH <i>et al.</i> , 2007)
Black Box	Tests case descriptions and /or past results	Optional	Machine Learning and optimization	(GÖKÇE; EMINLI, 2014), (LACHMANN <i>et al.</i> , 2016), (NOGUCHI <i>et al.</i> , 2015)
Reinforcement Learning	Tests case descriptions	Mandatory	Artificial Intelligence	(SPIEKER <i>et al.</i> , 2018)

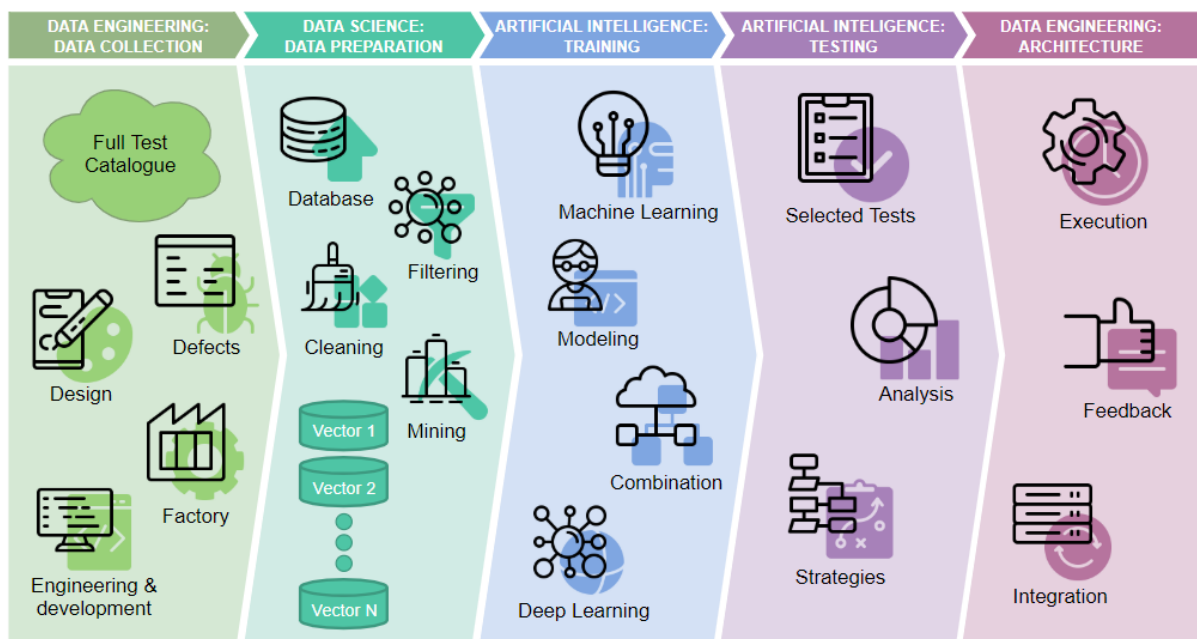
Source: BRZEZINSKI MEYER, Maria Laura (2020).

4 PROJECT OVERALL

The testing phase is crucial in every product development, finding defects cannot be seen as a harmful result, but as an opportunity to improve quality and customer experience. As explained in chapter 2, it is important to find errors as soon as possible, giving more time to correct them and, doing so, increase its reliability. That's why continuous integration is implemented, even if efforts are needed to set the right tests to be executed. In this context, the search for automatic test selection methods is extremely important and it is becoming a broad field of study with artificial intelligence applicability.

This project has the aim to contour the time and cost challenge shown in the previous chapter and it was divided into five phases as can be seen in figure 11: data collection, data preparation, AI training, AI testing and architecture.

Figure 11 – Project overall.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

First of all, data needs to be collected to see what was going on in past test executions, this is the data engineering phase for **data collection**. Among these data and knowledge, there are:

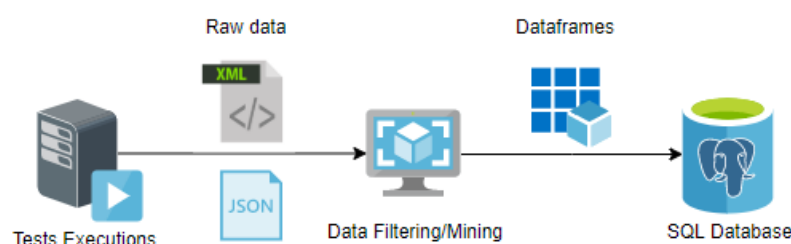
- Test case catalog: composed by all Renault's tests, including descriptions, domain, environment, requirements, steps to be reproduced, level, and others test's parameters;
- Defects: all software bugs reported, whether they are already fixed or not, their priority and the way they interfere with the functioning of the system that was

being tested;

- Design: understanding of the creating tests process and using of experts experience in the field to know how to analyze tests results;
- Factory: development method knowledge, division of Renault's projects and tests case applicability in each development step and for each project's domain;
- Engineering and development: data about test plans and executions, as well as tests run results obtained by tests case from the catalog.

All data collected in the previous phase is considered raw, thus a **data preparation**, using data science methods, must be done. When data is collected from other platforms, it is gathered in a data-interchange format, like JavaScript Object Notation (JSON) or Extensible Markup Language (XML), as will be explained in chapter 5. The features that stand out in the data are selected, cleaned and passed to the numerical format, creating data vectors to be used as input of the algorithms to select the tests in the future. In this step, data mining is also done, which is a process where large amounts of data are explored in order to find consistent patterns, such as association rules or time sequences, to detect systematic relationships between variables. This process is illustrated in figure 12, where data frames are a two-dimensional array-like structure in which each feature is represented in columns and each row is an object, for example, one test case per row.

Figure 12 – Data filtering process.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

Data analyzes help to decide how to approach the problem, so that associations between features can be found to help in the future decision. A data mining technique called featurizing engineering is used, allowing to more accurately represent a data structure, to prepare the input that will create the best artificial intelligence model to select tests in the next phase. From then on, the entries will also be in the correct format so that the model learns in the best possible way.

In the next step, **intelligence artificial** algorithms and statistical methods are used to find patterns, cluster the data and make decisions. After studies and data

analysis, algorithms models can be created in order to automate the test selection process. Different models must be created and compared according to their precision, so the hyper-parameters (such as the number of hidden layers in a neural network or number of clusters in a K-Means algorithm) can be better adjusted and tuned.

In the phase called **intelligence artificial testing**, the models created previously will be put into practice, that is, for a given code, a set of tests will be selected and executed. At this stage it is very crucial to maintain feedback and a results record, so the algorithm can improve during the process. The results will be analyzed by specialists in software tests so that they can be validated. In this way, improvement strategies can be taken, as well as adjustments to the models and inputs.

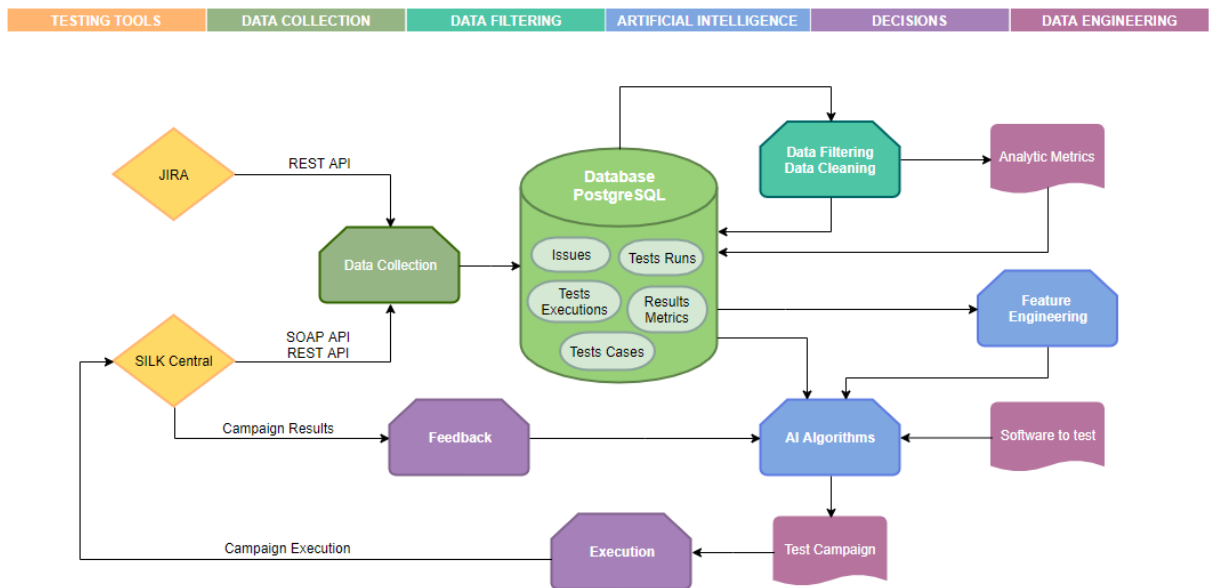
At the end, another data engineering phase must be done, the system **architecture**. In this phase test planning and execution process will be automated, so that after the selection of a test campaign by the algorithm, the selected tests will be executed and final results collected. This process counts on the help of tools already used in the company for planning and executing automatic tests, like the *Silk Central*, *Jira* and *Git Lab* platforms, where Application Programming Interfaces (API) helps to execute actions through requests in a script format.

Despite the division of the project into stages, which helps in the development of the research, the phases previously described are extremely dependent and complementary to each other and some share similar techniques. As the project is complex, the goal of the internship was focused on the first steps, that is, on the collection, analysis and filtering of data. As will be explained in the following chapters, artificial intelligence algorithms and statistical methods were also used to assist in the featuring engineering step.

To sum up the project, a pipeline was created, as it can be seen in figure 13. All the phases described above are represented in it and separated by colors, the arrows show dependence between processes and the testing tools Jira and Silk Central are also present.

First it is important to know the approaches already followed in the test selection domain. Therefore, in the next chapter, the solutions proposed in articles to solve this problem, previously explained in chapter 2, will be studied. Next in the chapter 5, the data collection phase will be better detailed. The data format (JSON, XML and data frames) and the tools from which they are extracted (Silk Central and Jira) will also be explored. After, the work done in data preparation phase will be addressed in chapter 6.

Figure 13 – Project pipeline.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

5 DATA COLLECTION

Data collection is a very important phase for any project that involves artificial intelligence, as it is through these collected data that the machine will learn to perform the desired function. A large amount of data is required in the construction of a model in machine learning, as the algorithm will learn through patterns found in these data representing previous behaviors that need to be copied and improved.

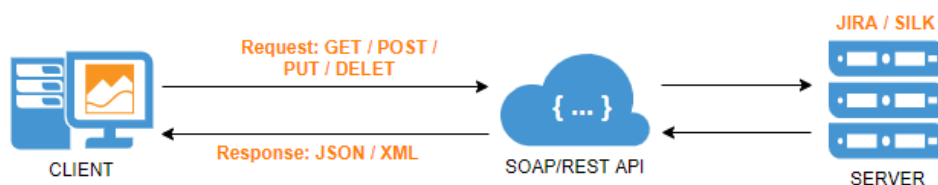
In this context, the term big data is introduced, used in computing science for large data sets that need to be processed and stored for future use. Big data main aspects are usually described with the five V's: volume (referring to data amount), variety (different data types mixed), veracity (data authenticity and availability), velocity (the speed at which new data is created) and value (regarding the data usage).

All the software testing phase information is collected from two platforms, initially the data came from the Jira tool and a new platform, Silk Central, is currently being implemented. The structure in which the data is stored on both platforms will be discussed in this chapter. In addition, how data collection is done and the formats of data will also be explained.

5.1 DATA FORMAT

Before collecting the data, it is necessary to identify the formats in which they are transmitted. The most practical way to request data from a server is through an API, which is a set of routines and standards established by a software for the use of its features by external applications that do not intend to be involved in implementation details, but just want to use their services. This way, it doesn't matter the language in which the platform containing the data was built. When a request is made to an API, an answer returns in a JSON or XML format, like illustrated in figure 14.

Figure 14 – API usage.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

JSON (JavaScript Object Notation) is a Java Script based data-interchange format, it is simple to be read by human beings and to be parsed by machines. It is an object-centric standard since it intends to describe an object's parameters. JSON is a lightweight format constructed by keywords and contents separated by curly braces, like:

`{"name": 'John', "Age": 10}`, `{"name": 'Mary', "Age": 15}`. Thus it is simple to find all values that describe people's age in the previous example. JSON standard is described in (CROCKFORD, 2000), where a schema of each data type structure can be founded.

Extensible Markup Language (XML) is a widely used sub type of the international standard SGML (Standard Generalized Markup Language) and it was developed in 1998 by World Wide Web Consortium (W3C). It is typically used in complex data representation that tends to be document-centric. XML data are organized in a hierarchical form using tags, and it must always begins with a single root tag that contains all the other tags in the file. Taking up the example used to describe JSON format, in XML "age" can be a *child* of "name", like: `<People> <John> <Age>10</Age> </John> <Mary> <Age>15</Age> </Mary> </People>`. More information about this standard can be obtained in (W3C, n.d.).

A Data Frame is a structure derived from the *pandas* library (NUMFOCUS, 2015) for python programming language used to manipulate data more easily. It is a 2-dimensional labeled structure formed by columns with potentially different types that describe each row. For the same example used in the two other formats, a Data Frame is like:

Figure 15 – Data Frame example.

```
In [8]: import pandas as pd
        pd.DataFrame({"Name": ['John', 'Mary'], "Age": [10, 15]})
```

Out[8]:

	Name	Age	Data types:
0	John	10	Name object Age int64
1	Mary	15	dtype: object

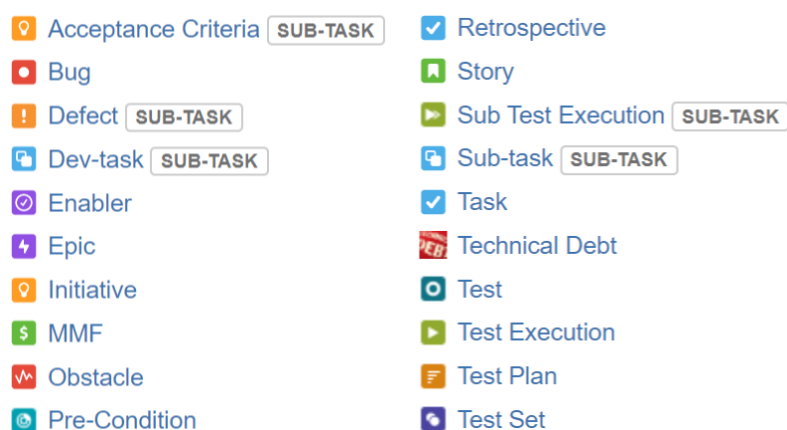
Source: BRZEZINSKI MEYER, Maria Laura (2020).

5.2 FIRST DATA SET - JIRA

Jira is a platform developed by *Atlassian* that helps teams and enterprises to plan, assign, track, report and manage work (ATLASSIAN, n.d.[a]). Currently, all activities are planned with the help of Jira in Renault Software Labs. In this way, the creation, planning and execution of all tests are monitored through this platform, making it possible to know which test was used to validate certain components developed in a project and the result of it. Such structure allows the use of the agile development method in continuous integration, formed by cycles of planning, development, test and improvement.

In Jira, all tasks are managed as *issues* like tickets and their types can be seen in figure 16 below:

Figure 16 – Issues from Jira Project Settings.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

In figure 17 the main issues types are described. Four of them are directly connected to Jira and are organized like this: an *epic* encompasses several issues of the type *story* and / or *tasks*, which can be blocked by a *bug*. The other four are created by a plug-in called Xray, that is a complete test management tool that transforms Jira's issues. It supports the entire testing life cycle, that is test planning, design, execution and reporting. For each issue, there is an *workflow* where the progress to solve it is traced.

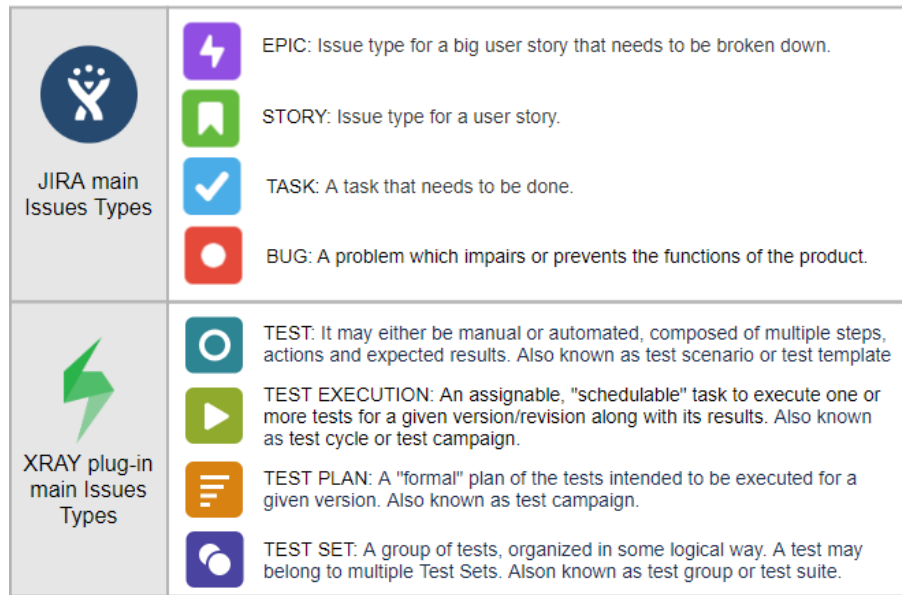
For some issues implementation (Test, Pre-condition, Test set, Test execution and Test plan), the plug-in uses Jira tickets, so entities can be created or updated using Jira's native API taking into account the custom Xray fields. Requests can be done to this API using OAuth 2.0 authorization, which is an industry-standard protocol developed by the IETF OAuth Working Group.

In addition, there is an API for Xray, named *raven*. It provides additional endpoints specifically designed to handle test fields and entities. It is from *raven* that results from running automated processes can be exported.

The two APIs used to collect data from Jira are based in a representational state transfer (REST) structure, more about it and a comparison with another structure will be described in the next section. For more information on the use of such APIs, see the documentation for Xray in (ATLASSIAN, n.d.[c]) and documentation about the cloud platform of Jira in (ATLASSIAN, n.d.[b]).

For this project, data from five tickets types will be necessary, they are: tests case, test plan, test executions, test runs and bugs. Each test case describes all the steps for such a test to be carried out, as well as the required parameters and environment. The

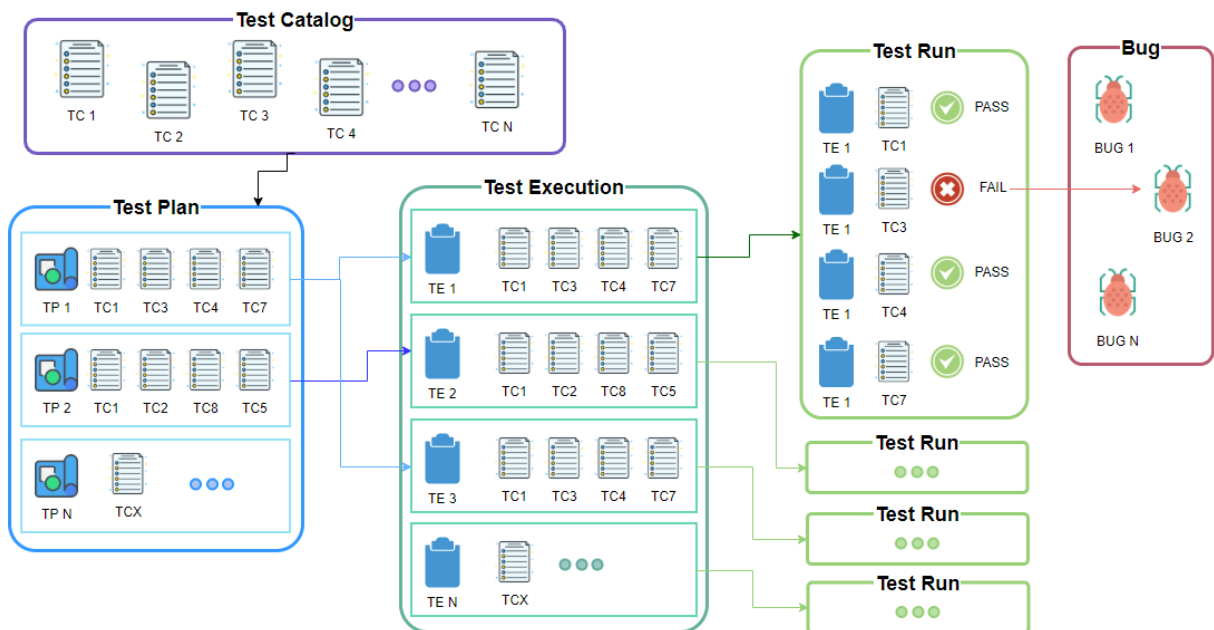
Figure 17 – Main issues types.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

set of tests case form the Renault test catalog, from where the tests will be selected to form a test plan. A test execution is created when a test plan is executed, each test case execution forms a test run with a result. If an error is founded during testing, a ticket but is created to be fixed in the future. Figure 18 illustrate these data structure:

Figure 18 – Jira data structure.

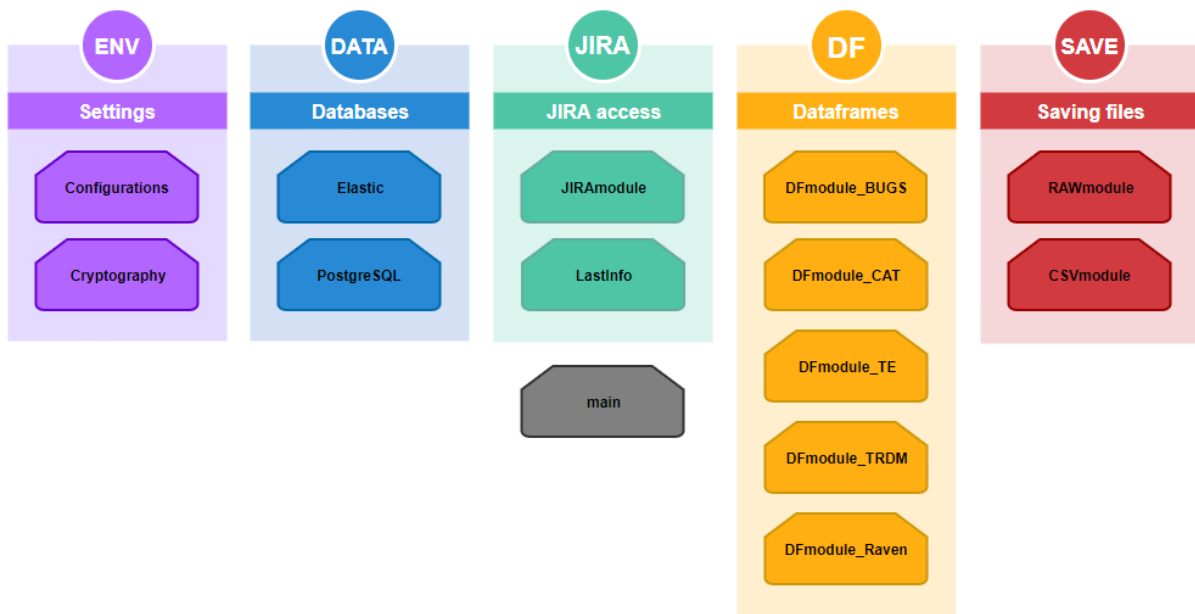


Source: BRZEZINSKI MEYER, Maria Laura (2020).

To collect these data, a script is executed daily by the configuration file *crontab*, which specifies shell commands to be executed periodically through a given schedule. A first version of this script was developed by the previous intern Juan Martinez Gil (GIL, 2019) and the mentor Fernand Cuesta.

During this work, the script for the collection was adjusted according to data acquisition interests, and also improved aiming saving time and code re-usability, resulting in a modular project structured like in figure 19.

Figure 19 – Structure of the Jira data collection code.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

Some performance improvements were also applied, like executing some tasks in parallel and updating the database using SQL commands instead of dumping the entire table each time. The job previously performed in 4 hours, now takes approximately 15 minutes. Besides that, passwords and users used to logging into databases and Jira are now encrypted. A new git directory was created, where a test pipeline is used to verify the main code. Collection is done as follows:

1. Settings: where all variables are set, such as passwords, URL links to servers, saving directories, and other variables for internal uses;
2. Jira access: a request is made to take all new data, that is, all tests executed in the current day. These data is received in a JSON format and taken using the two REST (Representational State Transfer) APIs described previously;
3. Dataframes: relevant data is extracted from the raw data collected previously, forming five DataFrames, for issues information (*DFmodule_BUGS*), tests case (*DF-*

module_CAT), tests executions (*DFmodule_TE*), tests runs (*DFmodule_Raven*) and test results metrics (*DFmodule_TRDM*);

4. Saving files: the JSON raw data is saved locally, and the five Data Frames are saved locally and in a SQL based database (*PostgreSQL*).

There are two table types in the PostgreSQL database for each data frame created. The first one is used to track all modifications made in the issues collected, so there is a field to indicate whether this data is a new ticket created or is the update of an existing one. The second one is used to keep only the most recent information about each issue, it is called *Last Info* in this project.

5.3 NEW DATA SET - SILK CENTRAL

Silk Central is a test platform developed by the company *Micro Focus* that helps in the management of quality control projects (MICRO FOCUS, n.d.). It uses collaborative testing, making it possible to develop in continuous integration. Thus, all activities, progress and results are accessible to the entire team. Both automated and manual tests can be executed using this management system, and, after execution, reports containing specific desired data can be generated.

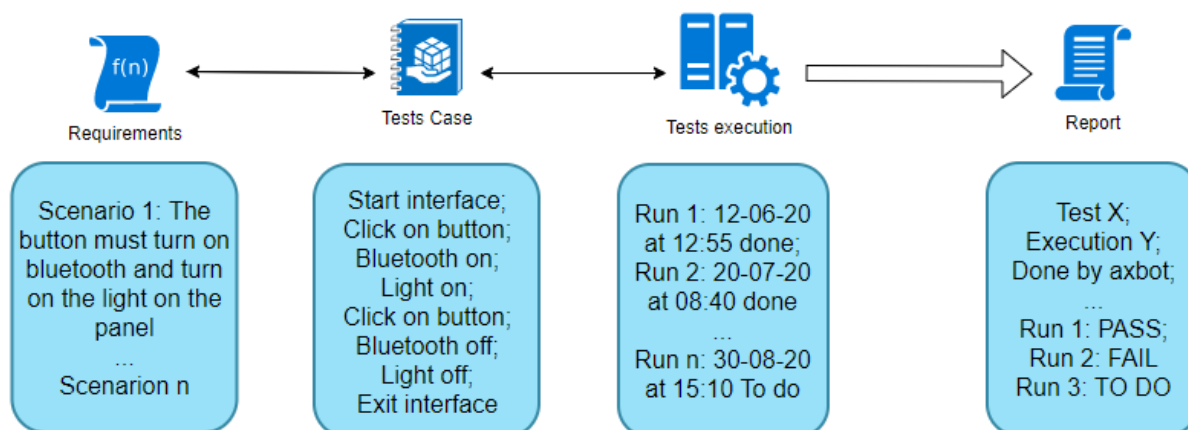
Data organization is different from Jira, there are projects in Silk Central and each one is composed by folders. One folder represents a domain, that is also divided into sub-domains. Thus one test can be in more than one sub-domain, depending on the need to use it.

There are four main elements in Silk Central structure: test cases, test executions, requirements and reports. Tests case are the same as in Jira, they form the Renault test catalog and are defined by many parameters, like a description of steps, environment to execute, domain, version, type, level, and others. Test execution represents a campaign, which is a set of tests case, that was executed in a certain product, thus results are provided by each execution. The requirements are linked to test cases, they describe all tests particularity to be executed. Finally, reports can be created with specific test results from the test executions. Figure 20 illustrates the structure described with a simple example.

In this platform, data collection can be done using two types of web drive protocols:

- Simple Object Access Protocol (SOAP): it is a function-driven protocol based on XML, since it transfers structured information. It acts through an envelope, that is, defining the message's content and informing how to process it; determining a set of rules for data types; and adjusting the layout for call and response procedures.

Figure 20 – Silk Central data structure with example.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

- Representational State Transfer (REST): created in 2000 by Roy Fielding, is it a data-driven architectural style since its main function is to access a resource for data. It is a faster, simple and lighter way to convey data then SOAP because it doesn't require an envelope structure. Normally data is in a JSON format.

The data storage procedure is the same as for Jira, after the collection, Data Frames will be generated and stored in a database. This application is still under development together with a team responsible for the integration between the automated test execution platforms (such as Git and Artifactory) and those for managing executions (such as Silk Central and Jira).

In this project, two collect approaches are implemented to explore all information that Silk Central could provide. The first one is the use of the *Requests* library (REITZ, n.d.) for python in order to make *GET* requests to Silk's API, the result of this development was a simple script that facilitates SOAP's collection and a page on the company's wiki. The second approach was done using *Selenium* web driver (APACHE, 2018) for python with google chrome driver (GOOGLE, n.d.). This script simulates a human action: it opens a google chrome, reaches the Silk Central site, log in, searches for the information using pre-determined filters from Silk and download it.

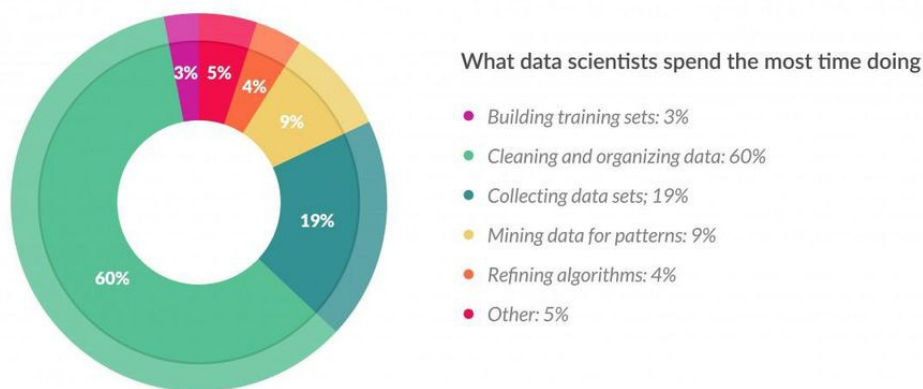
6 DATA PREPARATION

For artificial intelligence algorithms to be able to learn through data, they must be arranged in the correct format. That is why a preparation phase is needed, where data is filtered, cleaned and mined. According to Gil Press survey about big data in Forbes magazine (PRESS, 2016),

"Data scientists spend 60% of their time on cleaning and organizing data. Collecting data sets comes second at 19% of their time, meaning data scientists spend around 80% of their time on preparing and managing data for analysis."

Figure 21 shows the proportion of each step in data preparation.

Figure 21 – Time spent by data scientists.



Source: (PRESS, 2016).

As mentioned in chapter 3, the data preparation phase includes data cleaning, filtering and mining. Data cleaning is a process in data science that aims to detect corrupted or inaccurate records in a data set, table, or database to correct or delete them. Such term refers to the identification of incomplete, incorrect, inaccurate, or irrelevant parts of the data. Cleaning was done to have the correct information for each test, rearranging the data into data frames.

The term data filtering is linked to the action of allowing only certain data to be selected for later use. This is useful for focusing only on specific information in a large data set or table. Filtering does not remove or modify data. The two main filters used in the project were to filter automated tests from manual ones and to select test cases from *SWCAT-6* catalog, which is all integration tests used in the SIT team (its cycle was described in figure 6).

Data mining is a method that groups tools and techniques capable of exploring a large data set, extracting or highlighting patterns. These tools use learning or classification algorithms based on neural networks and statistics, helping to discover knowledge.

The result can be presented by groupings, hypotheses, rules, decision trees, graphs or dendrograms.

The principle of artificial intelligence algorithms is to make a decision based on information about the data. For example, to find out if a fruit is an apple or not, you need to know its color, size and perhaps even its taste. For a human being, this task is done by the senses, which capture all information so that the brain can decide whether that is an apple or not. For computers the process is similar, but many trials and errors are needed to find out which features are most relevant to model a problem. That is where featuring engineering, a data mining tool, is necessary.

Emre Rençberoğlu enumerate nine feature engineering techniques in (RENÇBEROĞLU, 2019): **Imputation** to handle with missing values; **Handling Outliers** to don't have aberrant or atypical value in the set; **Binning** to categorize data and avoid overfitting; **Log Transform** to adjust data magnitude order; **One-Hot Encoding** that enables to transform qualitative data into binary; **Grouping Operations** to handle with data groups, **Feature Split**, **Scaling** data, and **Extracting Date**.

Therefore a study and an analysis of the data collected from Jira was done to better understand how it can be used to select tests to be executed in a project. Besides, some features were better explored and analysis of the importance of each feature was made. The book Python for Data Analysis (MCKINNEY, 2018) helped with the data manipulation using the *pandas* and *NumPy* libraries.

6.1 THE DATABASE

For integration tests, there are 2000 test cases including different versions of 1500 unique test. These tests were executed 40000 times in the past three years, about 20000 executions if those linked to tests that are now obsolete are excluded. The main parameters for each test can be seen in figure 3.

Table 3 – Tests main parameters.

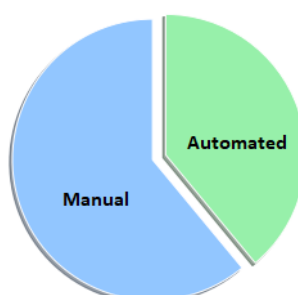
TEST CASE						TEST RUN			
Summary (unique)	Key	Status	Domain	Creation Date	Level	Bugs/ Defects	TE Keys	TR Status	First/Last Execution
Title or name of each test case, because they can have more then one version	Identifier of each version	Released, accepted, draft, in review, rejected, obsolete	Domain, sub-domain, meta-domain	Date when each test was created	Test level	Number of bugs	Identifier of all test executions that use this test case	Pass, fail, to do, aborted, blocked, executing	Start date of the first/last execution of this test case

Source: BRZEZINSKI MEYER, Maria Laura (2020).

In addition to the features described in the table above, there is other information obtained, totaling 20 columns of information for tests case, 20 for tests run, 13 for bugs, 30 for tests execution and 18 for metrics (see Jira's feature tables in appendix A).

Automated tests represent about 45% of tests integration data set, the report between automated and manual for integration tests is plotted in figure 22.

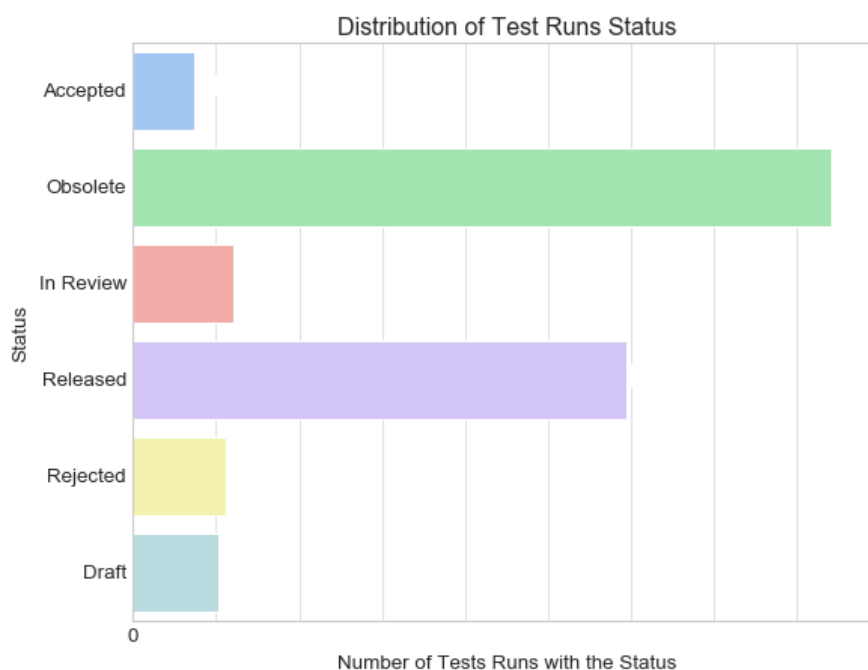
Figure 22 – Automated and manual tests.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

Each test case percentage concerning its status can be seen in the figure 23. Obsolete tests should be filtered out as they can no longer be used, but they can still provide information on past executions to help with decision making.

Figure 23 – Tests case status.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

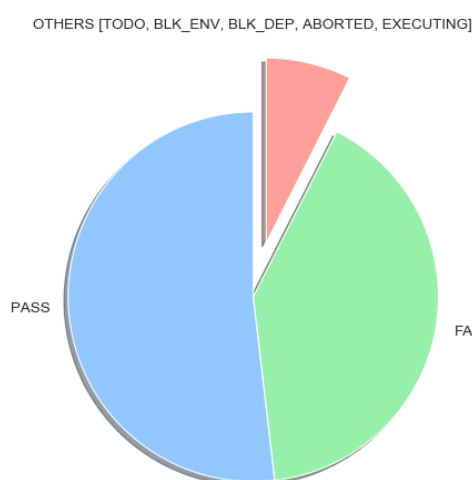
6.2 FEATURES

In this section, five characteristics will be better analyzed with the help of all data preparation techniques mentioned in this chapter's introduction.

6.2.1 Test run Status

The number of times that a test has failed or passed in past runs can be useful in deciding which test to select. It is possible to observe the relationship between the number of executions that passed with those that did not and the other results eventually obtained in figure 24, like blocked, aborted, to do or executing.

Figure 24 – Tests runs status.



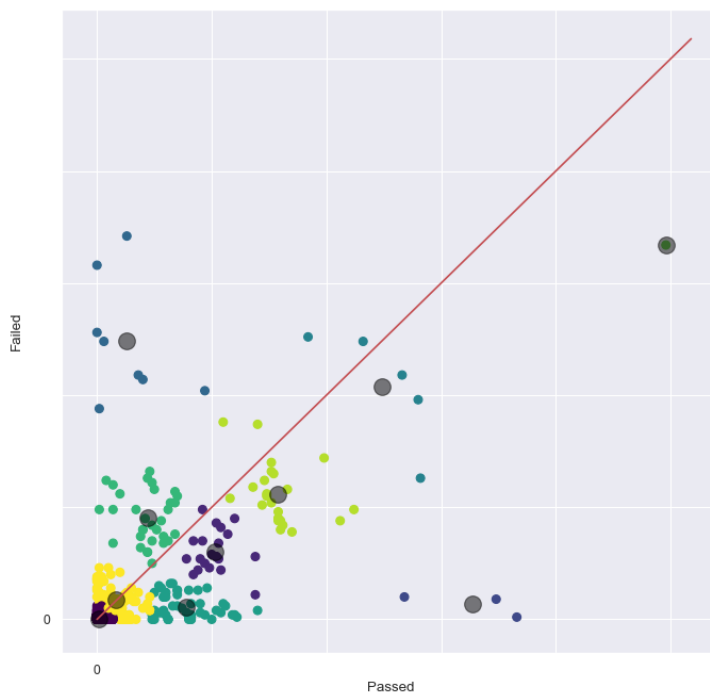
Source: BRZEZINSKI MEYER, Maria Laura (2020).

If a test has already failed a lot in past runs to analyze a certain component, it will likely fail if used again. This failure can mean two things, either the test is good at finding defects or it is not adapted for such components. To help to know which case fits a test, it is possible to analyze the relationship between the number of failed and passed results. In figure 25, an unsupervised machine learning algorithm, K-means, was used to group test cases into ten clusters depending on how many passed and failed results each one had. A line was drawn to separate cases upper to separate those that have more failed results than passed (above the line) and those that have more passed than fail (under the line). Six clusters are above, so the tests that belong to them are more likely to encounter a real error if the result is "fail".

6.2.2 Test execution Duration

The test run duration can provide information about the test's ability to find errors in the software being tested. A relationship between the result of the execution and

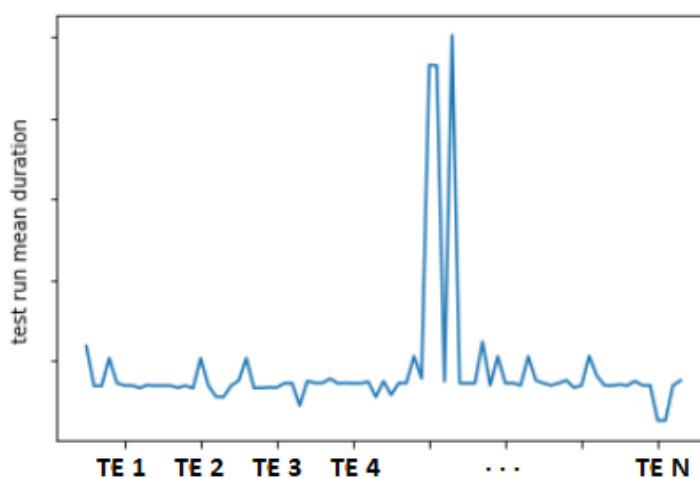
Figure 25 – Clusters PASS/FAIL.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

its duration was found when analyzing all the executions already made for each test case. This relation can be seen in figure 26, which shows the distribution of executions duration for a specific test case.

Figure 26 – Distribution of executions duration.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

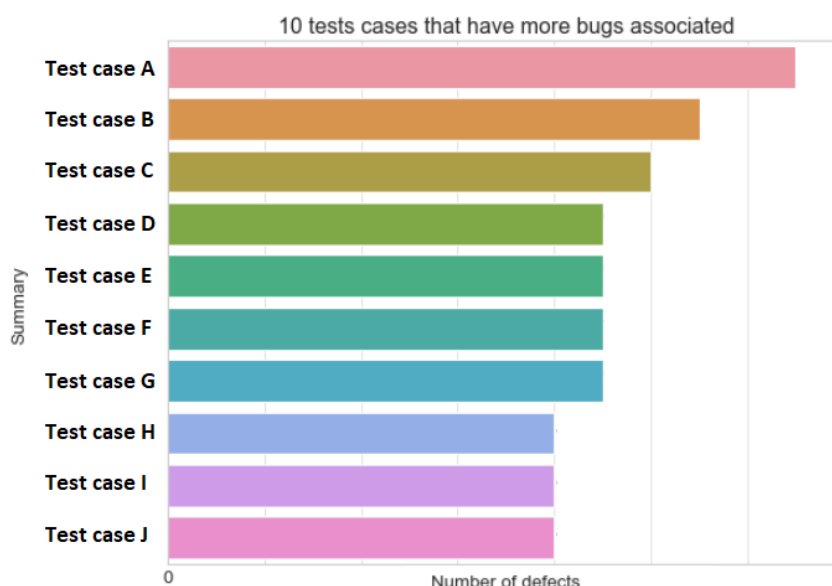
In this case, more than 90% of the executions that took longer than the aver-

age duration to finish have failed the test. Also, more than 70% of the executions that took less than the average duration to finish have passed the test. It reveals a strong connection between executions duration and test result.

6.2.3 Issues funded

It is necessary to remember that if a test fails, it does not mean that a bug was found. That is why the ticks related to each test case must be obtained to check if a bug type ticket is linked to the test in question. The tests case that founded more bugs are listed in figure 27:

Figure 27 – Top 10 tests cases for bugs founded.



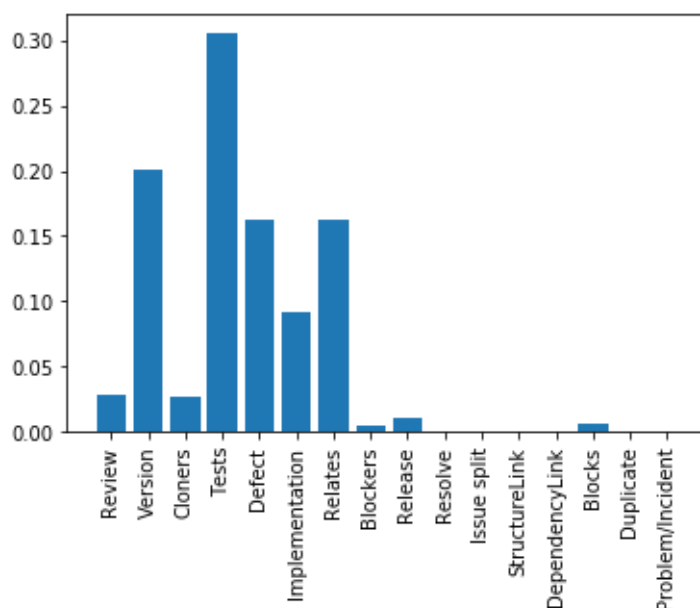
Source: BRZEZINSKI MEYER, Maria Laura (2020).

However, there is another way to find out what tickets were linked to a test case, this field is called *issues linked* and it can be: *Review, Version, Cloners, Tests, Defect, Implementation, Relates, Blockers, Release, Resolve, Issue split, Structure Link, Dependency Link, Blocks, Duplicate, Problem/ Incident*. In figures 28 and 29, a decision tree model was created using issues links as features, then it was possible to find out the importance of each feature in this model to determine if the results would fail or pass. It is possible to notice that the number of defects founded by a test case has a big impact on the final decision.

6.2.4 Tests case Usage

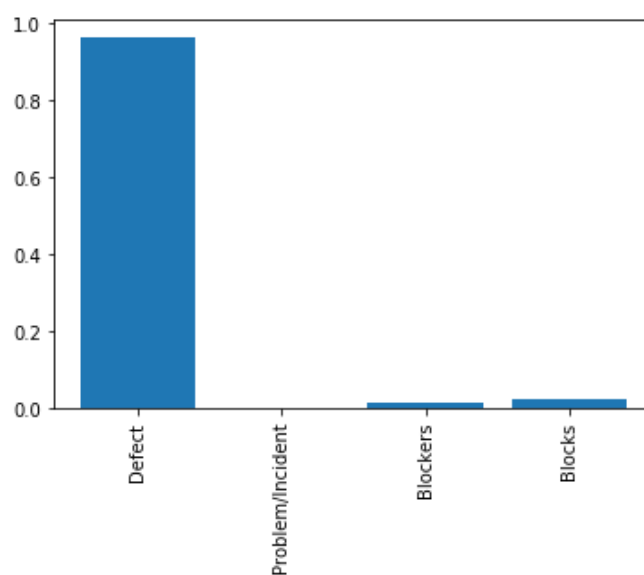
The executions number of a test case does not indicate whether the test was really used, there is no way to compare a newly created test with one created last year.

Figure 28 – Feature importance for all issues linked.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

Figure 29 – Feature importance for defects related issues.

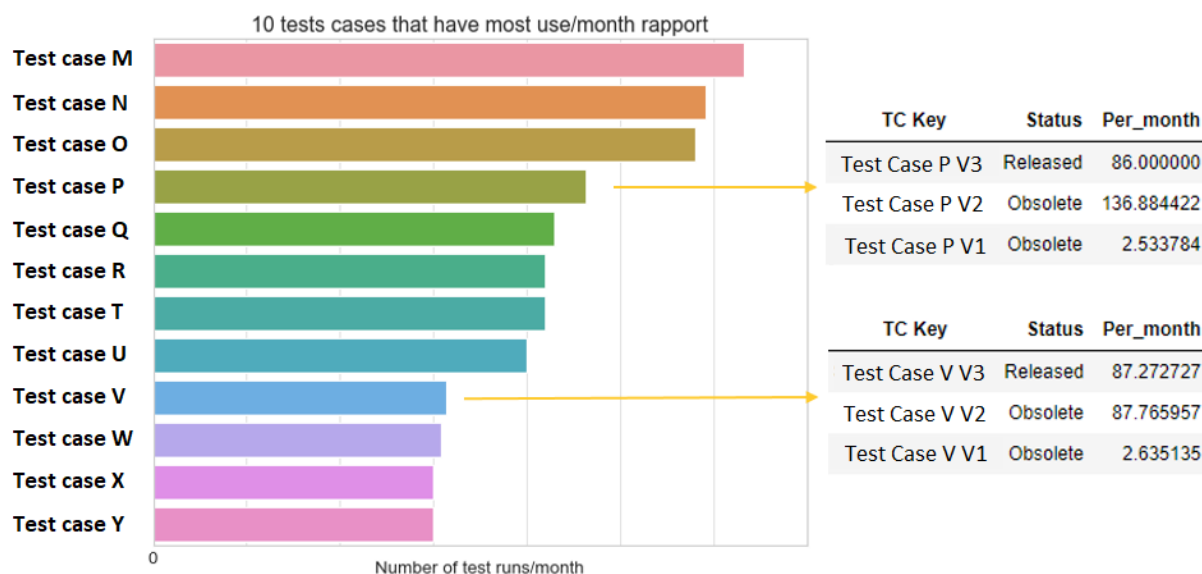


Source: BRZEZINSKI MEYER, Maria Laura (2020).

Therefore, it is necessary to normalize the data. To measure the use of each test, a relationship was made between the number of executions and the released date of each test.

In addition, it is necessary to verify the test case status, because if it was most used in its draft phase, it means that it was tested a lot before realised. The test

Figure 30 – Tests case most used.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

versioning is also important, because it is possible that a new version of a test is not used as much as its previous version. Figure 30 shows an example of two tests, each one has three versions. For the first one, it can be seen that one of its previous versions has been used approximately 136.8 times a month, but its newest version is not used as much anymore. For the second example, it has been used approximately 87.7 time and its newest version is still used as much as before.

6.2.5 Tests case Complexity

One test case may take longer to run than another because it is more complex. In an attempt to measure the complexity of each test, the number of steps performed in each run was calculated and the distribution of such metrics is illustrated by the graph in the figure 31.

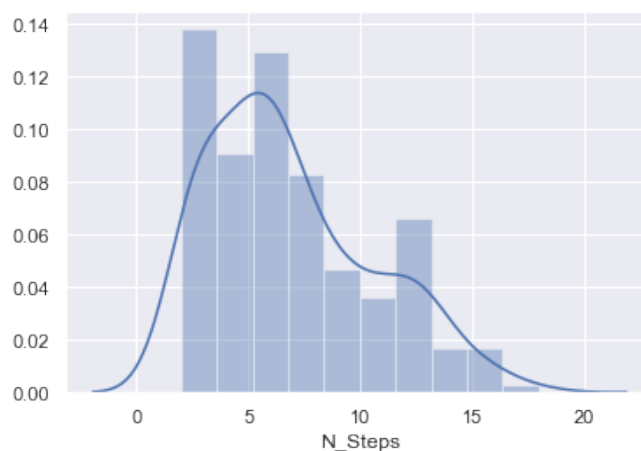
As the tests in this data set are of the integration type, it is normal that they have mostly medium complexity. The smooth line is obtained using kernel density estimation, and with it is possible to see that these tests are composed mostly by about five steps.

6.2.6 First vectors

Using the five features described previously, six vectors can be created. A descriptive statistics is done for these vectors in figure 4, summarizing the central tendency, dispersion and shape of a data set's distribution for these features.

They can be used as inputs for preparing machine learning models in the next stage of the project. The previous sections showed what each vector can provide as

Figure 31 – Number of steps distribution.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

information, and the use of artificial intelligence algorithms can help to combine such information to improve the final decision. Since the analysis of just one factor is simple, however when more than three vectors are used, it is not even possible to graphically visualize their combination.

Table 4 – Vectors descriptive statistics.

	Status FAIL	Status PASS	Duration	Steps	Usage	Bugs
count	300	300	300	300	300	300
mean	60	90	2e+06	6	7000	15
std	70	180	1e+07	3	6000	20
min	0	0	5e+03	1	0	0
25%	20	10	3e+05	2	2000	3
50%	40	50	3e+05	5	6000	10
75%	100	90	5e+05	7	10000	15
max	470	1000	9e+07	15	17000	150

Source: BRZEZINSKI MEYER, Maria Laura (2020).

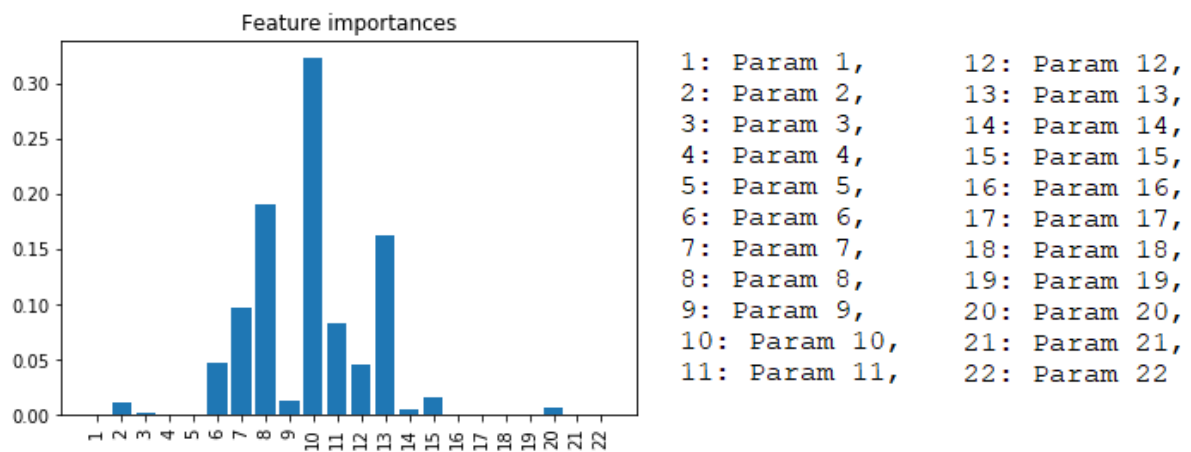
6.3 CATEGORICAL DATA

Most of the data obtained are not numerical, that is, they are words, phrases or even text. Thus, the use of text mining was necessary to separate the data into categories. Text mining is a process used to obtain important information in a text automatically. This information is acquired by developing standards and trends that will be compared with the text.

After mining all information desired, categories can be created using the one-hot encoding technique, which transforms a string variable into *dummy variables*. In

data science, *dummy variables* are numerical variables that represent categorical data, they are normally binary. The use of binary variables is necessary as it is possible to introduce a false bias if each category is mapped as being a number from 1 to 9. For example, for test case status, if the variables { *Released*, *Accepted*, *In Review*, *Draft*, *Rejected*, *Obsolete* } are mapped into { 1, 2, 3, 4, 5, 6 }, the machine learning algorithms will think that there is a distance of 2 between *Released* and *In Review*, but a distance of 3 between *Released* and *Draft*, which is not true. Using the same method as in section 6.2.3 to analyze features importance, it is possible to see the compare the influence of 22 test case features formed by *dummy variables* in figure 32.

Figure 32 – Feature importance for test case feature.



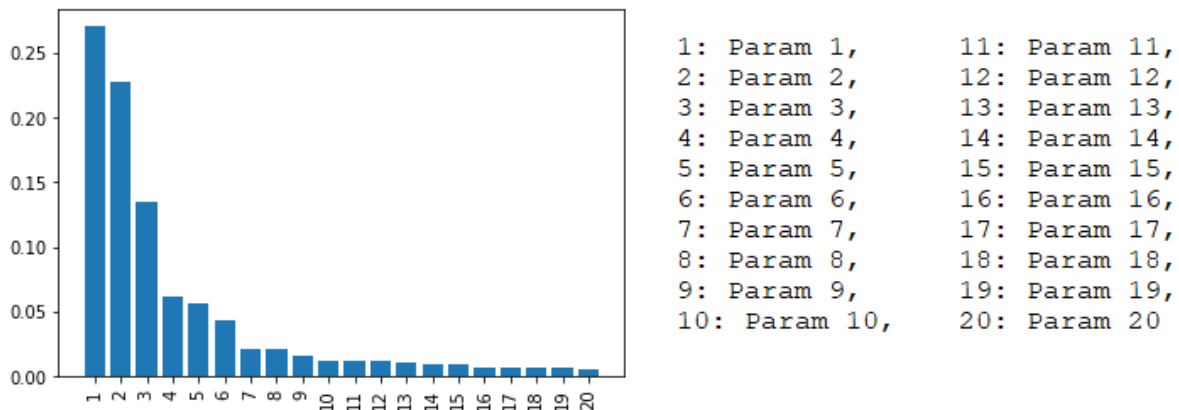
Source: BRZEZINSKI MEYER, Maria Laura (2020).

For each test execution, there are related tags that form a string array, so one test can have multiple tags. In this case it is very useful to apply a function called *MultiLabelBinarizer* from the machine learning library *scikit-learn* (SCIKIT-LEARN: . . . , n.d.). It transforms all tags into columns, so if the test execution has the tag it's value is 1 and if it doesn't the value is 0. Using the same method as in section 6.2.3 to analyze features importance, it is possible to see the 20 tags more relevant in figure 33.

Such analyzes allow the reduction of the number of features used as input to the machine learning algorithms, because, as mentioned earlier, overfitting can occur if many features are used. This means that the statistical model is very frightening to the previously observed data set, but it is ineffective to predict new results.

However some features appear to be very relevant to select pertinent tests, these vectors created with dummy variables are just a first version, more study and refinement is needed before using such data.

Figure 33 – Feature importance for test executions tags.



Source: BRZEZINSKI MEYER, Maria Laura (2020).

7 CONCLUSION

7.1 FUTURE WORK

The work carried out during the internship opened several questions to be explored. The analysis of the features will be continued and deepened so that relevant information is used to assist in the learning. In addition, the other three steps mentioned in chapter 3 will be developed, so the objective of selecting tests to be executed earlier can be achieved. Another important task will be to automate the process of creating test campaigns and executing them. Besides, the feedback of the system with the results obtained by the campaigns executed should enrich the database.

A new field in artificial intelligence is giving promising results, it is called Explainable AI (XAI). It is a method to use explain artificial intelligence algorithms decisions. This technique can be very useful in the software testing context, so humans can understand why such tests were chosen to be executed in a determinate time of development.

Regarding the validation of machine learning models eventually created, it is possible to simulate code defects by injecting errors. Then select tests using the models and compare then using the Average Percentage of Faults Detected (APFD) evaluation method. This makes it possible to check the models most adapted to the problem.

7.2 FINAL CONSIDERATIONS

The problem developed here is complex and wide-ranging, initiating a larger project. The complete development of the pipeline presented in chapter 3 will be done during a thesis with the SIT team, starting in the year 2021. The work carried out during the 6-month exchange allowed an initial analysis of the problem and possible approaches to be followed. In addition, the available data could be better understood and ways to explore them were implemented, as well as the constant feeding of a database that will serve in the future.

The internship enabled the contact with five major areas of knowledge: agile development method, because software development for Renault vehicles are made in continuous integration; software testing, that is where the problem is in; interconnection between programs and applications to collect data; data science to analyze and prepare the inputs; and artificial intelligence, used to attack the problem.

BIBLIOGRAPHY

APACHE. **Selenium**. [S.l.: s.n.], 2018. <https://www.selenium.dev/>. Accessed: 2020-06.

ATLASSIAN. **A brief overview of Jira**. [S.l.: s.n.]. <https://www.atlassian.com/software/jira/guides/getting-started/overview>. Accessed: 2020-06.

ATLASSIAN. **Jira Cloud platform Developer**. [S.l.: s.n.]. <https://developer.atlassian.com/cloud/jira/platform/rest/v3/>. Accessed: 2020-06.

ATLASSIAN. **Xray Documentation**. [S.l.: s.n.]. <https://confluence.xpand-it.com/display/XRAY>. Accessed: 2020-06.

BOOCH, Grady. **Object-Oriented Analysis and Design with Applications (3rd Edition)**. USA: Addison Wesley Longman Publishing Co., Inc., 2004. ISBN 020189551X.

CROCKFORD, Douglas. **Introducing JSON**. [S.l.: s.n.], 2000. <https://www.json.org/json-en.html>. Accessed: 2020-06.

FOO, Daniel. **Continuous Integration and Continuous Delivery with nuget**. [S.l.: s.n.], 2016. <http://danielcoding.net/continuous-integration-and-continuous-delivery-with-nuget>. Accessed: 2020-06.

FOWLER, Martin. **Continuous Integration**. [S.l.: s.n.], 2006. <https://martinfowler.com/articles/continuousIntegration.html>. Accessed: 2020-06.

GIL, Juan Martinez. **Data Science for Test Selection**. [S.l.: s.n.], 2019. Confidential internship report.

GÖKÇE, Nida; EMINLI, Mübariz. Model-Based Test Case Prioritization Using Neural Network Classification. **Computer Science Engineering: An International Journal**, v. 4, p. 15–25, Feb. 2014. DOI: 10.5121/cseij.2014.4102.

GOOGLE. **ChromeDriver - WebDriver for Chrome**. [S.l.: s.n.]. <https://chromedriver.chromium.org>. Accessed: 2020-06.

GROUPE RENAULT Site. [S.l.: s.n.], 2020. <https://group.renault.com/groupe>. Accessed: 2020-06.

LACHMANN, R. *et al.* System-Level Test Case Prioritization Using Machine Learning. *In: 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. [S.l.: s.n.], 2016. P. 361–368.

MCKINNEY, Wes. **Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython**. 2. ed. [S.l.]: O'Reilly, Oct. 2018. ISBN 978-149-195-766-0.

MICRO FOCUS. **Silk Central**. [S.l.: s.n.].

<https://www.microfocus.com/en-us/products/silk-central/overview>. Accessed: 2020-06.

MONI, Robert. **Reinforcement Learning algorithms — an intuitive overview**. [S.l.: s.n.], 2019. <https://medium.com/@SmartLabAI/>. Accessed: 2020-06.

NOGUCHI, T. *et al.* History-Based Test Case Prioritization for Black Box Testing Using Ant Colony Optimization. *In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. [S.l.: s.n.], 2015. P. 1–2.

NUMFOCUS. **pandas - Python Data Analysis Library**. [S.l.: s.n.], 2015. <https://pandas.pydata.org>. Accessed: 2020-07.

PIVOTAL TRACKER. **Why agile development is the right choice for your team**. [S.l.: s.n.], 2020. <https://www.pivotaltracker.com/agile/what-is-agile-project-management>. Accessed: 2020-07.

PRESS, Gil. **Cleaning Big Data: most time-consuming, least enjoyable data Science task**. [S.l.: s.n.], 2016. <https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says>. Accessed: 2020-07.

REITZ, Kenneth. **Requests: HTTP for Humans™**. [S.l.: s.n.]. <https://requests.readthedocs.io/en/master/>. Accessed: 2020-06.

RENAULT Software Labs. [S.l.: s.n.], 2020. <https://group.renault.com/groupe/implantations/software-labs-toulouse>. Accessed: 2020-06.

RENAULT–NISSAN–MITSUBISHI Alliance. [S.l.: s.n.], 2020. <https://www.alliance-2022.com>. Accessed: 2020-06.

RENÇBEROĞLU, Emre. **Through Medium platform. Fundamental Techniques of Feature Engineering for Machine Learning**. [S.l.: s.n.], 2019. <https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114>. Accessed: 2020-07.

ROTHERMEL, G. *et al.* Prioritizing test cases for regression testing. **IEEE Transactions on Software Engineering**, v. 27, n. 10, p. 929–948, 2001.

RUTH, M. *et al.* Towards Automatic Regression Test Selection for Web Services. *In: 31ST Annual International Computer Software and Applications Conference (COMPSAC 2007)*. [S.l.: s.n.], 2007. P. 729–736.

SCIKIT-LEARN: Machine Learning in Python. [S.l.: s.n.]. <https://scikit-learn.org>. Accessed: 2020-07.

SPIEKER, Helge *et al.* Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. **CoRR**, abs/1811.04122, 2018. arXiv: 1811.04122. Available at: <http://arxiv.org/abs/1811.04122>.

W3C. **Extensible Markup Language (XML)**. [S.l.: s.n.]. <https://www.w3.org/XML/>. Accessed: 2020-06.

ZHAO, Y. *et al.* The impact of continuous integration on other software development practices: A large-scale empirical study. *In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. [S.l.: s.n.], 2017. P. 60–71.

APPENDIX A – JIRA'S DATABASE FEATURES

	Feature	Type	
BUGS	1	Key	string
	2	Summary	text
	3	Status	string
	4	Priority	string
	5	Description	text
	6	Domain	json
	7	Created Date	date time
	8	Resolution	string
	9	Resolution Date	date time
	10	Linked Issue	json
	11	Created Dump	boolean
	12	Updated dump	boolean
	13	Dump date	date time

	Feature	Type	
TE	1	Key	string
	2	Summary	text
	3	Status	string
	4	Description	text
	5	Fix Version	json
	6	Created Date	date time
	7	Source code data	string
	8	Test Plans	array
	9	Git Project	string
	10	Job Name	string
	11	Runner	string
	12	Branch	string
	13	Short ID	string
	14	Title	string
	15	Author Email	string
	16	Authored Date	date time
	17	Committer Email	string
	18	Committed Date	date time
	19	id	int
	20	name	string
	21	is_shared	text
	22	Description status	string
	23	access_level	string
	24	architecture	string
	25	tags	json
	26	revision	string
	27	Job Duration	float
	28	Created Dump	boolean
	29	Updated dump	boolean
	30	Dump date	date time

	Feature	Type	
CAT	1	Key	string
	2	Summary	text
	3	Status	string
	4	Description	text
	5	Created Date	date time
	6	Resolution	text
	7	Domain	json
	8	Issue Links	text
	9	Type	string
	10	Sub Type	string
	11	Test Level	string
	12	Environnement	text
	13	Automation	string
	14	OS	string
	15	Item Under Test	string
	16	Programs	json
	17	Test Set	json
	18	Created Dump	boolean
	19	Updated dump	boolean
	20	Dump date	date time

	Feature	Type	
TRDM	1	TestExecKey	string
	2	Count Passed	int
	3	% Passed	float
	4	Count Failed	int
	5	% Failed	float
	6	Count Aborted	int
	7	% Aborted	float
	8	Count Blocked_Env	int
	9	% Blocked_Env	float
	10	Count Executing	int
	11	% Executing	float
	12	Count Blocked_Dep	int
	13	% Blocked_Dep	float
	14	Count ToDo	int
	15	% ToDo	float
	16	Created Dump	boolean
	17	Updated dump	boolean
	18	Dump date	date time

	Feature	Type	
RUN	1	id	int
	2	testExecKey	string
	3	Key	string
	4	status	string
	5	type	string
	6	start	date time
	7	finish	date time
	8	executedBy	string
	9	defects	json
	10	evidences	text
	11	results	json
	12	steps	json
	13	assignee	string
	14	testEnvironments	text
	15	testIssueFields	json
	16	Summary	string
	17	comment	text
	18	Created Dump	boolean
	19	Updated dump	boolean
	20	Dump date	date time

Source: BRZEZINSKI MEYER, Maria Laura (2020).